TURBO PASCAL I.

Petr Drlík

ÚVOD

Kto vám môže narobiť problémy? Len ten, kto vie, na čo ste citliví a na čo vás "chytí". A mám za to, že tak presne a priamo pod kožu ich vie sformulovať iba informatik. Prečo také odvážne tvrdenie? Ak skutočne, a nie len listovaním, dočítate túto knihu do konca a sem-tam-stále si niečo skúsite vyriešiť, verím, že ďalšie dôkazy uvádzať nie je potrebné.

Ale naspäť k meritu (jadierku) veci. Chrobáka a výzvu menom PROBLÉM, resp. Programovanie v pascale mi nasadil do hlavy pokušiteľ-vydavateľ. Pozná moju minulosť i súčasnosť, a tým aj slabosti. Dal mi lukratívnu ponuku (písomne zvečniť to, čo mažem z tabule na prednáškach už nejaký ten rok. Som mu vďačný za "výzvu na súboj". S jeho "Dvojkou" (Turbo pascal II.) sa však bojuje ťažko. Sú tam použité tie najkrajšie, najťažšie a najúčinnejšie programátorské zbrane. Mne zostali iba nižšie "zbrane krátkeho doletu". Pre niekoho sú to možno primitívne záležitosti a partie. Ale bez základov nie je možné stavať dom. V jednom prípade sa to dá – ak si postavíte (najlepšie na kľúč) vzdušný zámok. Potom ale pozor, aby nezafúkal silný vietor alebo nepršalo! Pre programovanie – riešenie problémov na počítači to platí dvojnásobne. Hodiny "relaxu" za počítačom a hľadania chýb zúčastnených strán (človek kontra počítač, či naopak) môžu byť pre okolie signálom hodnotenia osoby okupujúcej počítač:

1. Ticho! Veď ON rieši tak zložitý problém!!
2. Preboha! TENTO by mal radšej vypnúť počítač a ísť od toho preč!

Prajem vám, aby o vás ostatní hovorili iba parafrázy prvého variantu. A keď (zo začiatku) nie, nevadí! Skúste natiahnuť plachty fantázie, chytiť ten správny vietor, nerobiť si z problémov ešte väčšie problémy a mať dosť trpezlivosti, intuície, potrebných znalostí a "strojového" času.

Dostal som striktný pokyn (nie príkaz!), že programovanie by sa nemalo realizovať cez matematiku, ktorá vraj veľa ľudí nebaví. Pokúsim sa ho držať, no niekde mi to nedá, niektoré deformácie sú trvalé. Ale nebojte sa! Bude vám stačiť iba trošku logicky rozmýšľať, aj keď je to niekedy, resp. u niekoho problém, a sem-tam si oprášiť svoje nesmrteľné matematické vedomosti zo základnej školy. Myslím si, že kto sa dobrovoľne rozhodol čítať knihu s takýmto názvom, nebude mať žiadne výhrady.

Ďakujem všetkým, ktorých som využil ako "pokusných králikov" – mojim bývalým i súčasným študentom; všetkým, ktorí boli a sú pre mňa vzormi – mojim učiteľom i kolegom, zvlášť docentovi Hvoreckému a profesorovi Kasatkinovi; ako aj tým, ktorí mi zabezpečili pohodu a inšpiráciu pri písaní – manželke Ľube, synom Martinovi a Ľubošovi.

1 PROBLÉM A RIEŠENIE PROBLÉMU (POSTUP – ALGORITMUS – PROGRAM)

Ako začať?

Chceme sa naučiť riešiť problémy pomocou počítača[1], vedieť s ním komunikovať nielen ako bežní používatelia hotových produktov, ale aj ako tvorcovia (autori) "nových" riešení. Je zrejmé, že je málo toho, čo by už nebolo povedané či napísané. Ale nie je isté, či to bolo povedané a napísané tak, ako to dokážeme my. Do programov – zjednodušene povedané predpisov toho, čo má vykonávať počítač – vkladá každý programátor kúsok seba, spolu s klasickým postupom riešenia dodáva k nemu aj svoj estetický a podľa neho efektívny pohľad na spracovávanú problematiku. O tom, či je to pre iného používateľa estetické a efektívne, by sa dalo meditovať. Ale základom programovania je práve možnosť realizovať riešenie problému podľa svojich predstáv, dať riešeniu svoj vlastný "image" (čítaj "imidž" - preložiť zrejmé pojmy je obtiažne). Je to podobné umeniu i skutočnému životu (nie škole), kde platí, že keď dvaja robia to isté, nikdy to nie je to isté.

Programovanie má oproti mnohým iným činnostiam, ku ktorým nás "dobrovoľne-povinne" vedú v škole, obrovskú výhodu. Často nás núti poriadne naplno roztočiť svoje mozgové závity, rozmýšľať o tom, čo je podstatné a čo nie, čo má byť viditeľné a čo skryté, realizovať svoju vlastnú predstavu použitím známych prostriedkov. Vždy, keď sa človek do niečoho pustí, musí mať správnu motiváciu. Tou môže byť aj skutočnosť, že s počítačmi a informačnými technológiami sa už dnes stretnete prakticky všade. Nechcete dokázať prinútiť stroj – počítač, aby robil presne to, čo mu zadáte? Je to výzva, ktorej sa ťažko odoláva, ak chcete naozaj niečo dosiahnuť. Skúste to!

Prvé, čo potrebujeme, je zaviesť základné pojmy, vzťahy medzi nimi a metódy – t. j. spôsoby práce s nimi, skrátka a múdro povedané - *terminológiu*. Mali sme tu pojmy "problém", "vyriešenie problému", "informatika", "programovanie", "počítač", určite toho však budeme musieť zvládnuť viac.

1.1 Terminológia

Každý vedný odbor má svoje základné pojmy a metódy ich spracovania, hlavne však špecifikáciu toho, čo je predmetom jeho záujmu. Matematika sa zaoberá predovšetkým

[1] "Osobné počítače, ktoré sa dostali aj do domácností majú oproti ostatným členom domácnosti, kde dnes často hrajú významnú úlohu aj domáce zvieratá, isté prednosti: a) nehryzú, neškriabu, neštekajú, b) nerobia kôpky a mláčky, nepotrebujú venčiť, c) s kýmkoľvek sa ochotne hrajú najrôznejšie hry, d) narozdiel od ostatných členov domácnosti sa dajú hocikedy vypnúť, e) môžeme ich chovať bez akýchkoľvek problémov a sporov so susedmi aj v špecifických podmienkach panelákov, f) v prenosnom vydaní sú vhodné aj na rande do parku, či na dlhé cesty (ak nemáte na lietadlo) za poznaním." (I. Kopeček, J. Kučera: Programátorské poklesky. Mladá fronta, Praha 1989.)

kvantitatívnymi a priestorovými vzťahmi, fyzika skúmaním podstaty a spôsobov zmien energie, biológia existenciou a fungovaním biologických systémov a procesov... Informatika je v porovnaní s týmito klasickými prírodovednými disciplínami dieťaťom v plienkach. Kým matematika, fyzika, biológia... sa vyučujú stovky až tisíce rokov, informatika nemá ako vedná disciplína ani päťdesiat (a ako predmet vyučovania ešte menej). Prečo sa potom dostala do takej pozornosti spoločnosti, prečo by mal jej základy ovládať každý (stredoškolsky) vzdelaný človek?

Všimnime si, čo je dnes dôležité pre každého z nás. Okrem klasických poznatkov, zručností a návykov sa musíme dokázať orientovať v množstve najrôznejších informácií (aj dezinformácií), efektívne využívať rôzne databázy údajov, komunikovať pomocou najrozličnejších technických prostriedkov nielen v regionálnom, ale aj v celosvetovom rámci. Dnešná spoločnosť prežíva informačnú revolúciu; bez potrebných znalostí a prostriedkov nie je možné udržať krok so svetom nielen vo vede, ale ani v obchode, priemysle, politike... Kľúčom k úspechu by mohla byť práve znalosť informatiky zaoberajúcej sa spracovaním informácií z rôznych zorných uhlov.

Pri chápaní informatiky, jej poslania a cieľov dochádza niekedy k deformácii. Väčšina laikov redukuje informatiku a jej vyučovanie len na znalosť a zručnosť práce s počítačom, prípadne výpočtovou technikou. Je to veľmi podobné situácii, keď by sme matematiku a jej vyučovanie "povýšili" iba na počítanie. "Veď z toho, čo sme sa učili v matematike v škole, dnes potrebujem iba sčítať, odčítať, násobiť, niekedy deliť, najviac desatinné čísla a vedieť si vypočítať nejaké percentá," - tvrdí veľa ľudí. Potom by ale stačilo, keby sme sa na matematike učili iba prácu s kalkulačkou?! (*Úloha*: Zistite, kedy sa objavili prvé kalkulačky! Pozn. Vieme isto, že Robinson Crusoe ich ešte nemal k dispozícii.)[1]

Vyučovanie (učenie sa) matematiky však sleduje iný cieľ, ktorý má podstatne väčší význam pre rozvoj osobnosti: učí človeka koncepčne a logicky rozmýšľať, dávať si do súvislostí logické vzťahy medzi presne definovanými objektmi, dokázať riešiť úlohy teoretického, ale aj praktického charakteru. Podobné je to aj s vyučovaním (učením sa) informatiky. Výpočtová technika je iba prostriedkom, ktorý umožňuje efektívne realizovať informatické poznatky. Informatika aj výpočtová technika sa veľmi úzko ovplyvňujú, spolu vytvárajú jeden celok a vzájomne sa rýchlo posúvajú dopredu. Informatika okrem poznatkov z hardware a software výpočtovej techniky dáva teoretické a praktické poznatky o tom, ako spracovávať, uchovávať, organizovať, prenášať a vyhľadávať informácie ľubovoľného druhu, rozvíja tiež schopnosti efektívne využívať rôzne technické prostriedky pre spracovanie informácií. Navyše umožňuje rozvíjať systematické a koncepčné myslenie, využívať obmedzenú sadu nástrojov pre riešenie daného problému – teda riešiť problémy. A to je jeden z najdôležitejších výsledkov, ktoré nám vyučovanie (učenie sa) informatiky môže priniesť.

Dnes sa často objavuje pojem informačné technológie. Zjednodušene si ich môžeme predstaviť ako spôsoby, metódy a prostriedky spracovania informácií najrôznejšieho druhu s využitím modernej techniky a spôsoby prenosu týchto informácií na veľké vzdialenosti. Informačné technológie sú istým spôsobom akýmsi praktickým zastrešením čiastkových poznatkov informatiky a výpočtovej techniky. Vzhľadom na iné ciele sa im však v tejto knihe venovať nebudeme.

[1] Keby ste náhodou nenašli príslušný zdroj informácií: Kalkulačka v dnešnom ponímaní sa objavila okolo r. 1970, kedy sa začali vyrábať LCD-displeje - zobrazovače, ktoré umožnili vybaviť kalkulačky menej energeticky náročným realizátorom a zobrazovačom činnosti.

1.2 Mám problém, čo s ním?

Kto dnes nemá problémy? Bolo by však dobré, keď sa chceme zaoberať riešením problémov, presne si používané pojmy charakterizovať. Pokúsme sa o to. Pripomínam však, že charakteristiky pojmov sú iba opisné, ich presné "definovanie" je dosť problematické.

Problém je stav, v ktorom jestvuje rozdiel medzi tým, čo v danom momente poznáme (vieme, máme), a tým, čo potrebujeme. Inak povedané: disproporcia medzi možnosťami a cieľom. Problém je viazaný na jeho "majiteľa" (pre iného to nemusí byť problém, ale nezmysel) a na isté problémové prostredie (citové, finančné, školské... problémy). V našom prípade sa zameriame na hľadanie riešení problémov z oblasti spracovania informácií.

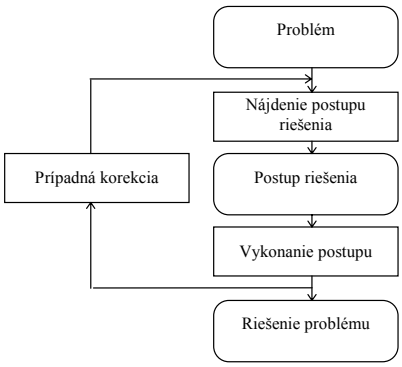
Riešenie problému chápeme vo význame splnenia cieľa, t. j. odstránenia disproporcie. Potom môžeme schematicky znázorniť postup vyriešenia problému ako na obrázku.

V ováloch sú uvedené stavy, ktoré pri odstránení problému potrebujeme absolvovať, v obdĺžnikoch činnosti, ktoré musíme pre zmeny od problému k riešeniu vykonať. Šípky naznačujú postupnosť zmien stavov a činností.

Všimnime si činnosti *Nájdenie postupu* a *Vykonanie postupu*.

Činnosť *Nájdenie postupu* je činnosť tvorivá – je potrebné aktívne premyslieť jej podrobnosti a postupné kroky. Vymyslením postupu riešenia je možné poveriť iba človeka, zatiaľ (pre človeka našťastie) nejestvuje také technické zariadenie, ktoré by bolo schopné tvorivo myslieť.

Inak je to s činnosťou *Vykonanie postupu*: Táto je rutinnou činnosťou v prípade, že postup je už daný alebo známy. Jej vykonaním môžeme poveriť niekoho, kto je schopný daný postup realizovať – či už človeka, alebo stroj. Nemusí rozmýšľať, stačí, ak bude daný postup presne realizovať. Takémuto zariadeniu sa hovorí *procesor*. (Pozor! Procesor je v tomto význame oveľa širší pojem ako zaužívané označenie pre technickú jednotku – súčasť hardware počítačov.) Ak uvažujeme, že budeme riešiť problémy z oblasti spracovania informácií, potom procesorom spôsobilým vykonávať nami vytvorené postupy môže byť buď človek, alebo počítač.



1.3 Algoritmus

Je samozrejmé, že postup musí byť určený pre nemysliace zariadenie, ktoré vôbec nevie, čo má byť výsledkom jeho realizácie. Preto máme isté obmedzenia, na ktoré musíme pri formulácii postupu myslieť. Postupu určenému pre nemysliace zariadenie (procesor) hovoríme **algoritmus**.

Algoritmus[1] je elementárnym pojmom informatiky. Elementárnym v tom zmysle, že nie je možné ho opísať pomocou ešte elementárnejších pojmov, podobne ako v geometrii bod. (Poznámka: Nie je to až tak úplne pravda, ale pre naše potreby toto priblíženie postačuje.)

Preto algoritmus definujeme iba opisne. Tu je jeden z možných opisov:

Algoritmus je postup, ktorého realizáciou získame zo zadaných vstupných údajov po konečnom počte činností v konečnom čase správne výsledky.

Pre upresnenie toho, či je postup algoritmom, sa používajú doplňujúce vlastnosti, ktoré znovu môžeme uviesť iba opisne:

Vlastnosti algoritmu

P1. Elementárnosť – Postup je zložený z činností, ktoré sú pre realizátora elementárne, zrozumiteľné.

P2. Determinovanosť – Postup je zostavený tak, že je v každom momente jeho vykonávania jednoznačne určené, aká činnosť má nasledovať, alebo či sa už postup skončil.

P3. Rezultatívnosť – Postup dáva pre rovnaké vstupné údaje vždy rovnaké výsledky (ak skončí).

P4. Konečnosť – Postup skončí vždy v konečnom čase a po vykonaní konečného počtu činností.

počtu činností.

P5. Hromadnosť – Postup je aplikovateľný na celú triedu prípustných vstupných údajov.

P6. Efektívnosť – Postup sa uskutočňuje v čo najkratšom čase a s využitím čo najmenšieho počtu prostriedkov.

Týchto "šesť pé" sa nemusí na prvé prečítanie zdať dôležitými, ale vzťahujú sa predovšetkým na to, aby postup mohlo realizovať nemysliace zariadenie. Toto si nevie uvedomiť, že postup sa vykonáva podozrivo dlho, nevie experimentovať, nemá žiadne skúsenosti, neučí sa z chýb (cudzích ani vlastných). Vlastnosť *P6 (Efektívnosť)* je viac-menej relatívna. Najlepšie sa o tom môžeme presvedčiť pri rôznych algoritmoch triedenia. Hľadanie efektívnych algoritmov je jednou z najdôležitejších oblastí informatiky.

Venujme sa teraz podrobnejšiemu opisu vlastností spolu s uvedením príkladov postupov, ktoré sú nám známe zo života a nie sú (v používanom tvare) algoritmami.

P1. ELEMENTÁRNOSŤ

Jeden postup môže byť zapísaný rôznymi spôsobmi, a tak (ne-)zrozumiteľný pre prípadných realizátorov. Čo je elementárnou činnosťou pre jedného, nemusí byť pre iného.

Príklad 1.1. Už deti na prvom stupni základnej školy vedia násobiť. Ale skúste im povedať: "Zistite šiestu mocninu čísla 2!" Asi nebudú vedieť reagovať. Keď im zadáme úlohu v tvare: "Zistite výsledok súčinu 2.2.2.2.2.2!", bude to pre nich hračkou. Zistenie mocniny pre nich totiž nie je elementárnou činnosťou.

[1] Meno algoritmus má zaujímavý pôvod: Vzniklo latinským prepisom mena arabského matematika Abú Jáfar Mohamed Ibn Músa al-Chworezmího, ktorý okolo r. 825 napísal knihu o riešení rovníc v desiatkovej číselnej sústave s názvom "Kitab al jabr w'al-mugabala" (podčiarknuté vám iste, a nie náhodou, pripomína slovo "algebra"). Kniha v latinčine začínala vetou "Algorithmi dici... ", t. j. "Al-Chworezmí hovorí... " (Pri preklade z arabských "klikihákov" nemusí meno byť vždy rovnaké.)

Príklad 1.2. Pre murárskeho učňa je iste elementárnou činnosťou pri murovaní "zarobenie kalfasu dobre mastnej malty". Obávam sa, že pre väčšinu z vás (vrátane mňa) to však bude značný problém.

Príklad 1.3. Mám veľmi rád kuchárske recepty, hlavne ich produkty. Realizácia je však pre mňa často problémom. V jednom z receptov je napríklad elementárnou činnosťou "Zarob bešamel", v inom "Meľ dva dni staré rožky", v ďalšom "Priprav marinádu"!? A čo tak "Pridaj štipku soli" alebo "Pridaj do toho dve celé vajcia"? Samé problémy s elementárnosťou, často však subjektívne.

Človek má jednu výhodu – dokáže sa učiť a vytvárať si stále zložitejšie a zložitejšie elementárne činnosti, ktoré už potom kombinuje do ešte zložitejších postupov. Takúto vlastnosť však nemajú počítače, majú iba obmedzenú sadu elementárnych činností z oblasti spracovania informácií.

P2. DETERMINOVANOSŤ

Zdanlivo nepochopiteľná vlastnosť – presné určovanie poradia činností, pomáhať si umelými odkazmi, čím pokračovať ako ďalšou činnosťou (príklad 2.1). Ľudia podvedome či cieľavedome dokážu chápať postupy, v ktorých nie je zvýraznená riadiaca zložka, teda poradie vykonávaných čiností, prípadne, čo a ako dlho opakovať. Pre nemysliace zariadenie je však uvdenie poradia nevyhnutnosťou.

Príklad 2.1. Popíšme si postup pri prechode cez ulicu. Nie je to veľmi jednoduchá úloha v prípade, že by sme chceli uvažovať všetky možnosti. Neuvažujme teraz svetelné križovatky, a pre istotu ani fakt, že by vodiči mali dať chodcovi na "zebre" prednosť. (Naivne veriť predpisom sa v tomto prípade dá iba raz.)

Tu je jeden pokus o zápis postupu:

1. Pozri doľava.
2. Ak ide auto, opakuj krok 1.
3. Prejdi do polovice ulice.
4. Pozri doprava.
5. Ak ide auto, opakuj krok 4.
6. Prejdi na druhú stranu ulice.

Postup by bolo možné zapísať aj ináč, ale je tu zvýraznené použitie špeciálnych príkazov na riadenie postupu "opakuj krok...", ktorým sa v programovaní ťažko vyhneme. Navyše postup určite nespĺňa všetky podmienky zrozumiteľnosti, ba aj bezpečnosti. Čo ak sa blíži autobus? Skúste napísať lepší postup na bezpečný prechod cez ulicu.

P3. REZULTATÍVNOSŤ

Znovu zdanlivo nezmyselná vlastnosť - ako môže byť výsledkom viacerých realizácií postupu s rovnakými vstupnými údajmi rôzny výsledok? Ale pozor! Nie je to v bežnom živote až tak nezvyčajné! Veď nie nadarmo sa používa výrok "Keď robia dvaja to isté, nemusí to byť to isté." Ešte neveríte?

Príklad 3.1. Skúste si vybrať ľubovoľný kuchársky recept a urobiť jedlo presne tej istej chute. Vždy keď sa v ňom objavia nejaké približné údaje, napr. ako je potrebné jedlo variť či piecť a pod., nebude to to isté. Aj najlepšej gazdinej sa z času na čas niečo pripáli, niekedy je slané,

inokedy plané... Ovplyvňujú to, samozrejme, tiež subjektívne okolnosti, ale ani ten plyn, na ktorom sa jedlo pripravuje, nemusí byť vždy rovnako silný.

Príklad 3.2. Častá situácia v škole: Začiatok prestávky po písomke z matematiky, a tým aj začiatok zisťovania, komu ako ktorý príklad vyšiel. Mali by predsa vyjsť rovnako! Ale je vopred isté, že to tak nebude, často nie sú ani dve riešenia v celej triede (ak sa nedá opisovať) rovnaké. Podobne je to aj v živote. Aj keď ste si istí, že sa správate rovnako ako minule, efekt (často aj afekt) môže byť úplne iný.

Vo všetkých týchto prípadoch je realizácia postupu závislá na subjektívnych aj objektívnych podmienkach, ktoré nedokážeme ovplyvniť. Vieme však zo skúsenosti, že s tým musíme počítať. O to je život zaujímavejší.[1]

P4. KONEČNOSŤ

Skúsenosti nám umožňujú prerušiť nejaký postup v momente, kedy vidíme, že nevedie k požadovaným výsledkom. Skúsenosti však od nemysliaceho zariadenia nevyžadujeme, preto zostáva na nás formulovať postup riešenia problému tak, aby určite skončil. Inak by mohli nastať situácie podobné uvedeným príkladom:

Príklad 4.1. Časť zo zaručeného návodu na nájdenie pokladu: "Poklad nájdeš, keď pôjdeš... Kop na tom mieste, kým nenarazíš na poklad."

Máte smolu, ak poklad už niekto vybral alebo tam nikdy nebol. Nemysliace zariadenie by sa tým asi zamestnávalo do konca svojej kariéry.

Príklad 4.2. Návod bez zmyslu, ale môže byť podobný niektorým vami vytvoreným programom:

1. Mysli si číslo.
2. Pokiaľ sa nebude rovnať 1, odčítaj od neho 2.

Netreba ani zvýrazňovať, že v prípade zadania desatinného, záporného alebo párneho čísla bude nemysliace zariadenie opakovane odčítať jednotku "pomerne dlho".

Je aj iný druh "nekonečnosti". Niektoré metódy, ktoré sú teoreticky konečné a algoritmicky správne, môžu trvať tak dlho, že sú vlastne prakticky nerealizovateľné:

Príklad 4.3. Majme zistiť počet zrniek piesku, ktoré sú na pláži.

Môžeme použiť najmenej dva postupy:

I. Urobiť si čisté miesto a zrnko po zrnku za neustáleho počítania tam prenášať.

II. Spočítať približný počet zrniek na lopate a prehádzať ňou celú pláž, počet lopát si pamätať.

Iste by ste našli aj lepšie riešenia. Väčšinou je potrebné vhodne zjednodušiť problém a riešiť ho iba približne - tak, že uvažujeme s istou chybou. Podobných úloh je možné nájsť viac.

P5. HROMADNOSŤ

Táto vlastnosť už patrí skôr k užitočným ako nevyhnutným. Existujú aj jednoúčelové postupy, ktoré nemajú premenlivé vstupné údaje, a pritom sú algoritmami. Takýmito príkladmi môžu byť nastaviteľné, ale po voľbe už pevné, programy pre automatickú práčku, výrobu súčiastok na NCR strojoch a pod.

[1] Ako by vám vyhovovalo, keby platila epizóda zo Židovských anekdot Karla Poláčka:
Pán Kohn stretne na ulici neznámeho pána, ktorý ho uctivo pozdraví: "Dobrý deň, pán Kohn!" "Prepáčte, ale ja vás nepoznám. Odkiaľ ma vy poznáte?" - čuduje sa pán Kohn. "Ja som si vás vypočítal!"

Príklad 5.1. Naučiť niekoho násobiť dve prirodzené čísla je niečo iné ako naučiť ho, koľko je 2x2. Cieľom je naučiť postup, ako medzi sebou násobiť ľubovoľné dve prirodzené čísla. A tento postup môžeme zovšeobecniť na násobenie dvoch ľubovoľných celých, racionálnych, ba dokonca až reálnych čísiel.

Hromadnosť postupu je teda skrytá v tom, že postup pripúšťa premenlivé vstupné údaje a umožňuje nám riešiť úlohy podobného typu.

P6. EFEKTÍVNOSŤ

Efektívnosť algoritmu je viac-menej doplnková, niekedy však veľmi potrebná vlastnosť. Zvlášť pri veľmi veľkom počte spracovávaných údajov, ako aj tam, kde potrebujeme veľký počet zmien spracovávaných informácií. Často je cieľom najskôr zostaviť hocijaký, ale funkčný algoritmus, potom ho v prípade potreby a s lepšou znalosťou problému vylepšovať. Ako kritériá efektívnosti slúžia časová a pamäťová zložitosť algoritmu, ktorým sa ešte budeme venovať. Uveďme si dva príklady pokusu o efektívnosť algoritmu (postupu).

Príklad 6.1. Prechod cez ulicu. Túto úlohu sme už mali. Ale máme aj jedno pre zápis určite efektívne riešenie, ktoré vytvoril žiak 5. triedy základnej školy (dnes prezident slušne prosperujúcej počítačovej firmy):

1. Choď cez ulicu tak, aby ťa nič nezrazilo.

Je jasné, že za toto riešenie získal plný počet bodov, aj keď je určené skôr pre človeka, ktorý má už svoje skúsenosti s dopravnou situáciou. Efektivita zápisu a skoro geniálny nápad sú však zrejmé.

Príklad 6.2. Na vyučovaní matematiky dostali asi 10-roční žiaci úlohu spočítať prvých 50 prirodzených čísiel. (Učiteľ si chcel v kľude prečítať noviny, a keby nestihol, zvýši počet na 100.)

Žiaci postupovali väčšinou podľa jeho predpokladov: sčítali postupne 1+2+3+4+5+... a trvalo im to úmerne ich "sčítacím" schopnostiam.

Našiel sa však žiak, nazvime ho malý Gauss[1], ktorý prevrátil plány učiteľa naruby. Nekonal hneď, ale rozmýšľal: 1+50=51, 2+49=51, 3+48=51,... , 25+26=51 a takýchto dvojíc je polovica z počtu čísiel, teda 25. Vynásobenie 25x51=1275 bolo otázkou okamihu.

Prichádza hneď do úvahy zovšeobecnenie riešenia tak, aby bol postup hromadný, teda riešil problém pre ľubovoľný počet prvých prirodzených čísel N. Tu je:

```
SÚČET = N/2 x (N+1)
```

Jeho efektívnosť je optimálna – nech je počet prirodzených čísiel ľubovoľne veľký, pre zistenie súčtu stačí jedno sčítanie, jedno násobenie a jedno delenie. Nie všetky problémy, dokonca ani matematické, sa však dajú previesť do tvaru vzorcov.

Vidíme, že je nevyhnutné, aby sme si presnejšie určili, čo má byť vstupom a čo výsledkom algoritmu. Preto sa dohodnime, že zadanie algoritmu budeme zapisovať takto:

```
{VST: vstupné podmienky}
?
{VÝS: výstupné podmienky}
```

[1] *Karl Friedrich Gauss (1777-1855) - jeden z najvýznamnejších prírodovedcov 19. storočia, nemecký matematik, fyzik, geofyzik a astronóm, univerzitný profesor a riaditeľ hvezdárne v Göttingene.*

kde na začiatok doplníme vstupné podmienky - vzťahy, ktoré platia na začiatku realizácie algoritmu, a na konci určíme výstupné podmienky - čo má byť výsledkom realizácie algoritmu.

Na záver tejto kapitoly si bez veľkého vysvetľovania uveďme ešte niektoré pojmy, s ktorými budeme v ďalšom texte pracovať.

Algoritmizácia

Algoritmizáciou rozumieme schopnosť aktívne vytvárať algoritmy určené pre nemysliace zariadenie. Je nevyhnutnou súčasťou schopnosti programovať na počítačoch. Tak ako iba teoretická znalosť jazyka nestačí, aby človek mohol komunikovať, tak ani zvládnutie iba prostriedkov algoritmického či programovacieho jazyka "od A po Z" nestačí na to, aby človek vedel vytvárať efektívne a správne algoritmy. Je na to potrebná aj dôkladná znalosť problémového prostredia, skúsenosť s formulovaním algoritmov a schopnosť využiť obmedzené prostriedky konkrétneho jazyka, resp. techniky.

Čo je to správny algoritmus? Môžeme sa dohodnúť na týchto termínoch:

- Algoritmus nazývame *čiastočne správny*, ak v prípade, že skončí, dáva vždy správne výsledky.
- Algoritmus nazývame *konečný*, ak skončí v konečnom čase pre ľubovoľné vstupné údaje.
- Algoritmus nazývame *správny*, ak je čiastočne správny a konečný.

Algoritmus a program

Aký je vzťah medzi algoritmom a programom? Pod *programom* chápeme algoritmus napísaný v programovacom jazyku. Oproti algoritmu obsahuje navyše ďalšie inštrukcie pre počítač, predovšetkým vzťahujúce sa k určeniu typov spracovávaných údajov, využívaniu hardware, prípadne ďalšiemu softwaru počítača. Veľa programov spolupracuje s rôznymi doplnkovými súbormi, ktoré obsahujú pomocné údaje pre činnosť programu, napr. obrázky, hudbu, tabuľky výsledkov... Preto je pri tvorbe zložitejších celkov vhodnejšie používať pojem *programový produkt*.

Programovanie

Programovanie je konštruktívna myšlienková, ale aj praktická činnosť, kedy vytvárame nové programové produkty realizovateľné na počítači. Programovaním sa učíme predovšetkým myslieť, organizovať svoje myšlienky a dokázať ich realizáciou poveriť počítač. A nielen to. Poznávame tým lepšie aj samého seba ako tvora nedokonalého a s radosťou rozvíjajúceho svoje chyby až k dokonalosti. Reálne programovanie na počítačoch zahŕňa v sebe viacero rovnocenných súčastí, bez vhodného skĺbenia ktorých nebýva výsledok oslňujúci.

Vytvorenie programu pozostáva z týchto činností:

- Algoritmizácia daného problému – určenie vstupných a výstupných podmienok.
- Vytvorenie programu (programového produktu) a vhodnej programovej dokumentácie.
- Zapísanie a odladenie programu priamo na počítači.

V mnohých prípadoch navonok skúsení programátori postupujú pri programovaní systémom "sadnem a píšem". Neberte si z nich príklad, pretože skôr či neskôr sa stratíte v záplave činností, ktoré je potrebné mať jednak dobre premyslené, jednak rutinne zvládnuté. A pri tvorbe zložitejších produktov je nevyhnutná dôkladná algoritmická a systémová príprava, ktorá jediná môže minimalizovať čas strávený pri počítači a vyhnúť sa nekonečnému prepracovávaniu zdanlivo už hotových častí.

2. JAZYK, ALGORITMICKÝ JAZYK, PROGRAMOVACÍ JAZYK

Ak máme s niekým komunikovať, potrebujeme zodpovedajúci prostriedok dorozumievania – jazyk. Jazyk je podľa J. Mistríka (Jazyk a reč, Mladé letá, Bratislava 1984) "súhrn pravidiel, na základe ktorých vzniká reč". Pôvodne bol jazyk iba prostriedkom komunikácie medzi ľuďmi[61], dnes už je aj prostriedkom komunikácie medzi človekom a strojom. Pozrime sa na jazyk tohto typu podrobnejšie.

Potrebujeme niekoho naučiť postupy – algoritmy. Pretože algoritmy sú postupy so špecifickými vlastnosťami, bolo by potrebné použiť aj špecifický jazyk. Jazyk určený pre zápis algoritmov nazývame algoritmický jazyk. Jazyky používané pri komunikácii medzi ľuďmi nevyhovujú z viacerých dôvodov:

a) Počet slov je neúmerne vysoký (napr. slovenčina pozná vyše 110.000 slov, angličtina takmer 800.000). Navyše sú tieto jazyky v neustálom vývoji, ročne pribúdajú desiatky nových slov.

b) V ľudských jazykoch existuje veľa výnimiek spôsobených historickým vývojom, zdanlivo nezmyselné príslovia (napr. "Lož má krátke nohy.", "Kúpil mačku vo vreci."), prirovnania, zaužívané slovné spojenia (napr. "to je babylon", "kocky sú hodené", "medvedia služba").

c) Existencia homoným (slov, ktoré majú viacero významov, napr. "strana", "koruna", "list", "dospievať", "mať"...) a synoným (rôznych slov, ktoré majú rovnaký význam, napr. "najmä" - "hlavne" - "predovšetkým" - "najskôr") môže spôsobovať nejasnosť vysvetlenia. (Obľúbené sú slovné hračky a tzv. dvoj- či en-zmysly.)

d) Niekedy nie je možné ani z kontextu odhadnúť, čo dané slovo vyjadruje. Napr. úryvok z náhodne vypočutého rozhovoru: "Jano išiel do mesta. Mesiac sa neukázal." (cit. z výbornej knižky Hvorecký, Kelemen: Algoritmizácia) Je slovo "Mesiac" príslovkovým určením času alebo podmetom druhej vety? Môžeme iba odhadovať podľa znalosti veci (Jana) alebo hlasovať. Podobné je to s obľúbenou otázkou majora Teraskyho: "Čím je vojak?" (Vyberte si z možností "Obrancom vlasti" či "Lyžicou".)

e) Prirodzený jazyk obsahuje veľa prvkov a konštrukcií, ktoré sú pri formulovaní postupov zbytočné, napr. nič nehovoriace debaty a zdvorilostné otázky ("Už ani to počasie nie je, čo bývalo." alebo "Ako sa máš?" – aj keď mi je to úplne ukradnuté, čo sa však nehovorí, a pod.)

Tieto dôvody viedli k potrebe vytvoriť formálne jazyky – umelo vytvorené a špeciálne určené pre zápis algoritmov. Podobne ako sa nepodarilo ľudstvu dohovoriť sa na jednotnom jazyku, ani pri algoritmických jazykoch nedošlo k dohode a vo všeobecnosti sa používa viacero z nich. Najčastejšie sú:

A. *Grafické* algoritmické jazyky – napr. vývojové diagramy, rôzne typy štruktúrogramov,

B. *Lineárne* algoritmické jazyky – napr. slovný zápis v národnom jazyku, programovací jazyk.

Algoritmický jazyk by mal svojou konštrukciou napomáhať splneniu vlastností algoritmu. Preto v jestvujúcich algoritmických jazykoch sú dve zvýraznené zložky – *operačná a riadiaca*.

Operačná zložka

Obsahuje sadu prostriedkov, ktoré umožňujú spracovávať údaje – elementárne činnosti, ktoré dokáže procesor vykonávať. Pretože naším cieľom je prechod k programovaniu, ako základ budú

[1] Na svete existuje 6528 známych, aj keď niekedy už nepoužívaných jazykov. Najviac ľudí hovorí čínsky – 726 miliónov, ďalej nasledujú angličtina, španielčina, hinština, arabčina, portugalčina,... , ruština je ôsma, nemčina desiata, francúzština jedenásta. Svetovým dorozumievacím jazykom je angličtina, ktorou sa bude vedieť dohovoriť okolo roku 2000 každý štvrtý človek na svete. (údaje z r. 1997)

pre nás slúžiť elementárne činnosti, ktoré sa používajú pri programovaní. Základnými činnosťami sú *príkazy* a *podmienky*.

Príkazy sú vety jazyka, ktoré prikazujú procesoru vykonať isté, presne stanovené činnosti. Pre začiatok vystačíme s príkazmi vstupu, výstupu a priradenia. Tieto príkazy musia spracúvať nejaké objekty. V programovaní sú nimi premenné, konštanty a výrazy.

Premenná je objekt, ktorý obsahuje počas realizácie algoritmu konkrétnu hodnotu[1] presne stanoveného typu (napr. celé číslo, reálne číslo, reťazec znakov...).

Konštanta je objekt, ktorý nadobúda počas celej realizácie algoritmu jedinú konkrétnu hodnotu príslušného typu. Je to obdoba konštánt známych z matematiky, napr. π, e, ale aj z fyziky: g, c, κ, ε, μ.

Výraz je predpis, ktorý obsahuje konštanty, premenné a spôsob ich spracovania pomocou operácií a funkcií podobných tým, ktoré poznáme z matematiky. Jeho výsledkom je hodnota príslušného typu, ktorá vznikne po vykonaní vo výraze naznačeného spracovania.

Príkaz priradenia má tvar:

```
p:= v
```

kde *p* je meno premennej, *v* je výraz. Vykonaním príkazu nadobudne hodnota premennej *p* hodnotu výrazu umiestneného na pravej strane priradenia. (Často sa tento príkaz pletie s rovnosťou známou z matematiky, tá sa v programovaní využíva v podmienkach.)

Meno premennej budeme zapisovať tak, že musí začínať písmenom (anglickej abecedy "A" až "Z", resp. "a" až "z" - bez "príkras" – dĺžňov, mäkčeňov...) a môže obsahovať ľubovoľný počet písmen a číslic. Je vhodné voliť také mená premenných, ktoré napovedia čitateľovi, aký má premenná význam. Malé a veľké písmena nerozlišujeme, ale je vhodné, ak si "dôležité" premenné budeme označovať veľkými, pomocné malými písmenami. Takže menami premenných môžu byť A, B, C, podiel, b1, pomoc, alfa... , ale nie 1A, c$, žvacheľ a pod.

Riadiaca zložka

V algoritmickom jazyku musí byť presne stanovené poradie vykonávania jednotlivých činností. Je to potrebné preto, aby realizátor (procesor) nemusel uvažovať, čo má kedy vykonať. Oproti prirodzenému jazyku sa preto do algoritmu vkladajú riadiace príkazy a činnosti, ktoré určujú presnú postupnosť vykonávania jednotlivých činností. Časom sa vyvinuli najefektívnejšie prostriedky pre organizovanie postupnosti vykonávania činností známe ako základné algoritmické konštrukcie. Preto sa im venujme podrobnejšie spolu s uvedením príkladu konkrétneho algoritmického jazyka.

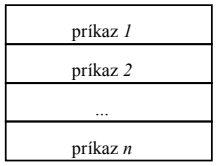
[1] Premenná v informatike a v matematike majú odlišný význam. V matematike je premenná chápaná ako symbol, t. j. zástupca celej triedy hodnôt určitého typu, napr. celých čísiel, nemusí to byť žiadna konkrétna hodnota. V informatike (presnejšie v programovaní) je premenná pamäťové miesto príslušnej veľkosti, ktoré vždy obsahuje istú hodnotu z určeného typu údajov. Je to teda vždy momentálna hodnota, ktorá je pod menom premennej uložená.

3. ZÁKLADNÉ ALGORITMICKÉ KONŠTRUKCIE A ICH ZÁPIS POMOCOU ŠTRUKTÚROGRAMOV

Každý postup sa dá vo všeobecnosti rozložiť na niekoľko za sebou idúcich, prípadne do seba vložených činností. Z hľadiska postupu vykonania môžeme rozložiť riešenie na tieto základné algoritmické konštrukcie:

– *Postupnosť* príkazov (činností),
– *Vetvenie* v závislosti od splnenia istej podmienky,
– *Cyklus* ako viacnásobné opakovanie istej činnosti.

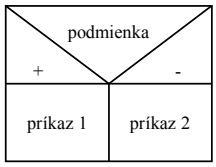
Na ukážku zápisu algoritmov v algoritmickom jazyku použijeme tzv. ***štruktúrogramy***. Nie sú (u nás) veľmi často používané, čo je škoda, pretože ich prepis do pascalu je, narozdiel od anachronicky používaných vývojových diagramov, veľmi jednoduchý.



Sekvencia – postupnosť príkazov: Vyplní sa v poradí, v akom sú príkazy pod sebou zapísané.

Napr. schéma zobrazuje zápis sekvencie pre postupnosť príkazov 1., 2. ... n.

Vetvenie (alternatíva): V závislosti od splnenia podmienky sa postup vetví na rôzne prípady. Ak je podmienka splnená (+), pokračuje sa plnením príkazu (činnosti) 1. , v opačnom prípade (-) sa pokračuje vykonaním príkazu 2.



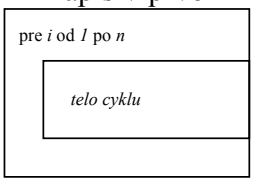
Ak potrebujeme vetviť postup na viacero rôznych riešení v závislosti od podmienky, vkladáme viacero alternatív "do seba".

Cyklus: Pri opakovaní nejakej činnosti musíme mať vyjasnené dve veci: čo sa má opakovať a dokedy sa to má opakovať. Činnosť, ktorá sa má opakovať, nazývame *telom cyklu*, podmienku, ktorá určuje dokedy sa bude telo cyklu opakovať, nazývame *podmienka cyklu*.

Vzťah medzi telom a podmienkou cyklu môže byť rôzny – podmienka môže predchádzať telu, cyklus sa môže vykonávať dovtedy, kým (ne-)bude splnená podmienka a pod. Najčastejšie sa používajú tieto typy cyklov:

1. Cyklus so známym počtom opakovaní: Telo cyklu sa opakuje vopred známy počet krát. Pre zisťovanie počtu už vykonaných opakovaní cyklu sa zavádza tzv. *riadiaca premenná cyklu*, ktorá nadobúda hodnoty od danej dolnej hranice po hornú hranicu (po "jednej").

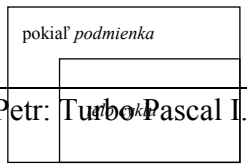
Zápis v prvom riadku naznačuje, že riadiacou premennou cyklu je *i* a cyklus sa bude opakovať pre *i* od 1 po N, teda N-krát.



Poznámka: Veľmi často sa používa práve označenie riadiacej premennej *i* alebo *j*. Je to tiež "anachronizmus" (niečo, čo sa vžilo v minulosti a dnes sa používa zo zotrvačnosti) – prvý vyšší programovací jazyk fortran mal nedeklarované premenné (tie, ktoré neboli popísané na začiatku programu) automaticky celočíselné, ak názov začínal písmenami "I", "J".

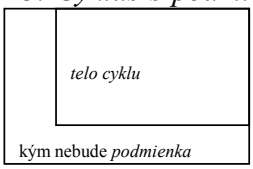
2. Cyklus s podmienkou na začiatku: Najčastejšie sa volí cyklus (nazvime ho "opatrný"), kedy sa telo cyklu opakuje, pokiaľ platí podmienka cyklu.

Podmienkou cyklu je tvrdenie, o ktorom dokáže procesor (ten, kto má algoritmus vykonávať) rozhodnúť, či je alebo nie je pravdivé. Tvar cyklu



naznačuje, že najskôr sa kontroluje splnenie podmienky. Ak je splnená, vykoná sa telo cyklu, a potom sa znovu kontroluje splnenie podmienky. V momente, keď prvýkrát podmienka cyklu neplatí, telo cyklu sa vynechá a pokračuje sa v plnení príkazov nasledujúcich za cyklom.

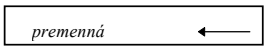
3. Cyklus s podmienkou na konci: Je prakticky opačný ako cyklus s podmienkou na začiatku - najskôr sa vykoná telo cyklu. Potom sa zisťuje splnenie podmienky cyklu. Ak je splnená, vykonávanie cyklu sa ukončí, v opačnom prípade sa riadenie procesu znovu vráti na vykonávanie tela cyklu. (Tento cyklus môžeme označiť ako "neopatrný": najskôr sa niečo vykoná, potom sa rozhoduje, či to bolo dobre.)



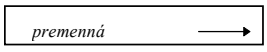
To sú dostačujúce prostriedky pre zápis ľubovoľného algoritmu – postupu. Ešte je potrebné okrem tejto riadiacej zložky algoritmického jazyka opísať **operačnú zložku**, t. j. aké elementárne činnosti (príkazy) vie procesor vykonávať.

Naším cieľom je pripravovať programy pre počítač. Preto budeme za základné činnosti algoritmického jazyka považovať tie, ktoré sú elementárne pre počítač. K nim patria:

Príkaz vstupu: Umožňuje zadať procesoru konkrétne hodnotu údajov, ktoré má spracovávať. Tieto hodnoty sa uložia do *premenných* (môžeme si ich predstaviť ako priehradky (skrine)) s pevne stanovenou veľkosťou. V každom momente vykonávania algoritmu by mala byť v priehradke - premennej nejaká konkrétna hodnota.

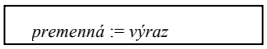


Príkaz výstupu: Umožňuje získať od procesora výsledky algoritmu alebo iné oznamy (napr. oznam o tom, že sme zadali nesprávne vstupné údaje). Preto sa pred "šípku von" zapisuje alebo názov premennej, ktorej hodnotu chceme získať, alebo text uzavretý medzi úvodzovky (napr. "nemá riešenie"). Súhrnne tieto rôzne druhy výstupných informácií nazývame *položky*.



Pôvodne dokázali počítače spracovávať iba čísla (preto sa nazývajú počítače). Neskôr sa ich využitie rozšírilo aj na spracovanie textových informácií (tzv. reťazce znakov), dnes je bežné, že počítač dokáže spracovávať aj grafickú informáciu (aspoň na výstupe).

Priraďovací príkaz: Zmena hodnôt premenných je počas vykonávania algoritmu možná dvomi spôsobmi: príkazom vstupu alebo priradením novej hodnoty. Priraďovací príkaz nariaďuje procesoru, aby vykonal na jeho pravej strane naznačené operácie alebo funkcie a výsledok uložil do premennej, meno ktorej je na ľavej strane.

V štruktúrogramoch ho zapisujeme v tvare 

, kde na ľavej strane priradenia je názov premennej, ktorej obsah sa má zmeniť, na pravej strane výraz, ktorý môže obsahovať konštanty, operácie alebo funkcie na spracovanie príslušného typu údajov (pozri kapitolu o typoch údajov).

Tieto príkazy sú dostačujúce pre spracovanie numerických a textových informácií, no ešte nám chýba dohovor o tvare podmienky (vo vetvení alebo v cykle).

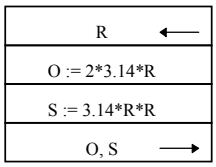
Podmienka je v programovacích jazykoch chápaná ako logický výraz, t. j. zistenie vzťahov (relácie) medzi výrazmi, prípadne zviazané logickými operáciami (*and* = "a súčasne", *or* = "alebo" a *not* = "neplatí, že").

To už je naozaj všetko, čo potrebujeme k tomu, aby sme si ukázali spôsob zápisu algoritmu na konkrétnych úlohách.

3.1 Príklady algoritmov

Vypočítajte obsah a obvod kruhu.

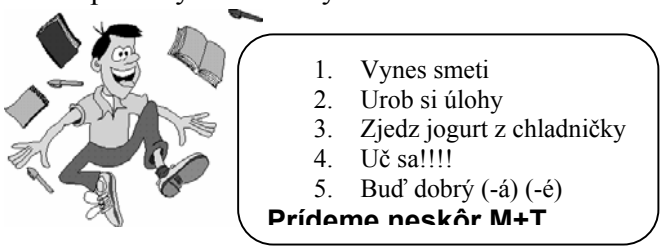
Nám známe vzorce na výpočet obsahu a obvodu kruhu musíme prepísať do algoritmického jazyka. Ak si uvedomíme, že na začiatku potrebuje procesor poznať konkrétnu hodnotu polomeru *r*, po vykonaní musí oznámiť hodnoty obsahu *S* a obvodu *o*, môžeme okamžite písať algoritmus v tvare štruktúrogramu.



V zápise algoritmu sú použité konvencie (dohovorené označenia) pre násobenie (*) a zápis desatinného čísla (desatinná bodka), ktoré sú bežné v programovacích jazykoch. Namiesto malých písmen *r, o* bežne používaných v matematike sme v algoritme použili veľké písmená *R, O*. Je to vecou dohovoru, pretože v algoritmoch (programoch) sa zvyčajne v názvoch premenných nerozlišujú malé a veľké písmená. Odporúčame však použiť veľké písmená pre tie premenné, ktoré sú zadávané zo vstupu, resp. pre tie, ktoré obsahujú výsledky spracovania algoritmu.

Sekvenciu – postupnosť príkazov poznáme dôverne aj z bežného života. Stačí si spomenúť na "prikázania", ktoré dostávame vo forme písomných odkazov rodičov, čo treba urobiť po príchode zo školy.

Po príchode zo školy sme našli prázdny dom a takýto odkaz:



Jasné, však? A je to iba iným spôsobom zapísaná sekvencia. V štruktúrogramoch netreba dopĺňať čísla jednotlivých činností, ich poradie je dané tým, ako sú zapisované pod sebou. A do algoritmu sa nepridávajú nijaké iné činnosti ako príkazy.

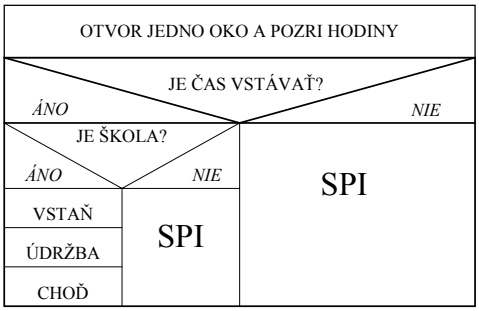
Napíšte podobný návod na činnosť po príchode zo školy v tvare štruktúrogramu. Vychádzajte z vlastných skúseností.

Rozhodovanie v závislosti od splnenia podmienky

Často sa v živote objavujú situácie, v ktorých sa musíme rozhodovať v závislosti od splnenia istých podmienok. Napr. dostanete ráno od krku rodiny jednoznačné inštrukcie: "Ak bude teplo, môžeš si vziať tričko, inak si musíš zobrať teplú košeľu!" Alebo: "Ak bude po kolená snehu, tak

si obuj zimné topánky, inak si zober tenisky!" Môže to byť rôznym spôsobom formulované – od prosby, vydierania až po tvrdý a jednoznačný príkaz. Smerom k počítačom je úloha zložitejšia. Všetko im musíme prikázať jednoznačne a presne. Relatívne podmienky, ako napr. "Ak bude teplo" nie sú vhodné, najčastejšie sa používajú porovnávania hodnôt a kombinácia logických operácií. Nezaškodí však, ak si najskôr ukážeme rozvetvenie postupu na príklade zo života. Zostavme algoritmus činnosti po zobudení.

Aj keď ráno väčšina z nás nemyslí na iné ako na to, kedy si znovu ľahne, postup by mal byť rôzny podľa toho, o aký deň v týždni ide. Cez pracovný týždeň musíme prekonať posteľnú príťažlivosť, vstať, zamaskovať sa, doplniť kalórie a obsah tašky a vyraziť do školy. Cez víkend sa môžeme spokojne obrátiť na druhú stranu a pokračovať v prerušenej piesni. Na obrázku je jeden z možných postupov zapísaný v tvare algoritmu.



Je zrejmé, že algoritmus nie je úplne v poriadku. Je totiž veľmi obtiažne opisovať reálne situácie jednoznačne. Pre zjednodušenie sme zapísali ďalšiu postupnosť činností po vynútenom vylezení z postele iba orientačne. Všimnite si, že v alternatíve s otázkou *Je čas vstávať?* je vložená do jednej jej časti ďalšia alternatíva s otázkou *Je škola?* Takýmto spôsobom sa riešenie realizuje jednou z troch možných postupností príkazov. Aj keď dve z nich končia rovnako – spaním.

Mohli by byť takéto pripomienky:
– Príkaz *SPI* (rozkazovací spôsob slovesa spať) je nedokonavý, môžeme kľudne spať nekonečne dlho.
– Škola je večná inštitúcia. Preto otázka *Je škola?* nie je úplne presná. Ale dá sa jej hovorovo rozumieť.
– Čo keď nie sú na dohľad jedného oka žiadne hodiny? Čo keď jediné dostupné hodiny stoja?...
– Ktorá z otázok by mala byť prvá? Ako je to u vás?

Skrátka, všeobecný algoritmus tak zložitej činnosti, akou je vstávanie, je problematické zostaviť.

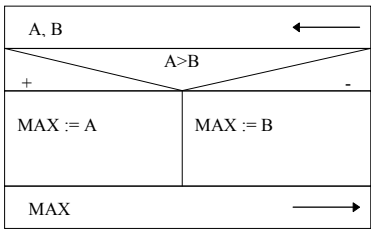
Upravte algoritmus tak, aby zohľadňoval vyššie uvedené pripomienky. Ako vyriešiť problém nedokonavého SPI?

Zistite maximálnu hodnotu z dvoch zadaných čísel.

Majme zadané dve hodnoty v premenných A, B. Potom algoritmus môžeme zapísať v tvare:

```
{VSTUP: A, B - čísla}
?
{VÝSTUP: MAX = maximum z hodnôt A, B}
```

Postup je jasný: Ak platí, že *A<B*, tak hodnota *MAX* bude rovná hodnote *A*, v opačnom prípade (keď *A>=B*) sa rovná hodnote *B*. To vedie k riešeniu v tvare algoritmu:



Komentár zrejme nie je potrebný. Ak sa hodnoty *A* a *B* rovnajú, *MAX* je jednou z nich.

Zistite maximum z troch zadaných hodnôt.

Riešeniu každého zložitejšieho algoritmu musí predchádzať rozbor úlohy, ktorý zabezpečí, že nezabudneme na žiadnu činnosť a správne rozvrhneme postup riešenia. Sme vopred nútení dobre si premyslieť štruktúru algoritmu (pre niektorých to asi nie je výhoda?!), čím sa hlavne pri zložitejších algoritmoch vyvarujeme zbytočnému prepisovaniu a učíme sa systematicky pristupovať k riešeniu problémov.

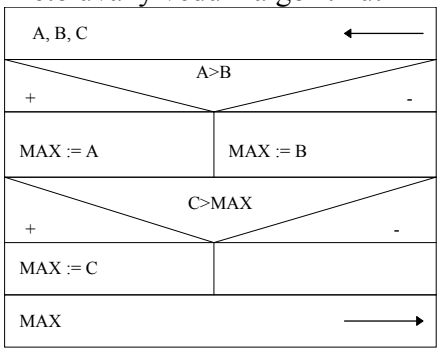
Predpokladajme, že máme tri hodnoty uložené v premenných *A*, *B*, *C*. Môžeme uvažovať všetky možné prípady: $A \geq B \geq C$ – potom *MAX* nadobudne hodnotu *A*; $B \geq A \geq C$ – potom *MAX* nadobudne hodnotu *B*; alebo $C \geq A \geq B$,...

Takýto prístup by vyžadoval prehodnotiť všetky možné situácie, ktoré medzi premennými môžu nastať. Treba si však uvedomiť, že my nerobíme usporiadanie, ale iba zisťujeme maximálnu hodnotu. A navyše! Môžeme predsa použiť už to, čo sme predtým vymysleli. Bolo by vhodné, keby sme si úlohu rozdelili na dve čiastočné podúlohy:

```
 {VSTUP: A, B, C - čísla}
Nájdenie maxima z čísiel A, B
{VÝSTUP: MAX = maximum z hodnôt A, B}
Nájdenie maxima z čísiel C, MAX
{VÝSTUP: MAX = maximum z hodnôt A, B, C}
```

Pritom postup pri nájdení maximálnej hodnoty spomedzi hodnôt *C* a *MAX* je taký istý ako pri riešení predchádzajúceho príkladu.

Tieto úvahy vedú k algoritmu:



Druhá alternatíva sa nazýva neúplná – v jednej z jej vetiev sa nemusí vykonať nič.

V týchto algoritmoch sa často objavujú programátorské konvencie: Znaky „≥” a „≤” nie sú na klávesnici počítača, preto sú nahrádzané zápismi „**>=**”, resp. „**<=**”; pretože sme nútení výrazy písať do jedného riadku, musíme pre určenie priority vykonávania operácií využiť okrúhle zátvorky.

Z tvaru algoritmu okamžite vidíme štruktúru riešenia, čo je výhodné hlavne pri hľadaní chýb (vo vlastnom algoritme, ale aj v algoritme napísanom niekým iným).

Najužitočnejšie je využívať algoritmy, v ktorých sa uplatňujú cykly. Tieto môžeme potom „prideliť na vykonávanie" nejakému zariadeniu, najlepšie počítaču. Ukážme si najskôr príklad zo života.

Predstavte si situáciu nie až tak veľmi odtrhnutú od života, ako na prvý pohľad vyzerá. Prišiel k nám na návštevu mimozemšťan Tobor. Pretože bol bez peňazí, vybavili sme mu brigádu v závode na výrobu hračiek. Jeho úlohou bolo stáť pri bežiacom páse, po ktorom prichádzali farebné kocky v náhodnom poradí, a tieto triediť – rozdeľovať do škatúľ podľa farby. Ako inteligentný tvor, Tobor vie:

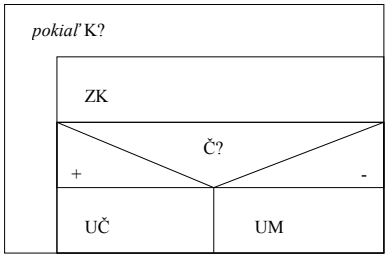
– čítať štruktúrogramy a riadiť sa ich obsahom
– zobrať kocku z pásu – označenie ZK
– zistiť, či je červená – označenie Č?
– uložiť kocku do „červenej" škatule – označenie UČ
– uložiť kocku do „modrej" škatule – označenie UM
– zistiť, či je na páse kocka – označenie K?

Zostavme algoritmus, podľa ktorého bude Tobor triediť kocky na červené a modré a ukladať ich do príslušných škatúľ.

Je zrejmé, že algoritmus musí obsahovať cyklicky sa opakujúcu činnosť – telo cyklu, ktorú si slovne môžeme zapísať takto:

1. Zober kocku (*ZK*).
2. Ak je červená (*Č?*), tak ju ulož do červenej škatule (*UČ*), inak ju ulož do modrej (*UM*).

Problémom je určiť typ cyklu. Vzhľadom na to, že nevieme, koľko kociek počas pracovnej zmeny príde, mal by to byť cyklus s neznámym počtom opakovaní. Vhodnejší by asi bol cyklus s podmienkou na začiatku. (Ľudia sú totiž „dobráci" a mohli by Tobora postaviť pred prázdny pás.) Riešením teda môže byť algoritmus:

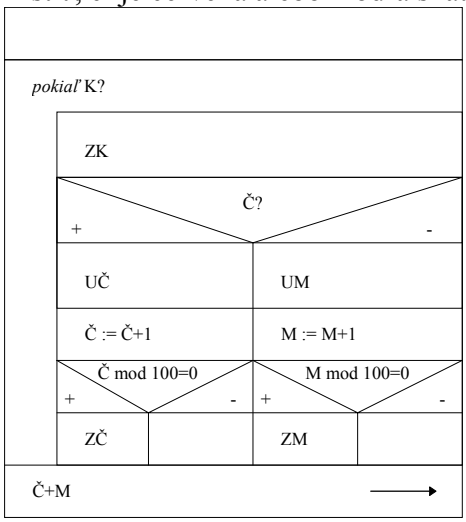


Čo sa stane, ak nejaký dobrák pošle na páse Toborovi zelenú kocku? Ako bude Tobor reagovať?

Zostaňme ešte chvíľu pri Toborovi a pomôžme mu riešiť veľmi dôležitý problém: Pretože si prišiel zarobiť, mal by vedieť, koľko vlastne kociek za smenu vytriedil. Navyše žiadna škatuľa nie je nekonečne veľká a mal by sa naučiť ich naplniť istým počtom kusov, zabaliť, odložiť a zobrať ďalšiu.

Je zrejmé, že budeme musieť rozšíriť Toborove schopnosti o ďalšie činnosti a zisťovania:

– zabaliť škatuľu príslušnej farby – označenie *ZČ* a *ZM* (už nebudeme uvažovať, že sa objavia iné ako červené a modré kocky); do tejto činnosti zaradíme aj odloženie škatule a prípravu novej,
– zvýšiť počet červených, resp. modrých o 1 - označenie *Č:=Č+1*, resp. *M:=M+1*,
– oznámiť počet roztriedených kociek, čo je súčet *Č+M*,
– zistiť, či je červená alebo modrá škatuľa plná.



Pri poslednej činnosti sa musíme trochu pozastaviť. Aby si Tobor pamätal počet všetkých červených a modrých kociek, ktoré počas smeny (činnosti algoritmu) roztriedil, nemali by sme *Č* a *M* okrem pridávania kociek meniť. Predpokladajme, že do škatule sa zmestí práve 100 kociek. Ako z hodnoty *Č*, resp. *M* zistíme, či už naplnil ďalšiu škatuľu? Jednoducho – ak príslušná hodnota je deliteľná číslom 100. Môžeme si to označiť už takmer "programátorsky": Ak platí, že *Č mod 100 = 0*, resp. *M mod 100 = 0*.

V algoritme je vynechaný obsah prvého príkazu. Vašou úlohou je doplniť ho. Mohlo by sa totiž stať, že Tobor by nesprávne zabalil prvú škatuľu každej farby a aj počet roztriedených kociek by nebol správny. Kedy to môže nastať?

Vypočítajte hodnotu súčtu prvých N prirodzených čísel.

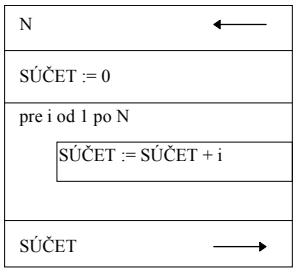
Máme pre konkrétnu hodnotu N zistiť súčet v tvare

```
SUČET = 1 + 2 + 3 + ... + N
```

Zapamätajte si, že tam, kde sa v rozbore úlohy objavia bodky "...", ide o cyklus. Navyše, ak vieme určiť počet jeho opakovaní (v tomto prípade N), použijeme cyklus so známym počtom opakovaní.

Pri písaní cyklu musíme vždy uvažovať počiatočné podmienky – prípad, kedy by cyklus neprebehol ani jedenkrát. Pre *N = 0* by mal byť výsledok *SÚČET = 0*, preto priradenie tejto hodnoty musí byť zaradené pred cyklus. (Umiestnenie hodnoty 0 má aj ďalší význam: Mohlo by sa stať, že v premennej *SÚČET* zostal z predchádzajúcich výpočtov nejaký "zvyšok". Potom by cyklus nepočítal stanovený súčet, ale dával by hodnotu *"zvyšok"+1+2+ ... +N*. Takto zvolená počiatočná hodnota sa nazýva invariant vzhľadom na sčítanie.) V každom ďalšom (pracovne nazvanom *i*-tom) kroku sa hodnota premennej *SÚČET* zvýši o *i*.

Riešením našej úlohy je algoritmus:



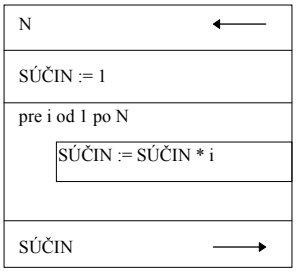
Príkaz v tele cyklu znamená toto: "K momentálnej hodnote premennej *SÚČET* pripočítaj momentálnu hodnotu *i* a výsledok ulož do premennej *SÚČET*." Takéto príkazy nám umožňujú cyklicky meniť obsahy premenných.

Vypočítajte hodnotu súčinu prvých N prirodzených čísel (tzv. N-faktoriál).

Máme pre konkrétnu hodnotu N zistiť súčin v tvare

```
SÚČIN = 1 * 2 * 3 * ... * N
```

Tento príklad je po vyriešení predchádzajúceho problému jednoduchý. Stačí si iba uvedomiť, že "najhorší možný prípad" – keď sa cyklus nevykoná ani jedenkrát, má dávať hodnotu *SÚČINU* rovnú 1 (tzv. invariant vzhľadom na násobenie).



Keď si prezrieme obidva algoritmy, vidíme, že sa neodlišujú štruktúrou použitých algoritmických konštrukcií – sú vlastne graficky rovnaké. A to je správne, pretože postup pri získaní výsledku zostal; len namiesto 0 sme priradili 1, namiesto sčítania sme násobili.
Na tomto príklade je možné ukázať význam efektívnosti algoritmu (P6). Okrem tohto cyklického postupu existuje možnosť zistiť hodnotu súčtu jednoduchým dosadením do známeho vzorca:

```
SÚČET = n*(n+1)/2
```

Bohužiaľ, podobný vzorec na výpočet súčinu prvých N prirodzených čísiel nemáme.

Vypočítajte ciferný súčet číslic daného prirodzeného čísla N.

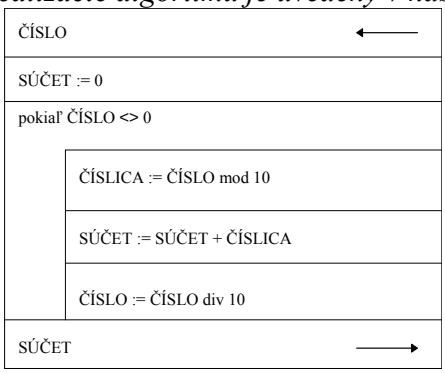
Ciferný súčet číslic daného čísla sa používa napríklad pri zisťovaní deliteľnosti niektorými číslami, pri zisťovaní "šťastného životného čísla" v rôznych horoskopoch a pod. Ciferným súčtom čísla 125 je 8, ciferným súčtom čísla 1997 číslo 26...

Ak je číslo veľmi dlhé, ani pre človeka nie je táto úloha úplne elementárna. Musí brať postupne číslicu po číslici a pripočítavať jej hodnotu k predchádzajúcemu súčtu. Podobným spôsobom musíme úlohy, v ktorých treba číslo spracovávať číslicu po číslici, riešiť aj na počítači. Navyše "odtrhnutie číslice" nie je pre počítač elementárnou činnosťou – musí byť uskutočnené pomocou napr. celočíselného delenia so zvyškom.

Celočíselné delenie sa zvykne zapisovať pomocou "kľúčových" (vyhradených) slov *div, mod*:
Výsledkom operácie "a *div* b" je celočíselný podiel po delení čísla *a* číslom *b*.
Napr. 7 div 2 = 3, 12 div 6 = 2, 17 div 19 = 0.
Výsledkom operácie "a *mod* b" je zvyšok po celočíselnom delení čísla *a* číslom *b*.
Napr. 7 mod 2 = 1, 12 mod 6 = 0, 17 mod 19 = 17.

Výsledkom podrobnejšieho rozboru úlohy môže byť algoritmus na nasledújúcej strane

Vyskúšajte si realizáciu daného algoritmu na niekoľkých konkrétnych príkladoch tak, že si budete písať postupné zmeny hodnôt jednotlivých premenných. Spôsob zápisu sledovania realizácie algoritmu je uvedený v nasledujúcej tabuľke pre hodnotu ČÍSLO = 1998.



ČÍSLO	SÚČET	ČÍSLICA
1997	0	8
199	0+8=8	9
19	8+9=17	9
1	17+9=26	1
0	26+1=**27**	

V tabuľke sa pod seba presne podľa algoritmu zapisujú zmeny jednotlivých premenných. Takejto tabuľke sa zvykne hovoriť *tabuľka sledovania výpočtu*. Bohužiaľ, nie je dôkazom správnosti algoritmu, iba jeho orientačným preverením pre konkrétne vstupné hodnoty.

Zostavte algoritmy nasledujúcich úloh:

1. V obale starej zámockej kroniky bol objavený lístok s takýmto odkazom (je trochu upravený, aby bol čitateľný): "Stanúc na poludnie v deň letného slnovratu na špicu tieňa Hlavnej veže odmeraj 30 lakťov v smere východu slnka. Otoč sa vľavo a odmeraj dva tucty stôp. Vyber kameň. V hĺbke kopy palcov je ukrytý poklad." Zostavte daný postup v tvare algoritmu zrozumiteľného dnešným ľuďom. Uvažujte aj nepriaznivý výsledok.

2. Postup pri prechode cez ulicu, napr. pre Tobora. (Zrejme bude potrebné vybrať konkrétny typ križovatky alebo komunikácie.)

3. Postup telefonovania z telefónneho prístroja doma alebo telefónneho automatu.

4. Postup pri príprave jednoduchého jedla.

5. Doplňte nové činnosti (ak sú potrebné) pre Tobora a upravte algoritmus jeho činnosti tak, aby po skončení práce oznámil celkový počet škatúľ jednotlivých farieb, ktoré zabalil.

6. V prípade zmeny činností pre Tobora by sa dal podstatne zefektívniť jeho postup pri práci. (Návod: Ak by sme uvažovali, že činnosť ULOŽ môže mať parameter FARBA – označme si to ako ULOZ(FARBA) – mohli by sme naučiť Tobora triediť kocky podľa farieb (môže ich byť ľubovoľné množstvo) do príslušných škatúľ. Skúste zmeniť činnosti pre Tobora a zostaviť takýto postup v tvare algoritmu.)

7. Ak ide v prípade činnosti Tobora o hračky - stavebnice farebných kociek by nemali byť jednofarebné. Napr. v jednej škatuli by malo byť 20 kociek, (po 4 červené, modré, biele, zelené, žlté. Zostavte postup činnosti pre Tobora, aby takto balil škatule "pestrofarebných" kociek.

8. Zistite počet všetkých deliteľov daného prirodzeného čísla N.

9. Zistite, či dané prirodzené číslo N je prvočíslo alebo číslo zložené. (Môžete využiť predchádzajúce riešenie?)

10. "Palindrom" je číslo, ktoré pri čítaní odpredu aj odzadu dáva ten istý výsledok. Zistenie, či dané prirodzené číslo N je palindrom.

11. Na základe znalosti hodnôt vzdialenosti stredov a polomerov dvoch kružníc vieme určiť ich vzájomnú polohu. Zostavte algoritmus na zistenie tejto polohy v rovine.

12. Určite si pamätáte na "strašiaka" našich školských liet – tzv. trojuholníkovú nerovnosť. Ak nie, nevadí, tu je upravená pre naše potreby: "Súčet ľubovoľných dvoch strán trojuholníka je väčší ako strana tretia". Zostavte algoritmus, ktorý pre zadanú trojicu čísiel určí, či môže alebo nemôže byť stranami trojuholníka.

13. Pre výpočet šťastného čísla sa používa ciferný súčet číslic nejakých osobných údajov (dátum narodenia). Platí však, že sa číslice sčítavajú dovtedy, kým nedostaneme jednociferné číslo. Napr. pre náš príklad 1998 sme dostali 27, a ešte raz sčítame 2+7 = 9, čo je výsledkom. Zostavte takýto postup určenia šťastného čísla pre dané prirodzené číslo N.

4. RÝCHLO DO PASCALU – ZAPNITE TURBO

4.1 Na začiatok trochu histórie a teórie

Programovací jazyk **pascal** (nazvaný podľa Blaise Pascala[1]) vytvoril Niklaus Wirth[2] s cieľom zabezpečiť vyučovanie systematického programovania. Rozšíril tak "babylon" programovacích jazykov, ktorých v tej dobe vznikalo neuveriteľné množstvo. Ideou bolo vytvoriť programovací jazyk, ktorý by bol kompromisom medzi abstraktnými štruktúrami algoritmov a konkrétnou reprezentáciou spracovávaných údajov v počítači a nutnosťou poznať technické parametre. "Je jednoduchšie vytvoriť program, v ktorom sa manipuluje s pojmami, ako sú čísla, množiny, postupnosti alebo cykly, ako taký, v ktorom sa používajú pojmy bity, slová alebo skoky," znelo jeho motto. Ako sa mu to podarilo, musíte posúdiť sami. Významnou črtou pascalu je jeho štruktúrovanosť, ktorá neumožňuje pri správnom programovaní vytvárať krkolomné monštrá príkazov, v ktorých sa nevyzná od istého momentu ani sám autor programu.[3]

Existuje mnoho programovacích jazykov, niektoré z nich sa pokúšajú o univerzálnosť, iné sú orientované na špeciálne oblasti. Tabuľka naznačuje krátky prierez vývojom programovacích jazykov, pričom obsahuje len tie známejšie:

obdobie	*programovanie*
40-te roky 20. storočia	programovanie v strojovom jazyku počítača
50-te roky 20. storočia	programovanie v jazyku symbolických adries
1956	programovací jazyk fortran (FORmula TRANslation)
1958	programovací jazyk algol (ALGOrithmic Language)
1961	programovací jazyk basic (Beginners All-Purpose Symbolic Instruction Code)
okolo 1970	programovací jazyk pascal (systematické, štruktúrované programovanie
okolo 1980	programovací jazyk C (prechod k objektovo orientovanému programovaniu
90-te roky 20. storočia	vývoj komplexnejších verzií jazykov s cieľom využitia nových možností predovšetkým osobných počítačov (grafika, zvuk, mul- timédiá), programovanie riadené udalosťami (Visual Basic)
súčasnosť	programovacie jazyky pre tvorbu aplikácií v globálnych sieťach (Java)
budúcnosť	nechajme sa prekvapiť

Každý programovací jazyk, ktorý sa presadil, má viacero rôznych verzií - jednak orientovaných na konkrétne počítače (napr. GW basic, Quick basic, Turbo basic...), jednak vývojových (napr. Turbo pascal 5.0, 6.0, 7.0 alebo C, C++).

V každom programovacom jazyku však existuje istá množina prostriedkov, ktoré sú rovnaké vo všetkých verziách. Hovorí sa jej štandard programovacieho jazyka.

---

[1] Blaise Pascal (1623-1662) – významný francúzsky matematik, fyzik a filozof. Okrem iného sa zaoberal binomickými koeficientami (Pascalov trojuholník), teóriou pravdepodobnosti, kuželosečkami. R. 1642 zostavil počítací stroj na aritmetické operácie, skúmal hydrostatický tlak, formuloval Pascalov zákon.

[2] Niklaus Wirth (nar. 1934) – profesor informatiky na Vysokej škole technickej a Univerzite v Zürichu (Švajčiarsko), popredný informatik, autor publikácií, ktoré sa stali klasickou výbavou programátorov napr.: N. Wirth: Systematické programovanie. Alfa, Bratislava 1979, N. Wirth: Algoritmy a štruktúry údajov. Alfa, Bratislava 1988, 1989.

[3] Čo dokáže najviac potešiť "notorického čitateľa programov"? (Tým môže byť učiteľ, kolega, niekedy aj sám autor po istom čase.) Je to názov, z ktorého nie je zrejmé, čo je úlohou programu (prípadne objektu). Veľmi účinné sú v tomto prípade názvy typu JANO1, JANO2... , ale aj SKUSKA, prípadne P1, P2, POKUS...

4.2 Kedy už budeme programovať?

Na to, aby sme dokázali napísať a spustiť program v pascale, si potrebujeme ešte niečo povedať o tvare programu v pascale a o prostredí, v ktorom ho budeme písať. Zameriame sa na Turbo pascal (TP), bežne realizovateľný na osobných počítačoch IBM PC. Takže do toho.

Tvar programu v pascale

Zápis programu v pascale má svoju konvenciu
(tzv. dobrovoľne povinnú dohodu). Má tvar:

```
Program meno;
deklarácie a definície objektov;
Begin {začiatok hlavného programu}
      vlastný program - algoritmus prepísaný do programovacieho jazyka
End. {koniec hlavného programu}
```

DOHODA: *Kurzívou*, teda "*šikmým"* písmom budeme označovať neterminálne symboly, ktoré budú ešte ďalej špecifikované. Do zátvoriek { } budeme umiestňovať komentáre pre lepšie pochopenie programu jeho čitateľom, nie počítačom, ktorý ich ignoruje. Neskôr budeme "príkrasy" vynechávať.

Medzi jednotlivými časťami, ako aj medzi príkazmi programu sú oddeľovačmi bodkočiarky!

Meno programu môže byť ľubovoľná postupnosť znakov (písmen, číslic a znaku podčiarknika "_"); najvhodnejšie je, ak vystihuje problém, ktorý program rieši[10]. Pritom nie sú rozlišované malé a veľké písmená a meno programu nesmie byť zhodné s nejakým menom objektu v ňom použitým alebo všeobecne definovaným.

4.3 Príkazy vstupu a výstupu v pascale

Príkaz výstupu umožňuje zobrazenie spracúvaných údajov - medzivýsledkov, výsledkov, ako aj zobrazenie komentárov pre lepšiu prehľadnosť a zrozumiteľnosť komunikácie s počítačom. Zamerajme sa zatiaľ iba na alfanumerický výstup prostredníctvom obrazovky monitoru – štandardná súčasť pascalu. Môže mať jeden z nasledujúcich tvarov:

`Write` (*zoznam položiek oddelených čiarkami*)
`WriteLn` (*zoznam položiek oddelených čiarkami*)

Pritom položka môže byť konštanta, premenná alebo výraz príslušného typu. Výsledkom príkazu *Write* ("píš") je vypísanie konkrétnych hodnôt položiek tesne za sebou a ponechanie kurzora v tom istom riadku. Príkaz *WriteLn* ("píš a odriadkuj") má ten istý efekt s rozdielom, že nakoniec sa odriadkuje, t. j. kurzor sa premiestni na ďalší riadok a čaká na ďalší príkaz výstupu.

Obrazovka monitoru má v TP v tzv. alfanumerickom móde 25 riadkov po 80 znakov. Program automaticky prenesie ďalšie zobrazované znaky na nový riadok, ak ich počet presiahne 80; ak má byť na obrazovke zobrazených viac ako 24 riadkov, všetky zobrazené riadky sú "rolované" smerom nahor, pričom obsah najhornejších je zabudnutý.

Uvedené poznatky nám stačia na napísanie jednoduchého programu, v ktorom sa predstaví počítač.

Najskôr uveďme program:

```
Program SLUSNE_POZDRAV_01;
  {žiadne objekty nepotrebujeme}
Begin
 WriteLn ('Dobry den, som Tvoj oddany pocitac! ');
 WriteLn;                                  {prazdny riadok}
 WriteLn (' Budem robit iba to, co ma naucis! ')
End.
```

Výsledok programu je zrejmý: Zobrazí na obrazovke text, ktorý je uvedený v apostrofoch ' ' – tak sa označuje konštantný text, ktorý môže obsahovať ľubovoľné znaky z klávesnice. Pozor! Aj medzera je znak často veľmi dôležitý pre lepšiu prehľadnosť výstupu.

Príkaz vstupu zabezpečuje voľbu vstupných hodnôt premenných užívateľom programu. V pascale môže mať tvar:

Read (*zoznam premenných oddelených čiarkami*)

resp. **ReadLn** (*zoznam premenných oddelených čiarkami*)

Rozdiel medzi nimi je predovšetkým v tom, že kurzor zobrazujúci miesto komunikácie pri príkaze *Read* zostáva v tom istom riadku (môže tam byť aj viac údajov), pri *ReadLn* odriadkuje. Významne sa odlišuje ich funkcia pri práci so súbormi (pozri J. Skalka: Turbo pascal II.).

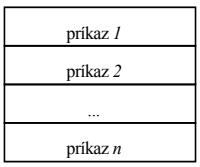
4.4 Prepis štruktúrogramov do pascalu

Veľkou prednosťou štruktúrogramov je možnosť priameho prepisu do programovacieho jazyka pascal. Stačí si uvedomiť niekoľko základných konvencií a tieto takmer mechanicky používať.

Samozrejme, že v štruktúrogramoch chýba meno a popis objektov, ktoré algoritmus spracúva, ale inak sa môžeme riadiť nasledujúcou tabuľkou prepisu:

Sekvencia príkazov

Je vhodné, ale nie je to podmienkou, aby jednotlivé príkazy sekvencie boli písané v samostatných riadkoch, a je vhodné zapisovať ich na rovnakej, od začiatku riadkov vhodne vnorenej pozícii. Oddeľovačom príkazov je bodkočiarka.




```
begin
        príkaz 1;
        príkaz 2;
        ...
        príkaz n;
end
```

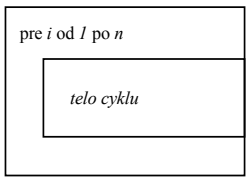
Alternatíva

Ani pri zápise alternatívy nie je pevne stanovené, ako ju prehľadne zapisovať. Je jasné, že namiesto príkazov 1 a 2 môžu byť pokojne zapísané iné algoritmické konštrukcie – sekvencia, ďalšia alternatíva, cykly atď. V týchto prípadoch je vhodné, ak si časti za *then* a *else* uzatvoríte do zátvoriek begin-end. Pred *else* však nesmie byť bodkočiarka!


```
if podmienka
    then  príkaz 1
    else  príkaz 2
```

Cyklus so známym počtom opakovaní

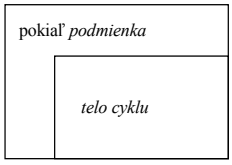
Riadiaca premenná cyklu (v tomto prípade *i*) sa po vykonaní cyklu zvýši o jedna (resp. na nasledujúcu hodnotu ordinálneho typu; pozri časť o typoch údajov). Náhradou kľúčového slova *to* na *downto* sa hodnota riadiacej premennej znižuje. Telo cyklu sa nemusí vykonať ani raz.




```
for i:= 1 to N do
        begin
                telo cyklu
        end
```

Cyklus s neznámym počtom opakovaní s podmienkou na začiatku

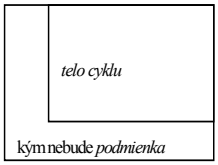
Cyklus sa opakuje, pokiaľ je splnená podmienka. Nesmieme zabudnúť, že v tele cyklu sa musí niečo meniť tak, aby od určitého okamihu podmienka cyklu prestala platiť. Cyklus nemusí prebehnúť ani jedenkrát.




```
while podmienka do
        begin
                telo cyklu
        end
```

Cyklus s neznámym počtom opakovaní s podmienkou na konci

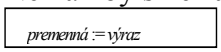
V tomto prípade nie je potrebné využívať begin-end, cyklus má vlastné "zátvorky" repeat-until. Cyklus sa uskutoční vždy aspoň raz; ukončí sa, ak bude prvýkrát splnená podmienka cyklu.




```
repeat
        telo cyklu
until podmienka
```

Príkaz priradenia

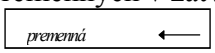
Nemali by sme zabudnúť na to, že premenná a výraz musia byť rovnakého typu.




```
premenná := výraz
```

Príkaz vstupu

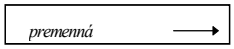
Ak chceme zabezpečiť vstup viacerých premenných pomocou jedného príkazu, mená premenných v zátvorke oddeľujeme čiarkami. (Podrobnejšie ďalej.)




```
ReadLn (premenná)
```

Príkaz výstupu

Vystupovať nemusí iba hodnota premennej, ale aj hodnota výrazu, číselná alebo reťazcová konštanta a ich kombinácie. (Podrobnejšie ďalej.)




```
WriteLn (premenná)
```

4.5 Deklarácie premenných

Na rozdiel od algoritmu musí byť pred začiatkom činnosti programu v pascale určené, s akými objektmi má program pracovať. Slúži to na vytvorenie príslušných pamäťových miest. Zatiaľ budeme pracovať iba s najjednoduchšími typmi údajov (s celými číslami a reťazcami znakov). Typ údajov je predstavovaný množinou prípustných hodnôt, operáciami a funkciami, pomocou ktorých je možné hodnoty spracúvať:

Typ **integer** – celé čísla – operácie +, -, * (krát), **div** (celočíselný podiel), **mod** (celočíselný zvyšok).

Typ **string** – reťazec znakov – operácia + (zlučovanie reťazcov za seba).

Deklaráciu premenných s určením ich typu uvádzame za hlavičkou programu s tým, že na jej začiatku uvedieme kľúčové slovo **var**, napr.:

```
Program DEKLARUJ_1;
var   a, b, c: integer;
      meno: string;

Begin
```

```
   Hlavný program;
End.
```

Teraz už môžeme bez problémov písať jednoduché programy v pascale. Uveďme si niekoľko príkladov.

Zostavte program, ktorý slušne pozdraví a vypýta si meno užívateľa.

```
Program SLUSNE_POZDRAV_02;
  var meno   :string;
Begin
 WriteLn(´Som super-pocitac a som pripravený sluzit Ti do strhania obvodov! ´);
 {1} Write (‘Ty si kto? ‘); ReadLn (meno);
 {2} WriteLn; WriteLn;
 {3} WriteLn (‘Velmi ma tesi, ‘, meno, ‘, ideme na to! ‘);
 {4} ReadLn;
End.
```

V riadku označenom poznámkou {1} sú uvedené dva príkazy: *Write* vypíše text *“Ty si kto?”* a kurzor zostane v tom istom riadku, *ReadLn* čaká na zadanie mena a ukončenie stlačením *ENTER*. Treba si uvedomiť, že pre užívateľa programu je program vlastne “čiernou skrinkou” – niečím, do čoho nevidí a komunikuje s “vnútrom” iba pomocou toho, čo vidí na obrazovke. {2} vynechá dva riadky, {3} vypíše text *“Velmi ma tesi,”*, za to doplní zadané meno a na koniec text *“ , ideme na to!”*. {4} – “prázdny” *ReadLn* zastaví činnosť programu do stlačenia *ENTER*. Je vhodné zadávať ho vždy na koniec programu.

Zostavte program, ktorý by N-krát pozdravil “Ahoj!”.

Bez zvláštneho komentára si uveďme program s riešením zobrazenia N pozdravov pod sebou:

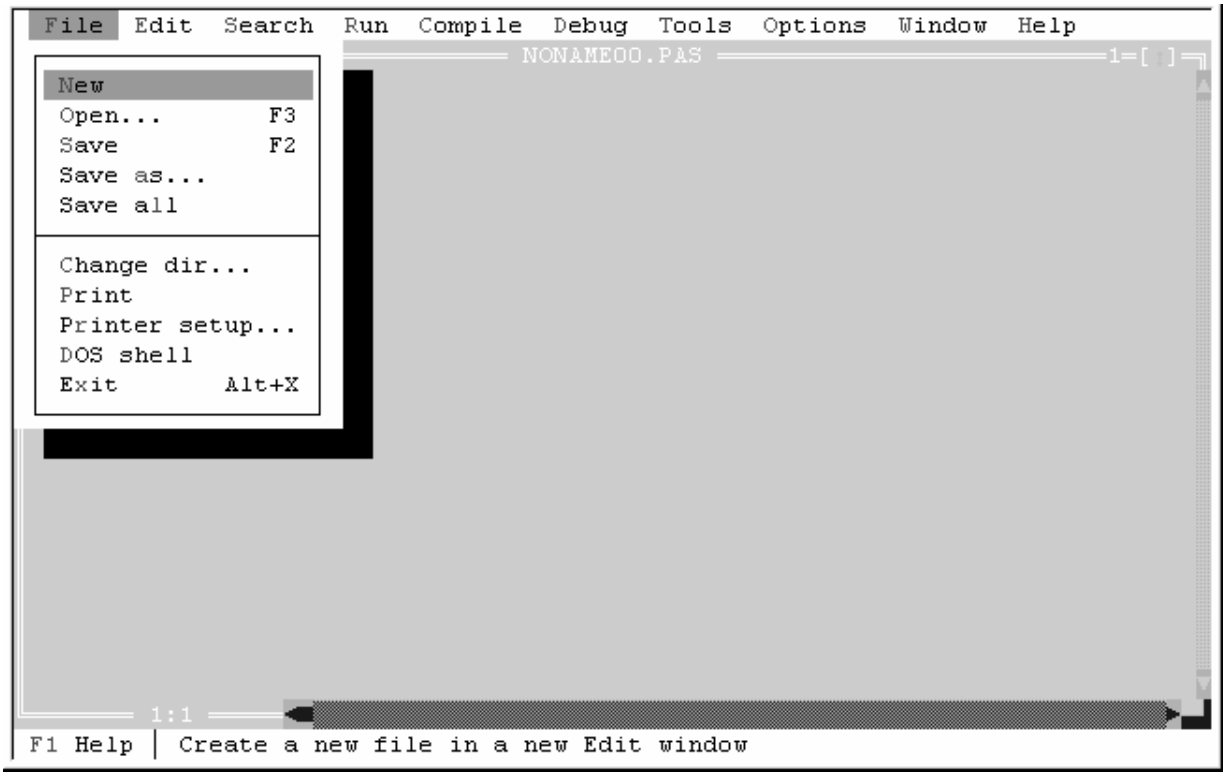
```
Program Pozdrav_N_krat;
var   i, j, N       : integer;
Begin
 Write (‘Zadaj pocet: ‘); ReadLn (N);
 for i:= 1 to N do
 begin
  WriteLn (‘Ahoj! ‘);
 end;
 ReadLn;
End.
```

4.6 Prostredie Turbo pascalu

Turbo pascal (TP) je prostriedok, ktorý nám umožňuje efektívne písať pascalovské programy na osobnom počítači. Spúšťa sa z príslušného adresára (podľa nastavenia ciest – PATH) pomocou súboru ***turbo.exe***. Po odštartovaní TP sa nám na obrazovke objaví základné vývojové integrované prostredie (Integrated Development Environment).

Obrazovka TP v stave editovania (písania, opráv a ladenia) programu má tri základné časti:

1. Editovací priestor – najväčšie (modré) okno, v ktorom sa zobrazuje zvolený program v tvare dokumentu – stránky. Po tomto okne sa pohybujeme pomocou šípok alebo myši, môžeme využívať všetky možnosti práce s editovaním textu programu (podobné práci s textovým editorom T602).



2. Hlavné menu – obsahuje zoznam základných činností, ktoré je možné vykonávať v TP. Každá z volieb je tvorená roletovým menu, ktoré otvára ďalšie (pod-)možnosti.

File – prostriedky pre prácu so súbormi a adresármi včítane ukončenia činnosti,
Edit – úpravy práve editovaného dokumentu programu,
Search – nájdenie zvolených častí programu,
Run – spustenie programu v režime interpretera,
Compile – preklad programu do tvaru vykonateľného súboru (prípona ***exe***),
Debug – ladenie programu, odstraňovanie chýb počas jeho behu,
Tools – ďalšie nástroje pre ladenie programu,
Options – nastavenie podmienok pre IDE, využívané technické a programové prostriedky,
Window – spôsob usporiadania a veľkosti okien pracovných plôch,
Help – široko rozvetvená nápoveda (bohužiaľ v angličtine), umožňujúca efektívnu orientáciu v prostriedkoch TP.

Okrem možnosti výberu pomocou klávesu *F10* a šípok alebo myši je efektívne využívať stlačenie *Alt* v kombinácii so zvýrazneným písmenom v menu.

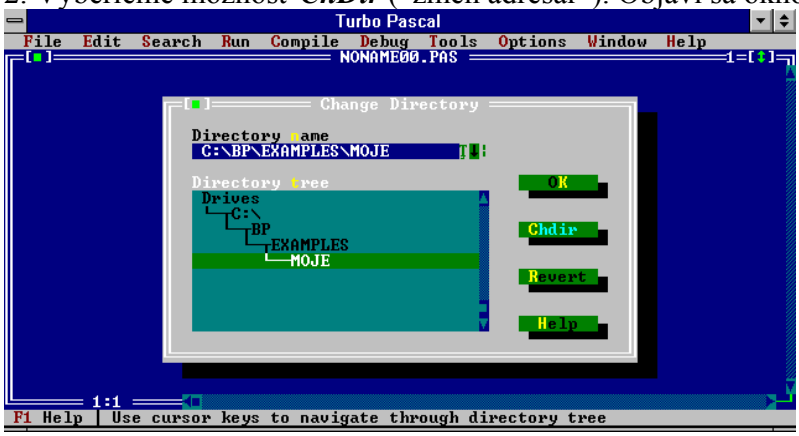
3. Okamžitá nápoveda – "**horúce klávesy**" (z angl. hot keys) – v každom momente činnosti IDE dáva možnosť rýchlej voľby výberu možností pomocou stlačení príslušných klávesov.

Pre toho, kto sa iba učí programovať, je väčšina volieb nepotrebná, zvlášť, ak máme už niekým nastavené prostredie (napr. v učebni). Pre osobné používanie je potrebné preštudovať odporúčanú literatúru.

Postup pri vytváraní nového programu

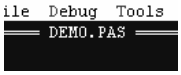
Ak chceme písať nový program, je dôležité dodržať nasledujúci postup. Inak by sa mohlo stať, že sa nám vytvorený program "stratí" v množstve ďalších súborov:

1. Otvoríme voľbu ***File*** v hlavnom menu (možno použiť kombináciu klávesov *Alt+F*).
2. Vyberieme možnosť ***ChDir*** ("zmeň adresár"). Objaví sa okno upresnenia parametrov.



Tu pomocou klávesnice (priamym zápisom, resp. využitím *TAB* a šípok) alebo myši vyberieme pracovný adresár, v ktorom máme svoje pascalovské programy.

3. Ak píšeme nový program, mali by sme ho už mať pripravený a vedieť jeho meno. Zvolíme skratku tohto mena (max. 8 znakov, prvý znak musí byť písmeno), ktorú zapíšeme po výbere voľby ***Save as...*** v menu ***File*** do okna upresnenia, ktoré sa objaví.



Po odoslaní pomocou *ENTER* sa objaví pracovná plocha, do ktorej môžeme písať nový program. V strede jej horného okraja vidíme názov súboru s príponou ***pas***, ktorú si IDE doplňa automaticky.

4. Je dobré, ak si z času na čas písaný program uložíme – pri náhodnom výpadku elektriny by mohlo dôjsť k nepríjemnej situácii. Optimálnu voľbu nám ponúka spodné menu stlačením ***F2***.

Otváranie, editovanie a spúšťanie hotového programu

1. Podobne ako v prípade písania nového programu najskôr pomocou voľby ***ChDir*** z menu ***File*** zvolíme pracovný adresár, v ktorom sa nachádza daný program.
2. V menu ***File*** si vyberieme voľbu ***Open***. Objaví sa okno, ktoré obsahuje zoznam všetkých súborov, ktoré sa v pracovnom adresári nachádzajú. Pomocou klávesnice (priamym zápisom, resp. využitím *TAB* a šípok) alebo myši vyberieme otváraný súbor a potvrdíme voľbu *ENTER*. Otvorí sa nám príslušný program, ktorý môžeme editovať (opravovať, prepisovať, doplňať).
3. Ak chceme, aby sa program začal vykonávať, otvoríme (napr. pomocou *Alt+R*) menu ***Run*** a v ňom voľbu ***Run*** (resp. *Ctrl+F9*). Vtedy IDE najskôr vykoná syntaktickú analýzu (preverí správnosť syntaxe programu) a ak je v poriadku, začne program vykonávať. (V tomto režime pracuje IDE ako interpretera, nevytvára cieľový program v strojovom jazyku.[1])

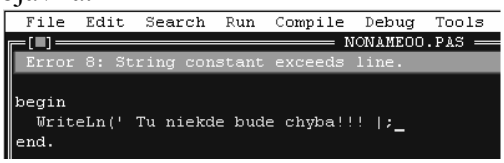
Editovacie okno zmizne a objaví sa obrazovka vykonávania programu, na ktorej vidíme výsledky činnosti programu.

4. Vykonávanie programu môže skončiť dvomi a neskončiť jedným spôsobom:

a) Program prebehne úspešne, po získaní výsledkov sa znovu zobrazí editačné okno. (*Poznámka*: Niekedy nestihneme "zachytiť" výsledky. Preto je dobré na koniec programu pridávať príkaz *ReadLn* bez parametrov – vtedy program čaká na stlačenie *ENTER*.)

[1] V Borland pascale (bp.exe), ktorý sa od verzie 7.0 dodáva spolu s TP, sa automaticky vytvára súčasne aj rovnomenný spustiteľný program s príponou **exe**.

b) Počas činnosti programu dôjde k chybe, ktorá preruší vykonávanie. Môže to byť jednak chyba nesprávne zadaného vstupu (napr. namiesto čísla znak) alebo chyba súvisiaca s nesprávnou konštrukciou programu. Vtedy sa zobrazí znovu editovacie okno a v hornom riadku svieti číslo chyby s jej (anglickým) komentárom. Kurzor editovania sa nastaví tam, kde sa chyba objavila.



Pri editovaní programu môžeme využívať kombinácie klávesov, ktoré sú podobné pre väčšinu DOS-ovských editorov:

Ctrl+N	vloženie riadku,
Ctrl+Y	vymazanie riadku,
Ctrl+Q Y	vymazanie do konca riadku,
Ctrl+K B	začiatok bloku,
Ctrl+K K	koniec bloku,
Ctrl+K C	kópia bloku,
Ctrl+K V	presun bloku,
Ctrl+Ins	kópia bloku do clipboardu (schránky),
Ctrl+Del	prenesenie bloku do clipboardu,
Shift+Ins	"nalepenie" bloku z clipboardu,
Shift+šípky	"ručné" označenie bloku.

Zvláštne postavenie má stlačenie kombinácie *Ctrl+F1*, ktorá predstavuje **kontextovo závislú nápovedu**. Ak je kurzor nastavený na niektorom z kľúčových slov pascalu a použijeme túto kombináciu, zobrazia sa informácie popisujúce význam a syntax (spôsob zápisu) daného slova (samozrejme v angličtine)[1].

[1] Podrobnejší popis jednotlivých možností ako efektívne využívať prostredie TP nájdete vo viacerých knihách, napr.: Mikula, P. , Juhová, K. , Soukenka, J.: Borland Pascal 7.0 – kompendium. Grada Praha 1994. Mikula, P. , Juhová, K. , Soukenka, J.: Turbo pascal 7.0 – kompletní průvodce. Grada Praha 1994. Mikula, P.: Pascal 7. 0 – od příkladů k příkazům. Grada Praha 1994.

5 TYPY ÚDAJOV

Doteraz sme sa venovali predovšetkým tomu, ako zorganizovať postupnosť vykonávania jednotlivých činností algoritmu. Algoritmus však vždy spracúva (ak má byť hromadný) nejaké údaje a zo vstupných údajov vytvára údaje výstupné. Takýmito údajmi môžu byť čísla, znaky, texty, ale aj obrázky, tabuľky hodnôt... Pretože ide o rôzne kategórie údajov, ktoré treba aj rôznymi spôsobmi spracúvať, hovoríme o tzv. **typoch údajov**. Typom údajov chápeme množinu prípustných hodnôt (čísla, znaky...) spolu s operáciami a funkciami, pomocou ktorých ich spracúvame. Najpoužívanejšími typmi údajov pri programovaní sú celé a reálne čísla, znaky, reťazce znakov, logický typ a niektoré ďalšie. Venujme sa im podrobnejšie.

5.1 Celé čísla

Údajový typ, pomocou ktorého v pascale môžeme pracovať s celočíselnými hodnotami, sa nazýva ***integer***. Nie sú to celé čísla v našom matematikou "deformovanom" chápaní od -∞ po +∞, ale iba čísla z presne stanoveného rozsahu:

```
-MaxInt < 0 < +MaxInt
```

Hodnota u TP je: *+MaxInt=32767* a *-MaxInt=-32768*.

(Premenné typu *integer* sú štandardne uložené v dvoch bajtoch.) TP rozširuje základný údajový typ *integer* o ďalšie štyri typy, ktoré majú rozdielny rozsah hodnôt:

byte	0..255	(zaberá 1 bajt)
shortint	-128..128	(zaberá 1 bajt)
word	0..65535	(zaberá 2 bajty)
longint	-2147483648..2147483647	(zaberá 4 bajty)

S celočíselnými hodnotami a výrazmi môžeme vykonávať operácie, ktoré k tomuto typu patria. Sú to: sčítanie (+), odčítanie (-), násobenie (*), celočíselný podiel (*div*), celočíselný zvyšok (*mod*).

Zložitejšie výrazy zapisujeme pomocou okrúhlych zátvoriek rovnako ako v matematike. Podobne je to aj pri vyhodnotení výrazu, kde platia zaužívané priority – ak nie sú zátvorky, najskôr sa vykoná násobenie a delenie, potom sčítanie a odčítanie.

V jednej prebohatej krajine za siedmimi horami, dolami, moriami poskytujú svojim obyvateľom bezúročné pôžičky! Zostavte program, ktorý pre danú pôžičku a mesačnú splátku určí, koľko mesiacov musíme túto pôžičku splácať.
Predpokladajme, že splácame iba jednu presne dohodnutú splátku za mesiac. Napr. si požičiame 9000 ELD (menová jednotka v onej krajine, nazývanej Eldorádo) a mesačnú splátku si dohodneme na 400 ELD. Potom budeme splácať po 400 ELD 22 mesiacov (koľko je to rokov?) a 23. mesiac doplatíme 200 ELD.

Skúsme si to zapísať schematicky:

```
{VSTUP: POZICKA, SPLATKA}
?
{VÝSTUP: ROKY, MESIACE, POSLEDNA}
```

Na riešenie môžeme využiť celočíselné delenie. Celočíselný podiel po delení pôžičky splátkou dáva počet mesiacov, kedy platíme celú splátku; zvyšok po celočíselnom delení určuje (ak je

nenulový) hodnotu poslednej splátky v nasledujúcom mesiaci. Potom stačí už iba zistiť, koľko je to rokov a mesiacov a výsledok oznámiť.

V programe budeme používať iba päť vyššie spomenutých premenných *POZICKA*, *SPLATKA*, *ROKY*, *MESIACE*, *POSLEDNÁ*. Všetky môžu nadobúdať iba celočíselné hodnoty. Túto skutočnosť oznámime počítaču tak, že ich mená s príslušným typom uvedieme za kľúčové slovo ***var*** (z angl. variable = premenná). Viac premenných rovnakého typu môžeme zapisovať do riadku za sebou oddelené čiarkou.

Od tohoto rozboru je už len krok k programu:

```
Program ELDORADO_01;
var   POZICKA, SPLATKA             : integer;
      ROKY, MESIACE, POSLEDNA      : integer;

Begin
  WriteLn (‘  VYPOCET POCTU ROKOV A MESIACOV PRI SPLATKE
              BEZUROCNEJ POZICKY’);
  Write (‘    Zadaj vysku pozicky a mesacnu splatku ‘); ReadLn
              (POZICKA, SPLATKA);
  MESIACE:= POZICKA div SPLATKA;
  POSLEDNA:= POZICKA mod SPLATKA;
  ROKY:= MESIACE div 12;
  MESIACE:= MESIACE mod 12;
  Write (‘    Pozicku ‘, POZICKA, ‘budes v splatkach ‘, SPLATKA,
              ‘ splacat ‘, ROKY, ‘ rokov, ‘);
  if MESIACE<>0 then Write (MESIACE, ‘ mesiacov ‘);
  if POSLEDNA<>0 then Write (‘ Posledna splatka bude ‘, POSLEDNA);
  WriteLn;
  ReadLn;
End.
```

Výpis je trošku skomplikovaný tým, že splátky môžu (nemusia) vyjsť na celé roky, resp. že posledná splátka je rovná presne dohovorenej splátke (*POZICKA* je násobkom *SPLATKY*).

Predpokladajme, že prvú splátku zaplatíme v danom mesiaci konkrétneho roku. Zostavte program, ktorý určí, v ktorom mesiaci ktorého roku budeme platiť poslednú splátku a akú.

Podobne ako v matematike môžeme aj s celočíselnými hodnotami vykonávať niektoré známe i menej známe funkcie. Sú to predpisy na vykonanie istej činnosti, pričom výsledok tejto činnosti sa stáva hodnotou funkcie. Pre celočíselné hodnoty môžeme použiť tieto funkcie (ich výsledok je celočíselný):

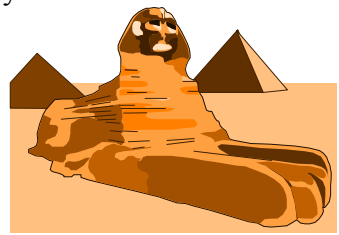
abs(x) – absolútna hodnota *x*,
succ(x) – nasledovník *x* (číslo o 1 vyššie),
pred(x) – predchodca *x* (číslo o jedna nižšie).

V TP sa používajú pre zjednodušenie a sprehľadnenie zápisu aj volania procedúr, ktoré spracúvajú celé čísla a ich výsledkom je celé číslo. Rozdiel oproti funkcii spočíva v tom, že procedúra sa zapisuje ako samostatný príkaz a funkcia ako súčasť výrazu. (Podrobnejšie v časti o podprogramoch.)

Najpoužívanejšie procedúry sú:
Inc(x) zvýšenie hodnoty premennej *x* o jedna (ekvivalent príkazu x:= x+1),
Dec(x) zníženie hodnoty premennej *x* o jedna (ekvivalent príkazu x:= x-1).

Čoho je viac – zrniek piesku alebo hviezd?

Nie som si celkom istý, čoho je viac, ale jedno viem – zrnko piesku asi ťažko nájdeme (štandardne) nalepené na klávesnici počítača. Preto budeme narábať s hviezdičkami, teda symbolmi "*".



V nasledujúcich príkladoch si precvičíme používanie cyklu so známym počtom opakovaní. Ako už vieme, cyklus so známym počtom opakovaní sa zapíše v tvare:

```
for premenná:= hodnota1 to hodnota2 do
begin
 telo cyklu
end
```

kde *premenná* je premenná ordinálneho typu, napr. integer, znak...
hodnota1 je výraz (konštanta), ktorá udáva prvú hodnotu premennej,
hodnota2 je výraz (konštanta), ktorá udáva poslednú hodnotu premennej.
Cyklus *for* sa opakuje pre meniacu sa hodnotu riadiacej premennej cyklu *hodnota2-hodnota1+1* krát. Nemusí sa vykonať ani raz – keď *hodnota1>hodnota2*.
Najčastejšie používame na označenie riadiacich premenných cyklu písmená *i*, *j* – je to "historicky" zaužívané.
Najlepšie bude, keď si cyklus so známym počtom opakovaní ukážeme na konkrétnych príkladoch.

Zostavte program pre zobrazenie N hviezdičiek za sebou v jednom riadku.

Pre zadanú hodnotu N by sme mali zobraziť n-krát hviezdičku a zostať v tom istom riadku. K tomu slúži príkaz *Write('*')*. Pretože telom cyklu je iba jeden príkaz, nemusíme ho zapisovať medzi "zátvorky" begin-end. Výsledkom je program:

```
* * ... ... * *
N hviezdičiek
```

```
Program HVIEZDY_01;
var i, N : integer;

Begin
 ReadLn (N);
 for i:= 1 to N do Write ('*');
 ReadLn;
End.
```

Zostavte program, ktorý pre dané celočíselné hodnoty M a N zobrazí M riadkov pod sebou a v každom bude N hviezdičiek.

Stačí si uvedomiť, že cyklus využitý v predchádzajúcom príklade musíme M-krát zopakovať a vždy po jeho skončení sa presunúť na nový riadok:

```
* * ... * *
*           *
:           :   M
* * ... * *
        N
```

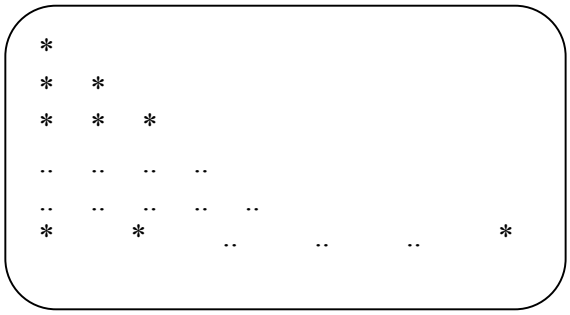
```
Program HVIEZDY_02;
var   i, j, M, N : integer;

Begin
 ReadLn (M, N);
 for i:= 1 to M do          {M riadkov}
```

```
   begin
    for j:= 1 to N do Write ('*');  {N hviezdiciek v riadku}
     WriteLn;                {odriadkovanie na konci riadku}
   end;
  ReadLn;
End.
```

Všimnite si, že cyklus s riadiacou premennou *i* je vložený do vnútra cyklu s riadiacou premennou *j* – medzi jeho "zátvorky" begin-end. To zodpovedá nášmu rozboru riešenia.

Zostavte program, ktorý zobrazí v 1. riadku 1 hviezdičku, v 2. dve, v 3. tri... , v N-tom N hviezdičiek.

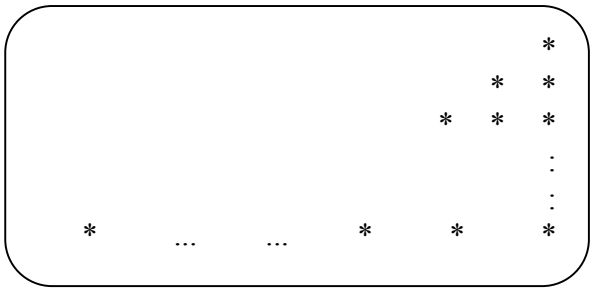
Za základ si znovu môžeme zobrať predchádzajúce príklady. Platí, že v *i*-tom riadku sa má zobraziť *i* hviezdičiek a potom musíme prejsť na nový riadok.



```
Program HVIEZDY_03;
var   i, j, N : integer;

Begin
 ReadLn (N);
 for i:= 1 to N do
  begin
   for j:= 1 to i do Write ('*');
   WriteLn;
  end;
 ReadLn;
End.
```

Zostavte program, ktorý v 1. riadku zobrazí N-1 medzier a 1 hviezdičku, v druhom N-2 medzier a 2 hviezdičky, v i-tom N-i medzier a i hviezdičiek... , v N-tom riadku 0 medzier a N hviezdičiek.

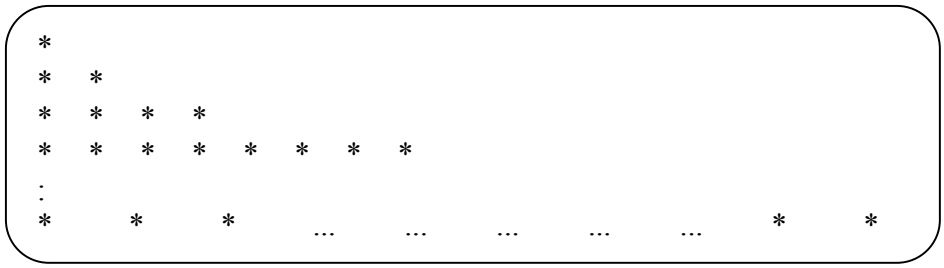
Máme zobraziť N riadkov, v každom najskôr istý počet medzier (cyklus), potom istý počet hviezdičiek (cyklus) a odriadkovať.



```
Program HVIEZDY_04;
var   i, j, N : integer;

Begin
 ReadLn (N);
 for i:= 1 to N do
  begin
   for j:= 1 to N-i do Write (' ');
   for j:= 1 to i do Write ('*');
   WriteLn;
  end;
ReadLn;
End.
```

V poslednom priebehu cyklu s riadiacou premennou *i* bude hodnota *i* rovná N, a preto sa cyklus zobrazenia medzier nezopakuje ani jedenkrát.

Zostavte program na zobrazenie N riadkov takto: V 1. riadku 1 hviezdička, v 2. dve, v 3. štyri, a v každom ďalšom dvojnásobok predchádzajúceho počtu hviezdičiek. Koľko ich bude v N-tom riadku?



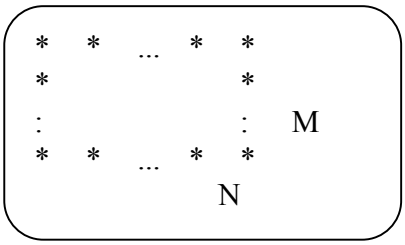
Najskôr napíšeme program:

```
Program HVIEZDY_05;
var   i, j, N, pocet :
            integer;

Begin
 ReadLn (N);
 počet:= 1;
 for i:= 1 to N do
  begin
   for j:= 1 to počet do Write ('*');
   WriteLn;
   počet:= 2 * počet;
  end;
 ReadLn;
End.
```

Do programu sme museli oproti predchádzajúcim doplniť ďalšiu premennú *pocet*, aby sme zabezpečili zdvojnásobovanie počtu hviezdičiek v riadku. Ak by sme chceli vyjadriť matematicky počet hviezdičiek v N-tom riadku – je to 2^{N-1}.

Zostavte program, ktorý zobrazí M riadkov po N znakov tak, že vnútro bude prázdne, hviezdičky budú len na obvode.

Ak chceme využiť cyklus pre M riadkov, musíme rozlišovať dve rôzne situácie: V prvom a M-tom riadku bude N hviezdičiek, v druhom až M-1 riadku bude 1 hviezdička, N-2 medzier a na konci 1 hviezdička. Preto musíme použiť podmienený príkaz, ktorý nám oddelí tieto dva rôzne prípady zobrazenia riadku:




```
Program HVIEZDY_06;
var   I, j, M, N : integer;

Begin
 ReadLn (M, N);
 for i:= 1 to M do
  begin
   if (i=1) or (i=M) then for j:= 1 to N do Write ('*')
   else
    begin
     Write ('*'); for j:= 1 to N-2 do Write (' '); Write('*');
    end;
   WriteLn;
  end;
 ReadLn;
End.
```

Pre úsporu miesta (a názornosť) je cyklus zobrazenia 2. až M-2. riadku napísaný v jednom riadku programu a príkazy sú oddelené bodkočiarkami.

Zostavte program, ktorý vytlačí v M riadkoch a N stĺpcoch "kríž". Jeho tvar si sami určte obrázkom.

Malá násobilka

Aký by to bol počítač, keby nevedel počítať. A jednou z našich najťažších úloh bolo kedysi (a je stále) násobenie dvoch čísiel. V hlave by sme mali mať tabuľku, ktorej sa hovorí malá násobilka (pozri Návšteva z Bilandu). Skúsme využiť počtárskych a zobrazovacích možností počítača k tomu, aby nám takúto tabuľku vypočítal a zobrazil na obrazovke monitora.

Zostavme program, ktorý vo vhodnom tvare zobrazí tabuľku nazývanú malá násobilka.

V peknom tvare môže mať tabuľka tvar:

x	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	10	12	14	16	18
3	0	3	6	9	12	15	18	21	24	27
4	0	4	8	12	16	20	24	28	32	36
5	0	5	10	15	20	25	30	35	40	45
6	0	6	12	18	24	30	36	42	48	54
7	0	7	14	21	28	35	42	49	56	63
8	0	8	16	24	32	40	48	56	64	72
9	0	9	18	27	36	45	54	63	72	81

Odhliadnime od čiar a zamerajme sa iba na obsah – konkrétne čísla a ich umiestnenie v riadku. Potrebujeme zobraziť rôzne veci. Prvý riadok a prvý stĺpec sú trošku rozdielne, vnútro tabuľky získame ako súčin príslušných čísel riadku a stĺpca.

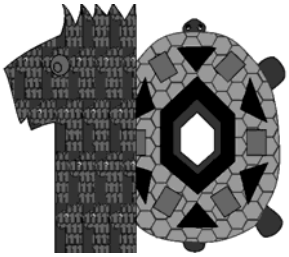
Program môže bez veľkých komentárov vyzerať nasledovne:

```
Program MALA_NASOBILKA;
uses  Crt;   {prístup k nástrojom na prácu s obrazovkou}
const posun = 6;
var   i, j: integer;

Begin
  ClrScr;                  {zmazanie obrazovky - musí byť Crt}
  WriteLn ('Mala nasobilka': 56);
  for i:=1 to 78 do Write('-');    {podčiarknutie nadpisu}
  WriteLn;                         {odriadkovanie}
  WriteLn;
  Write (' ': posun);
  for j:=0 to 9 do Write (j: posun);
                           {horný riadok s činiteľom 0 ..9}
  WriteLn;
  for i:=0 to 9 do         {0. - 9. riadok}
   begin
    Write (i: posun);      {stlpec s druhým činiteľom 0 ..9}
    for j:=0 to 9 do Write (i*j: posun);
                          {vypísanie hodnôt súčinov v riadku}
    WriteLn;               {prechod na nový riadok}
   end;
  WriteLn;
  ReadLn;
End.
```

Program je dostatočne opísaný poznámkami v programe. Snáď len konštanta *posun* si zaslúži pozornosť. Jej hodnota sa používa na "odsadenie" zobrazovanej hodnoty o *posun* miest v riadku doprava. Je použitá ako konštanta preto, aby sme ju nemuseli prepisovať v celom programe; ak vám nevyhovuje, stačí prepísať hodnotu 6 iba v riadku definovania konštanty.

Upravte program tak, aby v ňom boli doplnené aj "čiary", ktoré oddeľujú opis tabuľky – hodnoty činiteľov v hornom riadku a ľavom stĺpci od vypočítavaných súčinov – tabuľku vhodne "olemujte". Použiť sa dajú rôzne znaky.

Návšteva z Bilandu



Nech sa tomu bránime, ako vládzeme, okolo nás vzduchom okrem iného poletujú celé kŕdle čísiel; v škole dostávame známky alebo body, iní berú a míňajú výplatu, každý deň má svoj dátum, každý okamih svoje časové dimenzie... K zápisu čísel v dnešnej podobe sa ľudstvo prepracovávalo stáročia. Všetci vieme, že dnes vládne (až na výnimky) svetom desiatková číselná sústava. Má zaujímavú históriu: Vznikla veľmi dávno v Číne, Indovia k nej pridali nulu, Arabi ju priniesli do Európy niekedy v 11. storočí (preto sa čísliciam hovorí arabské). Je to ozaj to najlepšie, čo sme mohli mať? Otázka, nad ktorou sa zamýšľajú zvyčajne len tí, ktorých baví (a majú čas či potrebu) rozmýšľať.

Svojho času nás navštívili mimozemšťania z planéty Biland. Mali dva prsty na rukách aj nohách a veľmi sa čudovali tomu, ako si vieme komplikovať život. Ich argumenty stoja za zamyslenie:

1. Prečo sa musíme učiť písať číslice 0, 1..., 9, keď im stačia dve číslice 0 a 1? Nimi dokážu zapísať ľubovoľné číslo.
2. Prečo trápime to najdrahšie, čo máme - naše deti - zložitými výpočtami? Veď na zvládnutie sčítania a násobenia im do hláv "nabíjame" tabuľky 10 x 10 výsledkov, ktoré musia vedieť aj o polnoci. Veď poznáte násobilku – je už uvedená v predchádzajúcom príklade.

Pritom Bilanďanom stačí takáto tabuľka na násobenie:

x	0	1
0	0	0
1	0	1

Je to značný rozdiel, však?

3. "Navyše ste pokazili počítače, ktoré síce vo vnútri počítajú s dvojkovými číslicami, ale navonok sa musia tváriť, že počítajú v desiatkovej číselnej sústave," bola ďalšia pripomienka.

Je zrejmé, že v mnohom majú pravdu. Na našu obranu môže slúžiť iba jediný, ale zato dosť podstatný argument: Predstavte si, že rok 1998 by sa zapisoval v tvare 11111001110! Kto si to má zapamätať?!

Skúste napísať program, ktorý bude vykonávať prevody čísel z desiatkovej do dvojkovej číselnej sústavy.

Pre dané prirodzené číslo v desiatkovej číselnej sústave máme zistiť jeho hodnotu ako súčet mocnín dvojky. Obe sústavy sú tzv. polyadické – výsledná hodnota čísla je daná súčtom násobkov príslušných mocnín základu. Napr.

$128_{10} = 1.100+2.10+8.1$, čiže $1.10^2+2.10^1+8.10^0$,

$1101_2 = 1.8+1.4+0.2+1.1$, čiže $1.2^3+1.2^2+0.2^1+1.2^0$.

Nebudeme matematicky zdôvodňovať opodstatnenosť nasledujúceho algoritmu nájdenia prevodu do dvojkovej číselnej sústavy:

Opakovane celočíselne delíme dané číslo číslom 2 a zapamätávame si podiel a zvyšok. V ďalšom kroku delíme predchádzajúci podiel a tak to opakujeme, kým nie je podiel rovný 0. Postupnosť zvyškov, **zapísaná v opačnom poradí**, ako sme ich získali ("odspodu"), je prevodom daného čísla do dvojkovej číselnej sústavy.

Príklad aj so spôsobom zápisu*:

133 : 2 =

P	ZV
66	1
33	0
16	1
8	0
4	0
2	0
1	0
0	1

$133_{10} = 10000101_2$

Kontrola správnosti: $10000101_2 = 1.\ 128+0.\ 64+0.\ 32+0.\ 16+0.\ 8+1.\ 4+0.\ 2+1.\ 1 = 133_{10}$

Vytvorenie programu by nemalo byť problematické až na "zhromažďovanie" zvyškov a ich usporiadanie v opačnom poradí. Možné sú viaceré prístupy:

a) Vytváranie čísla s opačným poradím číslic – pridávanie príslušnej mocniny 10 dopredu. (Aj keď číslo bude vyzerať ako dvojkové, je v skutočnosti desiatkové obsahujúce iba 0 a 1. Pozor na prekročenie rozsahu prípustných hodnôt!)

b) Vytváranie čísla s "normálnym" poradím zvyškov pridávaných na koniec (predchádzajúcu hodnotu vynásobíme 10 a pripočítame zvyšok). Potom ale musíme číslo "obrátiť", viď problém zrkadla čísla. (Pozor na prekročenie rozsahu prípustných hodnôt!)

c) Ukladanie prevodu do reťazca, ktorý obsahuje iba znaky '0' a '1'. Navonok bude vyzerať ako dvojkové číslo, v skutočnosti je to iba reťazec znakov.

Ja si vyberiem ten najlepší – najľahší prípad (ako všetci učitelia), ostatné vám zostanú na "domácu úlohu":

```
Program PREVOD_10_2;
var   N, P, ZV : word;
      PREVOD : string;
Begin
 Write ('Zadaj N '); ReadLn(N);
 P:= N; ZV:=0; PREVOD:='';
 while P<>0 do
  begin
  ZV:= P mod 2;
  if ZV=0 then PREVOD:= '0'+PREVOD
          else PREVOD:= '1'+PREVOD;
  P:= P div 2;
 end;
 WriteLn(N, '(10)= ', PREVOD, '(2)');
 ReadLn;
```

```
End.
```

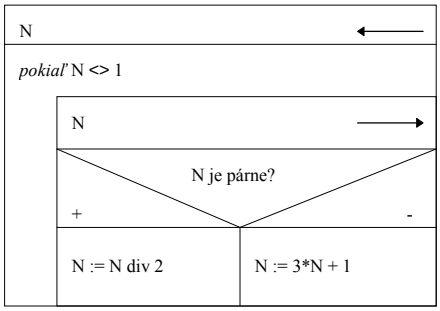
Výhodou tohoto riešenia je možnosť zobraziť (nie ďalej používať) aj veľmi dlhé dvojkové čísla – do 255 číslic, čo nám asi na prevod čísel typu *word* stačí. Orientačne platí, že dvojkové číslo má približne trikrát väčší počet číslic ako zodpovedajúce desiatkové. Prečo?

1. Aký bude výsledok, ak bude zadané N=0? Nie je možné prípadný nedostatok odstrániť?
2. Pozor! Priradenie novej hodnoty premennej P nie je možné predradiť pred výpočet zvyšku. Prečo?
3. Napíšte programy pre ostatné varianty.
4. Podľa mojich vedomostí existuje v TP možnosť prevodu reťazca znakov (ak obsahuje iba číslice) na číslo. Skúste využiť svoj prehľad v nápovede TP a preveriť takýto prístup aj naopak.
5. Podobným spôsobom sa dá urobiť prevod z desiatkovej do ľubovoľnej číselnej sústavy o základe 2, 3, 4 ... 9. Stačí len zmeniť číslo, ktorým delíme. Zostavte program, ktorý pre zadané prirodzené číslo a základ Z sústavy, do ktorej chceme prevod, tento prevod čísla vykoná.
6. Ako by to nestačilo, z desiatkovej číselnej sústavy môžeme previesť číslo aj do sústavy so základom vyšším ako 10?! Známa je napr. šestnástková číselná sústava, ktorá má 16 číslic (tak ako dvojková ma 2, trojková 3, desiatková 10...) 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Zostavte program, ktorý bude uskutočňovať prevod čísla z desiatkovej do šestnástkovej číselnej sústavy.
*7. Vráťme sa k prevodu z desiatkovej do dvojkovej (inej) sústavy. Skúste zostaviť program, ktorý by demonštroval na obrazovke algoritmus zisťovania prevodu tak, ako to bolo ukázané na príklade so spôsobom zápisu, označeným *. Teda pod seba sa budú vypisovať postupne podiely a zvyšky. (Poznámka: Nemusia (môžu) v ňom byť aj grafické prvky.)*

Skončí to niekedy?

Stáva sa nám, že niektoré činnosti sú nekonečné – napr. hodina v škole, keď sa skúša, čas v čakárni u zubára... Dôsledky sú rôzne, ale raz určite skončíme (prinajhoršom utečieme). Nie o všetkých činnostiach sa však dá tvrdiť, že sú konečné, aj keď sa tak javia. Tu je jeden príklad.

Neviem, kto vymyslel takýto postup znázornený v tvare algoritmu. Zrejme to boli matematici.



Ako tento postup dopadne pre N=15? N sa bude postupne meniť takto: 15, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1. Po 18 krokoch sme skončili, pretože sme prišli k číslu 1. Dodnes nie je známe, či existuje také prirodzené číslo N, pre ktoré tento algoritmus neskončí (teda nedosiahne hodnotu 1).

Skúste zostaviť program v pascale, ktorý by zisťoval postupnosť medzivýsledkov uvedeného algoritmu a zisťoval počet krokov – opakovaní cyklu.

Pretože algoritmus je jednoznačne daný, stačí ho iba prepísať do programu. Mali by sme však doplniť počítadlo počtu opakovaní cyklu a upraviť výpis všetkých hodnôt, ktoré nadobúda N.

```
Program SKONCI_TO;
var   N, POCET:     word;

Begin
 Write ('Zadaj prirodzene cislo '); ReadLn (N);
 POCET:= 0; Write (N, ', ');
 while N<>1 do
  begin
   if N mod 2 = 0   then   N:= N div 2
                    else   N:= 3*N+1;
   POCET:= POCET+1;
   Write (N, ', ');
  end;
 WriteLn (' pocet krokov = ', POCET);
 ReadLn;
End
```

Upravte program tak, aby vypisoval v jednom riadku počet opakovaní a za ním spôsob zmeny (párne alebo nepárne) a hodnotu upravenej premennej N.

Celé čísla v problémoch

Počítač by mal plne realizovať naše výpočty - aspoň vtedy, keď spracúvame celé čísla. Ale nie je to pravda! Vyššie uvedené rozsahy celých čísiel sú totiž iba zdanlivo dostatočne veľké.

Zostavte program na výpočet N-faktoriálu.

Postup sme si uvádzali už v časti o algoritmoch (násobenie prvých N prirodzených čísiel). Je veľmi jednoduchý, preto môžeme okamžite napísať program:

```
Program N_FAKT_001;
var   i, N, FAKT:   integer;

Begin
 ReadLn (N);
 FAKT:= 1;
 for i:=1 to N do
  begin
   FAKT:= FAKT*i;
  end;
 WriteLn(N, '! = ', FAKT);
 ReadLn;
End.
```

Program je bezchybný a keď ho spustíme v TP na počítači, zdá sa v poriadku. Počíta však správne iba hodnoty 1! až 7! . Hodnota 8! je totiž rovná 40320 a to prekračuje *MaxInt*. Preto od tejto hodnoty počnúc dáva program nesprávne výsledky, objavujú sa aj záporné hodnoty, čo asi nie je možné.

Odstránenie tohto nedostatku sa zdá jasné: Premennú *FAKT* zvolíme typu *longint*, čo je vyše 2147483647. Ale ani tu sa asi ďaleko nedostaneme. Overme si to na programe, do ktorého "zamontujeme" výpis prvých 40 hodnôt N-faktoriálu:

```
Program N_FAKT_002;
var   i, N:  integer;
      FAKT:  longint;

Begin
 for N:=1 to 40 do
  begin
   FAKT:= 1;
   for i:=1 to N do
    begin
    FAKT:= FAKT*i;
```

```
    end;
    WriteLn(N, '! = ', FAKT);
   end;
  ReadLn;
End.
```

V tomto programe dôjde k rýchlemu zmiznutiu výsledkov prvých faktoriálov – riadky sa rolujú smerom nahor (stránka má 25 riadkov). Upravte program tak, aby po výpise prvých 20 faktoriálov počkal na stlačenie klávesu ENTER (vhodne umiestnite príkaz ReadLn).

Bohužiaľ, ani tento program nám veľa nerieši. Aj keď sa to nezdá, už 13! nie je presný! Jeho hodnota je totiž 6 227 020 800, čo prekračuje rozsah typu longint. Existujú však spôsoby, ako tento problém spracovania veľkých čísiel obísť – pozrite príklad o 100-faktoriále alebo spracovanie veľkých čísiel.

Zoznámte sa s typom comp (podtyp typu real) a vyskúšajte jeho presnosť na našom príklade N-faktoriálu.

5.2 Reálne čísla

Dĺžka, hmotnosť a čas, prípadne iné fyzikálne veličiny, sú bežnými pojmami v našom živote. Výsledkom meraní však väčšinou nie sú celočíselné hodnoty, ale hodnoty reálne (napr. teplota tela 36,5° C, vzdialenosť 3,2 m, hmotnosť školskej tašky 10,48 kg). Ak chceme v pascale vytvárať programy pracujúce s reálnymi hodnotami, musíme mať pre reálne čísla k dispozícii podobné vyjadrovacie prostriedky ako pre prácu s celými číslami.

Typ údajov reálnych čísiel sa v pascale nazýva ***real***, a teda môžeme deklarovať premenné tohoto typu:

```
var v, alfa  : real;
```

Reálne čísla zapisujeme v pascale dvojakým spôsobom. Prvý z nich zodpovedá zvyčajnému desatinnému zápisu reálnych čísiel a skladá sa z celej a desatinnej časti. Tie sú od seba oddelené desatinnou bodkou. Napr.:

```
9.81        -0.234        10001.001     3.14159
```

Druhým spôsobom zápisu reálneho čísla je semilogaritický tvar. Umožňuje jednoduchý zápis hodnôt, u ktorých by desatinný zápis potreboval veľa núl pred alebo za desatinnou bodkou. Semilogaritmický tvar dovoľuje bezprostredne za každé celé alebo reálne číslo v desatinnom tvare zapísať exponent

Exx E++xx alebo *E-xx*

vyjadrujúci násobenie pôvodnej hodnoty mocninou čísla 10 (xx sú číslice) s významom 10^{xx} (prvé dva zápisy) alebo 10^{-xx} (tretí zápis).

Tak sa dajú napr. neprehľadné a dlhé reálne čísla v desatinnom tvare
10^{xx} (prvé dva zápisy) alebo 10^{-xx} (tretí zápis).

Tak sa dajú napr. neprehľadné a dlhé reálne čísla v desatinnom tvare
250000000000, resp. 0. 00000000314
jednoducho zapísať v semilogaritmickom tvare ako

```
25.0E10      2.5E11        0.25E12       25E10  250E9
```

resp.

```
31.4E-10     3.14E-9       0.314E-8      314E-11
```

Ako ďaleko odhodí guľu atlét(-ka), ak sa mu (jej) podarí hod pod uhlom α konštantnou začiatočnou rýchlosťou v? Odpor vzduchu radšej zanedbajme.

Je to klasická úloha z fyziky, preto teoretické odvodenie vzťahu použijeme z učebníc fyziky, kde nájdeme pre vodorovnú vzdialenosť *x* šikmého hodu nahor takýto vzorec $x = (v^2 * \sin(2 * \alpha)) / g$

Pritom platí, že $g = 9{,}81\ m.s^{-2}$, čo je gravitačná konštanta, α – uhol medzi smerom vrhu a vodorovnou rovinou.

Nič by nám nemalo brániť napísať program na takýto pre ručné počítanie nesympatický výpočet. Snáď len to, že okrem hodnôt, ktoré môže nadobúdať daný typ údajov (tu *real*) potrebujeme poznať aj operácie a funkcie, pomocou ktorých tieto hodnoty spracúvame.

S objektmi typu *real* môžeme vykonávať tieto operácie:

+ sčítanie,
- odčítanie,
* násobenie,
/ delenie.

Pascal nemá operátor pre výpočet mocniny x^y, čo mu zazlievame.

Veľmi využívanou súčasťou reálnych výrazov sú *funkcie*. Pascal má pre typ real preddefinovaný celý rad v praxi najčastejšie používaných funkcií. Sú to predpisy pre výpočet hodnoty. Funkcia má svoje meno a môže mať jeden alebo viac parametrov. Tu sú niektoré štandardné (preddefinované) pre typ *real*:

sqrt(x)	druhá odmocnina z *x*,
sqr(x)	druhá mocnina *x*,
sin(x)	sinus *x*,
cos(x)	cosinus *x*,
arctan(x)	arcustangens *x*,
ln(x)	prirodzený logaritmus *x*,
exp(x)	e^x,
abs(x)	absolútna hodnota *x*,

Zápis funkcií v pascale je pre nás prirodzený, t. j. meno funkcie a za ním argument (argumenty) v zátvorkách (x je reálny výraz). Na rozdiel od matematiky musíme argumenty dávať do zátvoriek vždy. Zápis funkcie vo výraze nazývame aj *volanie funkcie*.

Výraz pre výpočet vodorovnej vzdialenosti šikmého vrhu nahor bude potom mať tento tvar:

```
sqr(V) * sin (2*ALFA) / 9.81
```

Uhly musia byť zadávané a počítané v radiánoch, nie v stupňoch. Vzájomný prevod je jednoduchý: Ak S je hodnota uhla v stupňoch ako desatinné číslo (nie v tvare stupne, minúty, sekundy) a R v radiánoch potom platí

```
S/180 = R/3.141592654
```

Rozsah hodnôt údajového typu *real* je ohraničený vlastnosťami počítača a my budeme predpokladať, že je pre naše potreby dostačujúci. Vzhľadom k obmedzenému priestoru pre zobrazenie reálnej hodnoty v pamäti počítača dochádza často k zaokrúhľovaniu reálnych hodnôt na posledných platných miestach.

Vstup reálnych hodnôt sa vykonáva rovnakým príkazom *Read*, resp. *ReadLn* ako u celých čísiel, iba parametre príkazu sú premenné typu *real*. Hodnoty sú na vstup zapisované rovnako ako bolo uvedené vyššie. Do reálnych premenných sa môžu načítavať aj celočíselné hodnoty.

Neformátovaný výstup reálnych hodnôt sa zadáva podobným spôsobom ako u celých čísiel, t. j. príkazom *Write* alebo *WriteLn*. Ak je ich parametrom reálny výraz, je na výstupnom zariadení zobrazené reálne číslo v semilogaritmickom tvare. Počet znakov je daný vlastnosťami prekladača. Pre štandardnú dĺžku neformátovaného tvaru (napr. 16 znakov) by mal výstup hodnoty gravitačnej konštanty mať tvar 9. 810000000E+00.

Podobne ako u celých čísiel môžeme i pri reálnych zadávať formát výstupných hodnôt, a to ako pre desatinný, tak aj pre semilogaritmický tvar.

Formát semilogaritmického tvaru je rovnaký ako pre desatinné čísla, t. j.

```
WriteLn (výraz : počet znakov)
```

Dvojbodkou je od reálneho výrazu oddelený výraz celočíselný, ktorý udáva počet znakov, koľko sa vypíše vrátane miesta pre znamienko, desatinnú bodku, znak E, znamienka exponentu a exponentu samého. Exponent sa zvyčajne zobrazuje dvomi platnými číslicami. Ak je hodnota výrazu kladná, je prvým zobrazeným znakom medzera (nie "+"), ak je hodnota záporná, je prvým zobrazeným znakom "-".

Formát desatinného čísla je zložitejší:

```
WriteLn (výraz : počet znakov : dĺžka desatinnej časti)
```

Počet znakov je podobne celé číslo, ktoré udáva celkovú dĺžku výpisu. Dĺžka desatinnej časti udáva počet vypisovaných desatinných miest. Ak je počet znakov väčší ako dĺžka zobrazovaného čísla, doplnia sa zľava medzery.

Program na výpočet dĺžky vodorovnej vzdialenosti šikmého hodu nahor môže mať teda tvar:

```
Program SIKMY_HOD_NAHOR;
var   V, ALFA, DLZKA       : real;

Begin
  WriteLn (' VYPOCET VODOROVNEJ VZDIALENOSTI SIKMEHO VRHU
            NAHOR');
  Write ('   Zadaj rychlost a uhol v stupnoch ako desatinne
            cisla ');
  ReadLn (V, ALFA);
  ALFA:= (ALFA*3. 141596536) / 180;
  DLZKA:= sqr (V) * sin (2*ALFA) / 9. 81;
  WriteLn ('Dlzka hodu je ', DLZKA: 7: 2, ' m');
  ReadLn;
End.
```

Zostavte program na prevod stupňov Celzia na stupne Fahrenheita a naopak.

Predovšetkým v Spojených štátoch amerických sa ešte aj dnes používajú pri meraní teploty stupne Fahrenheita. Odlišujú sa od v Európe bežne používaných stupňov Celzia takto:

1 °F = (5/9) °C; 32 °F = 0 °C; 212 °F = 100 °C.

Úlohou je zostaviť program prevodu – pre zadanú hodnotu C (°C) zistiť hodnotu F (°F).

```
Program PREVOD_CELZIUS_FAHRENHEIT;
var   F, C   : real;

Begin
 WriteLn ('PREVOD STUPNOV CELZIA NA STUPNE FAHRENHEITA');
 Write ('Zadaj stupne Celzia '); ReadLn (C);
 F:= C*(9/5)+32;
```

```
 WriteLn (C, ' C = ', F: 5: 1, ' F');
 ReadLn;
End.
```

1. Upravte program tak, aby počítal aj prevod stupňov Fahrenheita na stupne Celzia.
2. 100 °F je približne rovné teplote ľudského tela. Zistite, o akú hodnotu v °C ide.

Napíšte program na prevod času zadaného v hodinách, minútach a sekundách na desatinné číslo udávajúce zodpovedajúci počet minút.

Každá hodina má 60 minút, každá minúta 60 sekúnd. Stačí zostaviť vhodný výraz, ktorý bude reálnym číslom. S riešením by preto nemali byť zvláštne problémy:

```
Program PREVOD_H_M_S_01;
var   H, M, S, CAS  : real;

Begin
 WriteLn ('PREVOD CASU Z HODIN, MINUT, SEKUND NA MINUTY');
 Write ('Zadaj hod, min, s '); ReadLn (H, M, S);
 CAS:= H*60 + M + S/60;
 WriteLn (' Cas v minutach = ', CAS: 8: 3); ReadLn;
End.
```

Napíšte program, ktorý bude čas prevádzať na hodiny a ich desatinné časti.
Podobne ako typ *integer* je aj typ *real* v TP rozšírený o viaceré špeciálne "podtypy":

real	$2.9*10^{-39}$	až	$1.7*10^{+38}$	(zaberá 6 bajtov)
single	$1.5*10^{-45}$	až	$3.4*10^{+38}$	(zaberá 4 bajty)
double	$5.0*10^{-324}$	až	$1.7*10^{+308}$	(zaberá 8 bajtov)
extended	$3.4*10^{-4932}$	až	$1.1*10^{+4932}$	(zaberá 10 bajtov)
comp	$-2^{-63}+1$	až	$2^{+63}-1$	(zaberá 8 bajtov)

Základný typ *real* môže mať maximálne 11 miest mantisy, typ *single* môže mať maximálne 7 miest mantisy, typ *double* 15 miest mantisy, typ *extended* 19 miest mantisy. Typ *comp* umožňuje spracovávať iba celé čísla do maximálneho počtu 19 miest.

5.3 Znaky

Jednotlivé znaky môžu byť uchovávané a spracúvané pomocou údajového typu ***char***. Vyjadrujú sa zobraziteľnou reprezentáciou ohraničenou apostrofmi, napr. 'A' alebo 'm'. Inou možnosťou je vyjadrovanie znakov pomocou ASCII kódu znaku, ktorému musí predchádzať znak #. Pri práci s typom znak si musíme uvedomiť rozdiel medzi ASCII reprezentáciou malých a veľkých písmen. Napr. #65 alebo 'A' je vyjadrenie písmena A.

Znaky s kódmi 0–31 vrátane sa nazývajú riadiace znaky, pretože sa pôvodne používali k riadeniu operácií ďalekopisov. Teraz sa používajú pre riadenie niektorých funkcií výstupného zariadenia.

Tabuľka kódu ASCII, ktorý je dnes už celosvetovým štandardom kódovania znakov, má tvar:

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*30*			medzera	!
"	#	$	%	&	'	*40*	(	)	*	+	,	-	.	/
0	1	*50*	2	3	4	5	6	7	8	9	:	;	*60*	<
=	>	?	@	A	B	C	D	E	*70*	F	G	H	I	J
K	L	M	N	O	*80*	P	Q	R	S	T	U	V	W	X
Y	*90*	Z	[	\	]	^	_	'	a	b	c	*100*	d	e
f	g	h	i	j	k	l	m	*110*	n	o	p	q	r	s
t	u	v	w	*120*	x	y	z	{	|	}	~	del		

Znaky s kódom 0–31 sú riadiace, kód znaku v tabuľke je súčtom čísel v riadku a stĺpci.

Zostavte program, ktorý bude pre zadaný znak z klávesnice udávať jemu zodpovedajúci kód.

Typ *char* je ordinálnym typom (pozri ďalej), preto zodpovedajúci kód získame využitím funkcie *Ord* (z angl. "order" = poradie):

```
Program PREVOD_ZNAK_KOD;
var   ZNAK:  char;
      KOD:   byte;

Begin
 ReadLn (ZNAK);
 WriteLn ('Znak ''', ZNAK, ''(ma kod ', Ord(ZNAK));
 ReadLn;
End.
```

Znak sa zo vstupu nezadáva v apostrofoch, tie si doplní program sám.

Predchádzajúci program nie je veľmi užívateľsky príjemný, vždy ho musíme odznova spúšťať. Upravte program tak, aby vypisoval znaky a ich kódy, kým nezadáme nejaký vopred zvolený znak (napr. písmeno 'k' alebo 'K') pre ukončenie.

V tomto prípade je vhodné použiť cyklus s podmienkou na konci, pracovne v pascale nazývaný repeat-until. Zadávanie a výpis znakov a ich kódov sa bude opakovať dovtedy, kým hodnota znaku nebude '*k*' alebo '*K*'. Je jasné, že tieto znaky sú rôzne. Môžeme hneď písať program:

```
Program PREVOD_ZNAK_KOD_REPEAT;
var   ZNAK:  char;
      KOD:   byte;

Begin
 repeat
  ReadLn (ZNAK);
  WriteLn ('Znak ''', ZNAK, ''' ma kod ', Ord(ZNAK));
 until (ZNAK='k') or (ZNAK='K');
End.
```

Ak chceme vypísať na obrazovku aj apostrof ako znak, musíme ho "zdvojiť", resp. "ztrojiť" (ak je na začiatku alebo konci vypisovaného reťazca, ako v programe). Podmienka ukončenia cyklu sa musí v tomto prípade kombinovať pomocou logickej operácie *or* ("alebo"). V TP je možné združiť niektoré podmienky do jednej – buď pomocou vytvorenia vlastnej logickej funkcie (podrobnejšie v časti o podprogramoch), alebo využitím niektorých špeciálnych preddefinovaných funkcií. Napr. máme možnosť previesť malé písmeno na veľké. Slúži k tomu funkcia *UpCase*. Jej výsledkom je zmena malého písmena na veľké, iné znaky sa nemenia. Mohli by sme teda využiť príkaz *ZNAK:= UpCase(ZNAK)*, ktorý zmení premennú *ZNAK* (ak obsahuje malé písmeno) na veľké písmeno. Výsledkom týchto úvah je program:

```
Program PREVOD_ZNAK_KOD_REPEAT;
var   ZNAK:  char;
      KOD:   byte;

Begin
 repeat
  ReadLn (ZNAK);
  WriteLn ('Znak ''', ZNAK, ''' ma kod ', Ord(ZNAK));
  ZNAK:= UpCase(ZNAK);
 until ZNAK='K';
End.
```

Vidíme, že zápis podmienky ukončenia cyklu sa podstatne zjednodušil. A používateľ navonok vôbec nevie, že sme ním zadaný znak upravovali.

Chceme získať opačný výsledok: Pre dané kódy vypísať zodpovedajúce znaky.

Zjednodušme úlohu tak, aby si používateľ zadal interval kódov, z ktorého chce vypísať pre dané čísla príslušné znaky. Na zmenu kódu (čísla) na znak slúži v pascale štandardná funkcia *Chr* (z angl. "character" = znak). Algoritmus je s malou obmenou podobný ako v predchádzajúcom príklade. Tu je program:

```
Program PREVOD_ZNAK_KOD_REPEAT;
var   ZNAK:                char;
      KOD, OD, PO:         byte;

Begin
 ReadLn (OD, PO);
 for KOD:=OD to PO do
  begin
   WriteLn ('Kod =', KOD, ' , znak = ''', Chr(KOD), '''');
  end;
End.
```

Premenná *ZNAK* v tomto prípade nie je použitá. Malo by sa tiež ošetriť prípadné zobrazenie viacerých riadkov, ako je na obrazovke. Navyše sa pri použití ľubovoľného intervalu môžu diať nejaké čudné veci – napr. pri použití riadiacich znakov.

Zostavte program, ktorý bude zobrazovať tabuľku príslušných znakov a ich kódov podľa vzoru tabuľky uvedenej na začiatku tejto časti.

5.4 Reťazce znakov

V štandardnom pascale sa údajový typ *reťazec* nenachádzal. Pretože sa však s reťazcami znakov (napr. slovami) pracuje veľmi často, v TP bol zavedený údajový typ reťazec ***string***. Reťazce znakov sú postupnosti znakov ohraničené apostrofmi. Maximálna dĺžka reťazca je 255. Napr. '*Ahoj123*' je reťazec znakov dĺžky 7. V pamäti počítača je reťazec uložený v toľkých bajtoch, koľko znakov reťazec obsahuje plus jeden. V nultom bajte reťazca je uložená skutočná dĺžka reťazca, ktorá udáva, koľko znakov reťazec obsahuje - počíta sa od prvého (nie od nultého). Keď deklarujeme premennú typu reťazec (*string*) s určenou dĺžkou a skutočný počet znakov reťazca je menší, pamäť je vyhradená pre deklarovanú dĺžku reťazca, ale platné sú iba naplnené znaky. V programe v TP môžeme použiť aj všeobecnú deklaráciu typu *string* s uvedením maximálnej dĺžky (*string*[255]) alebo bez uvedenia dĺžky.

S reťazcami môžeme v programoch pracovať ako s poľami znakov, nultý bajt (prvok) poľa vždy obsahuje údaj, ktorý vyjadruje skutočnú dĺžku reťazca.

Veľmi často sa v programoch používa tzv. prázdny reťazec, ktorý medzi apostrofmi neobsahuje nič. Pokiaľ chceme do reťazca vložiť apostrof, musíme uviesť dva bezprostredne za sebou nasledujúce apostrofy. Do reťazcov sa môžu vkladať aj riadiace znaky vyjadrené ich ASCII hodnotou, ktorej predchádza znak #.

Medzi typom reťazec a typom znak je zabezpečená kompatibilita, pokiaľ je dĺžka reťazca nenulová. S reťazcom je možné v programe pracovať ako s poľom znakov (pozri časť o štruktúre údajov pole). Index poľa určuje pozíciu znaku v reťazci. Rozmer poľa sa počíta od nuly, v nultom znaku je uložená aktuálna dĺžka reťazca. Pokiaľ do reťazca vkladáme znaky ako do poľa znakov, nesmieme zabudnúť aktualizovať znak s indexom nula. Pokiaľ naplníme reťazec príkazom priradenia, nultý znak sa aktualizuje automaticky po naplnení.

Napr. deklarácia

```
var     s, r    : string [10];
        retaz   : string;
```

určuje vytvorenie dvoch reťazcov znakov *r*, *s* dĺžky 10 a jedného reťazca *retaz* dĺžky maximálne 255 znakov.

Prehľad procedúr a funkcií pre spracovanie reťazcov

Bez zvláštnych komentárov si tu uvedieme nástroje na spracovanie reťazcov znakov. Ich použitie je ukázané v zložitejších príkladoch jednak v tejto knihe, jednak v knihe Turbo pascal II.

Length	funkcia, ktorá zistí aktuálnu dĺžku reťazca,
Concat	funkcia, ktorá spojí za seba dva reťazce (“sčíta ich”),
Copy	funkcia, ktorá vyberie (skopíruje) podreťazec z reťazca,
Delete	procedúra, ktorá zruší podreťazec v reťazci,
Insert	procedúra, ktorá vloží podreťazec do reťazca,
Pos	funkcia, ktorá zistí pozíciu podreťazca v reťazci,
Str	procedúra, ktorá prevedie číslo na reťazec,
Val	procedúra, ktorá prevedie reťazec na číslo.

5.5 Logický typ

Údajový typ ***boolean*** sa využíva pre vyjadrenie logických hodnôt a má iba dve možné hodnoty:

true	pre stav pravda,
false	pre stav nepravda.

Údajovému typu *boolean* môžeme v programe priradiť buď konkrétnu hodnotu *true* alebo *false*, prípadne výsledok vyhodnotenia logického výrazu. Logické výrazy nájdu uplatnenie predovšetkým v podmienených príkazoch, príkazoch cyklu apod.

Veľmi dôležité je použitie logických hodnôt a výrazov pri vyhodnocovaní podmienok podmienených príkazov a podmienok pre ukončenie (opakovanie) cyklov.

Priorita operátorov výrazov rôznych údajových typov:

Priorita	***Operátor***
4	not
3	*, /, div, mod, and
2	+, -, or
1	=, <>, >, <, <=, >=

Operácie and, or, not:

X	**Y**	**X and Y**	**X or Y**	**not(X)**
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

Logický typ sa využíva predovšetkým pri vyhodnocovaní podmienok v podmienených príkazoch a cykloch. Jeho využitie môže značne sprehľadniť zápis zložitých podmienok. Uveďme si iba rámcovo takýto prípad – časť programu:

```
:
usporiadane:= (A <= B) and (B <= C);
:
```

```
if usporadane then ...
```

Ďalšie príklady využitia logického typu nájdete v zbierke zložitejších úloh v ďalšej časti.

5.6 Ordinálne typy údajov – interval a vymenovaný typ

Ordinálne typy údajov patria medzi jednoduché typy. Všeobecne jednoduché typy údajov definujú usporiadanie množiny hodnôt. Ordinálne typy majú naviac tieto vlastnosti:

– hodnoty ordinálneho typu sú usporiadanou množinou hodnôt,

– každá hodnota je spojená s ordinalitou, čo je jedinečná celočíselná hodnota, ktorá vyjadruje umiestnenie prvku v množine,

– prvá hodnota každého ordinálneho typu je 0, výnimku tvorí typ *integer*, ďalšia hodnota ordinálneho typu je 1 atď.,

– každá hodnota ordinálneho typu má svojho predchodcu (s výnimkou prvej) a svojho nasledovníka (okrem poslednej),

– na zistenie každej hodnoty ordinálneho sa môže použiť funkcia *Ord*, ktorá vráti ordinalitu prvku typu,

– na zistenie hodnoty predchodcu ordinálneho typu sa môže použiť funkcia *Pred*,

– na zistenie hodnoty nasledovníka ordinálneho typu sa môže použiť funkcia *Succ*.

Turbo pascal má sedem predefinovaných ordinálnych typov:

integer shortint longint byte word boolean char

Okrem toho je možné použiť ďalšie, používateľom vytvorené typy – typ ***interval*** a ***vymenovaný typ***. Medzi vymenované typy patrí napr. logický typ boolean (hodnoty *True* a *False*).

Medzi ordinálne typy patrí aj typ znak – *char*. Množinu hodnôt typu znak tvoria znaky usporiadané podľa rozšíreného súboru znakov ASCII (pozri časť o type znak). Ordinalita je v súlade s pozíciou znaku v súbore ASCII.

Vymenovaný typ určuje množinu hodnôt vymenovaním identifikátorov (názvov), ktoré tieto hodnoty označujú. Ordinalita identifikátora (názvu) vo vymenovaní je daná jeho polohou v zozname. Prvý identifikátor má hodnotu 0.

Typ ***interval*** je určený rozsahom hodnôt ordinálneho typu. Pri definovaní typu interval udávame najmenšiu a najväčšiu hodnotu v intervale. Obidve hraničné konštanty musia byť rovnakého ordinálneho typu. Hodnota premennej typu interval môže počas priebehu programu nadobúdať iba hodnoty z tohoto intervalu. (Ak je nastavená automatická kontrola rozsahov v IDE TP, počítač nedodržanie rozsahu oznámi prerušením programu.)

Často sa tieto typy údajov zavádzajú už po hlavičke programu v tzv. definičnej časti objektov (typov). Predchádza jej kľúčové slovo ***type***. Za ním uvádzame definície nových (používateľom vytvorených) typov. Takto zadefinované typy sa potom používajú pri deklarácii premenných. Napr.:

```
type roky = 1900..2000;   {definovaný typ interval celých čísel roky}
     pismena = 'a'..'z';  {definovaný typ interval znakov pismena }
     karty = (dolnik, hornik, kral, eso);
                          {definovaný vymenovaný typ karty}
var  r, y: roky;
     znak: pismena;
     karta: karty;
```

5.7 Štruktúrovaný typ pole

V živote bežne pracujeme s postupnosťami nejakých hodnôt – čísiel, mien... Dôležitou charakteristikou je pre ne často poradie, v ktorom sú tieto hodnoty uložené. Pretože táto štruktúra je veľmi používaná, má svoje miesto aj medzi typmi údajov v programovacích jazykoch ako ***jednorozmerné pole***.

Typ *pole* má v pascale pevne stanovený počet zložiek rovnakého typu. Pri jeho zavádzaní sa určuje rozmer poľa a typ jeho zložiek. Rozmer určuje, koľko prvkov sa môže do poľa vložiť, a udáva sa v hranatých zátvorkách. Typ zložky poľa sa určuje identifikátorom typu za spojkou *of*. Napríklad:

```
postupnost: array [1..10]of integer
```

je pole s desiatimi zložkami typu integer

```
meno : array [0..20]of string
```

je pole s 21 zložkami typu reťazec

Rozmer poľa môže byť ľubovoľný interval ordinálneho typu. Tak sa môže stať, že index – spôsob prístupu k určitej zložke poľa – nemusí byť iba kladné číslo, ale aj záporné celé číslo alebo dokonca znaková hodnota.

Napr. *vyskyt : array ['a'..'z'] of word* určuje pole *vyskyt*, zložkami ktorého sú síce čísla, ale prístup k nim určuje index typu znak.

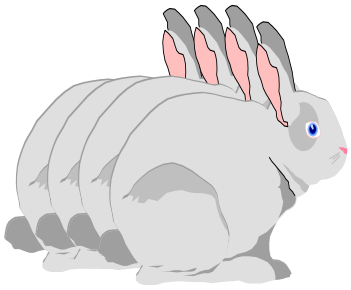
Príslušná zložka (tiež prvok) poľa uložená na *i*-tom mieste má označenie *meno_poľa*[*index*], napr. *postupnost[1]* je prvá zložka poľa *postupnosť*, *meno[i]* je *i*-ta zložka poľa *meno*, *vyskyt['b']* je "*b*-ta" (druhá) zložka poľa *vyskyt*.

Časté je aj využívanie tabuliek rôznych údajov. Pre ich spracovanie používame dvojrozmerné pole. Jeho vytvorenie je podobné ako u jednorozmerného, mení sa iba počet indexov – namiesto jedného (určujúceho poradie v postupnosti) sú indexy dva. Zjednodušene si ich môžeme uvádzať ako riadok a stĺpec. Napr. *tabulka : array[1..5, 1..7] of real* zavádza pole *tabulka*, ktoré má 5 x 7 zložiek typu *real*.

Vhodnejšie je uviesť spracovanie polí na konkrétnych príkladoch.

Králici á la italiano

Taliansky matematik Fibonacci[1] v knihe Liber Abacci ("kniha o súčte") zadal zaujímavú úlohu:



"Koľko párov králikov budeme mať po jednom roku, ak platí: Na začiatku máme jeden pár, každému páru sa po mesiaci narodí vždy ďalší pár a králiky nezomierajú prirodzenou ani násilnou smrťou."

Táto úloha vedie k vzniku postupnosti čísel, ktorým dnes hovoríme *Fibonacciho čísla*. Každé nasledujúce číslo dostaneme ako súčet dvoch predchádzajúcich. Platí teda:

```
F(0)=0, F(1)=1, F(2)=1, F(3)=2, F(4)=3, F(5)=5, F(6)=8, F(7)=13,
F(8)=21, F(9)=34,...
```

Zostavte program pre určenie postupnosti prvých N Fibonacciho čísel.

[1] Leonardo Fibonacci (tiež Leonardo Pizánsky, 1170-1250) - významný taliansky matematik, ktorý sa zaslúžil o rozšírenie desiatkovej číselnej sústavy v Európe, venoval sa hlavne aritmetike a geometrii. Od r. 1963 vychádza časopis Fibonacci Quarterly, v ktorom sa objavuje množstvo zaujímavých aplikácií Fibonacciho čísel.

Pevne si "nasadíme" prvé dve hodnoty 0 a 1. Všetky čísla budeme ukladať do postupnosti – budeme si ich zapamätávať v jednorozmernom poli. Program je jednoduchý:

```
Program FIBONACCI_1; {so zapamätaním}
const max = 100;
var   i, N: word;
      FIB : array [0..max] of word;

Begin
 Write ('Pocitam Fibonacciho cisla. Zadaj počet '); ReadLn(N);
 FIB[0]:=0; FIB[1]:=1;
 for i:= 2 to N do FIB[i]:= FIB[i-1]+ FIB[i-2];
 for i:= 0 to N do WriteLn ('F(', i, ')= ', FIB[i]);
 ReadLn;
End.
```

Pretože čísla budú určite iba nezáporné, zvolili sme typ údajov *word*. Telá cyklov obsahujú iba jeden príkaz, teda môžeme ich písať do riadku priamo za príkaz cyklu bez dvojice begin-end. Výpis sa realizuje po riadkoch, v prípade potreby ho môžete zmeniť na "rozumnejší" – do stĺpcov.

Zostavte program, ktorý určí hodnotu N-tého Fibonacciho čísla.

V tomto prípade si nepotrebujeme zapamätávať hodnoty jednotlivých čísel postupnosti. Použijeme iba jednoduché číselné premenné:

```
Program FIBONACCI_2;       {bez zapamätania, oznámi iba N-té
                           Fibonacciho číslo}
var x, y, z, i, N : word;

Begin
 ReadLn (N); x:= 0; y:= 1,
 for i:= 2 to N do
  begin
   z:= x+y; x:= y; y:= z;
  end;
 WriteLn (N, '-te Fibonacciho cislo= ', y); ReadLn;
End.
```

V premenných *x*, *y*, *z* sa "točia" postupne hodnoty troch po sebe nasledujúcich Fibonacciho čísiel. Vidno to aj z tabuľky sledovania výpočtu pre N = 8:

N	x	y	z	i
7	0	1		
	1	1	1	2
	1	2	2	3
	2	3	3	4
	3	5	5	5
	5	8	8	6
	8	13	13	7
	13	21	21	

N-té Fibonacciho číslo je vlastne N+1-té v postupnosti F(0), F(1)..., F(N). Takže F(8) = 21.

Zovšeobecnenie Fibonacciho radu vymyslel jeden študent matematiky a nazval ho Tribonacciho rad. Už názov naznačuje, že ide o rad, do ktorého sú vkladané tri prvé hodnoty, a až ďalšie sú potom vypočítavané ako súčet troch predchádzajúcich:

```
T(n) = T(n-1) + T(n-2) + T(n-3)
```

Rad narastá rýchlejšie ako Fibonacciho. Zvyčajne sa prvé tri hodnoty rovnajú jednej, ale nie je to podmienkou.

Zostavte program na výpočet členov Tribonacciho radu.

6. PODPROGRAMY

Programovací jazyk pascal umožňuje vytváranie relatívne samostatných častí programov, nazývaných ***podprogramy***. Vďaka nim sa štruktúrovanosť riešenia dostáva na vyššiu kvalitatívnu úroveň a vzniká hierarchia členenia programu, dovoľujúca vytvárať akúsi stavebnicu častí, kombináciou ktorých získame celkové riešenie.

Podprogramy môžeme úspešne použiť v týchto prípadoch:

1. V programe sa vyskytuje úplne rovnaká postupnosť príkazov viackrát na rôznych miestach.
2. Potrebujeme zvýšiť prehľadnosť programu a priblížiť jeho zápis "ľudskému" riešeniu.
3. V programe sa viackrát vyskytuje podobná postupnosť príkazov odlišujúca sa iba parametrami.
4. Riešenie je formulované rekurzívne: Celkové riešenie získame tak, že pojem je opísaný pomocou "samého seba".

Zhrňme si v stručnom prehľade niektoré všeobecné závery pre pascalovské podprogramy:

1. Podprogram je relatívne samostatný čiastočný algoritmus. Spravidla ide o postup, ktorý bude v programe opakovaný viackrát, a to na rôznych miestach príkazovej časti programu.
2. Podprogram sa deklaruje v časti deklarácií a definícií objektov ako posledná deklarácia.
3. Deklarácia podprogramu má podobný tvar ako program. Skladá sa z hlavičky podprogramu a tela podprogramu. Podľa tvaru tela môže mať podprogram dve podoby:

```
<hlavička podprogramu>;          <hlavička podprogramu>;
<deklaračná časť>;
begin                            begin
 <podalgoritmus>                  <podalgoritmus>
end                              end
```

Hlavička podprogramu

a) určuje typ podprogramu – či je funkciou alebo procedúrou (pozri ďalej),
b) dáva podprogramu meno,
c) špecifikuje formálne parametre (určuje ich počet, meno a typ).

Deklarácia svojim miestom v hierarchickej štruktúre programu určuje rozsah platnosti deklarovaného podprogramu. Inak povedané: určuje miesta programu, odkiaľ môže byť daný podprogram volaný. (Ak vytvoríte podprogram v deklaračnej časti iného podprogramu (aj to je možné), budete ho môcť používať len v ňom – pre ostatné podprogramy nebude prístupný.)

V pascale existujú dve formy podprogramov: **procedúry** a **funkcie**.

Funkcia sa používa vtedy, ak potrebujeme zostaviť čiastkový algoritmus, ktorého volaním sa získava funkčná hodnota (podobne ako pri funkciách na kalkulačke, v matematike...). Získaná funkčná hodnota sa ukladá do identifikátora (mena) funkcie. V jej príkazovej časti musí byť teda najmenej jeden priraďovací príkaz, na ľavej strane ktorého je identifikátor funkcie. Vo všeobecnosti má hlavička v deklarácii tvar:

Function *<meno> (<formálne parametre>) : <typ výsledku>*

kde typ výsledku môže byť ľubovoľný jednoduchý typ (a typ ukazovateľ). Typ výsledku musí byť určený identifikátorom typu (napr. pri intervale nie je možné písať priamo 1..10). Funkcia sa volá tzv. zápisom funkcie vo výraze – musí byť teda volaná z pravej strany priradenia alebo ako položka vo výstupnom príkaze... Volaním funkcie prebehne stanovený postup a vypočítaná hodnota "sa vynesie" na miesto volania ako výsledok.

Procedúra predstavuje čiastkový algoritmus, ktorého výsledkom nemusí byť iba jediná hodnota. Procedúra je teda všeobecnejšia ako funkcia. Procedúra má hlavičku:

Procedure *<meno> (<formálne parametre>)*

Volanie procedúry sa zabezpečí zapísaním mena procedúry s uvedením prípadných skutočných parametrov ako samostatného príkazu.

Postup popísaný v podprograme môže pracovať s objektmi troch druhov:
a) *s globálnymi premennými*. Sú to premenné, ktoré sú deklarované vo všetkých "nadradených" (z hľadiska hierarchie) blokoch, ktoré obsahujú tento podprogram.
b) *s lokálnymi premennými*. Lokálne premenné sú tie, ktoré sú deklarované vo vnútri podprogramu a ich platnosť (možnosť získať ich hodnoty) je obmedzená iba na príkazovú časť podprogramu (prípadne ďalších vložených podprogramov). Lokálna premenná v príkazovej časti podprogramu zatieňuje všetky premenné a formálne parametre rovnakého mena deklarované v nadriadených blokoch.
c) *s formálnymi parametrami*. Formálne parametre zastupujú v podprograme triedu objektov, pre ktorú je postup zostavený. V príkazovej časti podprogramu predstavujú objekty, s ktorými príkazy operujú.

Ako formálne parametre sa v deklarácii podprogramu uvádzajú objekty, ktoré chceme pri volaní podprogramu podľa potreby nahrádzať konkrétnymi objektmi v mieste volania.
Súčasťou volania podprogramu sú *skutočné parametre*. Pri volaní sa zadané skutočné parametre dosadia za zodpovedajúce formálne parametre. Postup popísaný na formálnych parametroch sa tak aplikuje na objektoch určených pri volaní ako skutočné parametre. Pre jednotlivé volania podprogramu môžeme dosadzovať rôzne skutočné parametre a tým je postup, zostavený a napísaný jedenkrát, aplikovateľný na rôzne objekty.

Podľa smeru odovzdávaných hodnôt sa v podprograme rozlišujú parametre *vstupné, výstupné* a *vstupno-výstupné*. Vstupné parametre umožňujú odovzdať hodnoty z miesta volania do podprogramu. Výstupné parametre umožňujú odovzdať hodnoty z podprogramu do bloku volania podprogramu. Vstupno-výstupné parametre umožňujú odovzdanie hodnôt obidvomi smermi.
Formálne parametre v deklarácii podprogramu a skutočné parametre pri volaní tohto podprogramu musia korešpondovať – t. j. počet a poradie formálnych a skutočných parametrov musia byť rovnaké a náhrada formálneho parametru skutočným parametrom musí spĺňať pravidlá určené pre jednotlivé druhy parametrov.

Pascal rozlišuje tri druhy parametrov pre údajové typy:
1. parametre volané hodnotou,
2. parametre volané odkazom,
3. konformné pole.
Nie je nič lepšie, ako ukázať si využitie podprogramov na riešení konkrétnych príkladov.

6.1 Procedúra bez parametrov

Napíšte program, ktorý vypíše hviezdičky v tvare

Prvé riešenie by mohlo mať tvar:

```
Program Hviezdy_zaklad;
var   N, i, j : integer;

begin
 Write('Zadaj pocet hviezd pre vypis! '); ReadLn(N);
```

```
*
**
***
…
****…*
****…*
…
***
**
*
```

```
 for i:=1 to N do begin          {rastuci pocet}
  for j:=1 to i do Write('*');
  WriteLn;                       {odriadkovanie}
 end;
 for i:=N downto 1 do begin      {klesajuci pocet}
  for j:=1 to i do Write('*');
  WriteLn;                       {odriadkovanie}
 end;
 ReadLn;
end.
```

Ak sa rozhodneme použiť procedúru pre výpis daného počtu hviezdičiek, získame program:

```
Program Hviezdy_1;
var   N, i, j : integer;
```

```
Procedure Vypis_hviezdicky;
begin
 for j:=1 to i do Write('*');
 WriteLn;                        {odriadkovanie}
end;
```

```
begin                                          {hlavny program}
 Write('Zadaj pocet hviezd pre vypis! '); ReadLn(N);
 for i:=1 to N do
  Vypis_hviezdicky;
 for i:=N downto 1 do
  Vypis_hviezdicky;
 ReadLn;
end.
```

Na prvý pohľad sa vám môže zdať, že sme získali len veľmi málo (dvojicu Write - WriteLn sme nahradili jedným príkazom), no okrem malého ušetrenie miesta program získal na prehľadnosti. Namiesto riadkov s málo hovoriacimi príkazmi máme jednoznačné Vypis_hviezdicky.

Tento spôsob riešenia a procedúry bez akýchkoľvek parametrov sa používajú najčastejšie na "rozdrobenie" problému na stále menšie podproblémy, až kým ich nedokážeme vyriešiť jednoduchým spôsobom (pozri *Konečne si prečítam noviny*).

Všetky premenné sú globálne, počet hviezdičiek sa do procedúry prenáša cez premennú *i*.

6.2 Procedúra s parametrom volaným hodnotou

Riešte predchádzajúci problém pomocou procedúry s parametrom udávajúcim počet hviezdičiek, ktoré sa majú vykresliť. Na začiatok a na koniec vypíšte plný riadok hviezdičiek (t.j. 80).

```
Program Hviezdy_1;
var   N, j : integer;
```

```
Procedure Vypis_hviezdicky(pocet:integer);
var j:integer;
begin
 for j:=1 to pocet do Write('*');
 WriteLn;                        {odriadkovanie}
end;
```

```
begin                            {hlavny program}
 Write('Zadaj pocet hviezd pre vypis! '); ReadLn(N);
 Vypis_hviezdicky(80);
 for i:=1 to N do
  Vypis_hviezdicky(i);
 for i:=N downto 1 do
  Vypis_hviezdicky(i);
```

```
 Vypis_hviezdicky(80);
 ReadLn;
end.
```

1. V tomto príklade sa objavuje prvýkrát formálny parameter. Je uvedený v hlavičke procedúry (*pocet*). Pri volaní procedúry musí byť hodnota v zátvorke rovnakého typu ako deklarovaný parameter - integer.
2. Po volaní procedúry sa všetky výskyty formálneho parametra (tu *pocet*) nahradia jeho hodnotou. Formálnym parametrom môže byť konkrétna hodnota (*80*) alebo premenná, ktorá hodnotu obsahuje (*i*).
3. V deklarácii procedúry sa objavuje deklarovanie lokálnej premennej *j*. Táto existuje iba počas činnosti procedúry (inde jej hodnota nie je potrebná), po jej skončení je neprístupná.
4. Problémy sa nevyskytnú ani v prípade, ak definujete premennú v procedúre pod menom *i*. Pascal ju po zavolaní procedúry vytvorí a použije, po skončení na ňu zabudne a pokračuje s hodnotou *i* pre hlavný program.

6.3 Procedúra s parametrami volanými odkazom (referenciou)

Napíšte procedúru, ktorá zaokrúhli zadané číslo na určený počet desatinných miest.

Na zaokrúhľovanie čísel používa pascal príkaz *round*, ktorý však dokáže zaokrúhliť len na celé čísla. Ak chceme zaokrúhliť desatinné číslo, použijeme jednoduchý prepočet:

*zaokruhlene:=round(povodne*10^pocet_desatinnych_miest)/10^pocet_desatinnych_miest*

čo napr. pre *2.4569* zaokrúhlené na *2* desatinné miesta vykoná:

1. vynásobí (posunie čiarku o zadaný počet miest) - $2.4569*10^2 = 245.69$,
2. zaokrúhli *round(245.69)=246,*
3. a pre získané číslo posunie desatinnú čiarku nazad - $246/10^2=2.46$.

Použijeme teraz procedúru s dvomi typmi parametrov: prvý parameter *cislo* (je volaný odkazom, o čom hovorí kľúčové slovo *var* v hlavičke procedúry) "prinesie" hodnotu čísla, ktoré chceme zaokrúhliť a zároveň ju aj vynesie; druhý parameter *des_miesta* určí počet desatinných miest. Zatiaľ, čo pri použití formálneho parametra sa každý jeho výskyt nahrádza hodnotou, parameter volaný odkazom sa nahrádza premennou. Procedúra by mohla vyzerať:

```
Procedure Zaokruhli(var cislo:real; des_miesta:integer);
var mocnina:longint;              {mocnina desiatich}
    i:integer;
begin
 mocnina:=1;
 for i:=1 to des_miesta do        {vypocita 10^des_miesta}
   mocnina:=mocnina*10;
 cislo:=round(cislo*mocnina)/mocnina;
end;
```

Volanie procedúry v programe môže vyzerať:

```
  ...
  a:=1.239;
  zaokruhli(a,2);
  ...
```

výsledok bude uložený v premennej `a`.

Ani toto riešenie ešte nie je optimálne.

6.4 Funkcia s parametrami

Riešte predchádzajúcu úlohu za pomoci funkcie.

Výsledkom je hodnota priradená menu funkcie v jej tele:

```
Function Zaokruhli(cislo:real; des_miesta:integer):real;
var mocnina:longint;              {mocnina desiatich}
    i:integer;
begin
 mocnina:=1;
 for i:=1 to des_miesta do              {vypocita 10^des_miesta}
   mocnina:=mocnina*10;
 Zaokruhli:=round(cislo*mocnina)/mocnina;
end;
```

Funkciu možno volať ako

```
vysledok:=zaokruhli(11.2569,3);
```

Toto je optimálne riešenie, ktoré umožňuje priblížiť sa bežnému zápisu používanému človekom. Je súčasne prehľadné aj z hľadiska čitateľnosti hlavného programu.

6.5 Využitie logickej funkcie

Ako ukážku "inteligentného" zápisu programu s využitím logického typu *Boolean* uveďme bez komentárov program na zistenie, či dané prirodzené číslo je prvočíslo.

```
Program TEST_PRVOCISIEL;
var N: integer;
```

```
Function PRVOCISLO(prirodzene: integer): boolean;
{dáva hodnotu true, ak je prirodzene prvocislo, inak false }
var   i:     integer;
begin {Prvocislo}
 if prirodzene=1 then Prvocislo:= false
    else if prirodzene=2 then Prvocislo:= true
            else
             begin {prirodzene>2}
              Prvocislo:= true;
              for i:=2 to prirodzene-1 do
               if prirodzene mod i=0 then Prvocislo:= false;
             end; {prirodzene>2}
end; {Prvocislo}
```

```
Begin {hlavny program}
 WriteLn('Urcim, ci je dane prirodzene cislo prvocislom');
 Write('Zadaj cislo, pre skoncenie zadaj 0 '); ReadLn(N);
 while N>0 do
  begin {while}
   if Prvocislo(N) then WriteLn(N, ' je prvocislo')
      else WriteLn(N, ' nie je prvocislo');
   Write('Zadaj N '); ReadLn(N);
  end; {while}
  WriteLn('Koniec programu': 40);
end.
```

Funkcia, ktorej výsledkom je logická hodnota *true* (pravda), resp. *false* (nepravda), slúži na sprehľadnenie testovania splnenia istej podmienky. V tomto prípade postupuje presne podľa

definície prvočísla: Číslo 1 nie je prvočíslo, 2 je prvočíslo, u čísel väčších skúša deliteľnosť so zvyškom pre čísla 2, ..., N-1.

6.6 Deklarácia podprogramu pomocou poprednej deklarácie *forward*

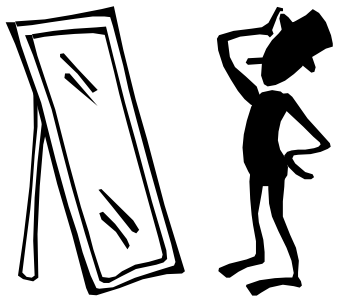
Zostavte program, ktorý prevedie do základného tvaru (vykráti) zlomok A/B.

Na prevod do základného tvaru (čitateľ a menovateľ sú nesúdeliteľné) potrebujeme zistiť najväčšieho spoločného deliteľa čísel A, B. Toho vypočítame pomocou funkcie *NSD*. V programe je použitá tzv. popredná deklarácia: Pretože chceme funkciu *NSD* použiť už pri výpočte krátenia (procedúra *KratZlomok*), je najskôr uvedená jej hlavička, potom za procedúrou hlavička bez parametrov a telo funkcie:

```
Program KRATENIE_ZLOMKOV;
var A, B: integer;

Function NSD(x, y: integer): integer; forward;

Procedure KratZlomok(var citatel, menovatel: integer);
var pomoc: integer;

begin { KratZlomok }
 pomoc:= NSD(citatel, menovatel);
 citatel:= citatel div pomoc;
 menovatel:= menovatel div pomoc;
end; { KratZlomok }

Function NSD; { už bez uvedenia parametrov a typu výsledku}
var pomoc: integer;

begin { NSD }
 while x<>0 do
  begin {while}
   pomoc:= x mod y;
   x:= y;
   y:= pomoc;
  end; {while}
 NSD:= x;
end; { NSD }

Begin { Hlavný program / "ozdoby" si každý dorobí sám! }
 ReadLn(A, B);
 KratZlomok(A, B);
 WriteLn(A, B);
End.
```

6.7 Povedz mi, zrkadielko...

Šofér si na našich cestách užije zábavy dosť. Napriek tomu má niekedy potrebu vymýšľať aj niečo iné ako "myšičky" na ceste. Jednému šoférovi takto tvorivo padol pohľad na tachometer. Bolo na ňom päťmiestne číslo, napr. 23456. "To je pekné číslo," zahundral si. Kedy bude nejaké podobné pekné? Asi až po veľa kilometroch (po koľkých?). "A sú aj iné pekné čísla? Áno, napríklad tie, ktoré pri čítaní odpredu aj odzadu dávajú rovnaký výsledok! Kedy bude najbližšie také číslo?" Pomôžme šoférovi vyriešiť tento problém. Najskôr si ho ale rozmeňme na drobné, teda vyriešme postupne jednotlivé jeho časti.



Zostavte najskôr program, ktorý pre dané prirodzené číslo určí, či toto číslo je "pekné", t. j. pri čítaní odpredu aj odzadu dáva rovnakú hodnotu. Takémuto číslu sa hovorí palindrom.

Mohli by nám napadnúť rôzne varianty. Napr. prístup: porovnať prvú číslicu s poslednou, druhú s predposlednou atď. a ak sa všetky dvojice rovnajú, je číslo palindrom. Ale toto je skôr prístup "ľudský", ktorý považuje číslo za postupnosť číslic.

Zvoľme inú možnosť: Vytvorme k danému číslu jeho "zrkadlo". Je to číselná hodnota, ktorá vznikne postupným "odtrhávaním" číslic z konca čísla a ich pridávaním na začiatok zrkadla. Takto sa to robí, kým nedostaneme z pôvodného čísla nulu. Odtrhávanie číslice zabezpečíme celočíselným delením so zvyškom, vytváranie zrkadla násobením desiatimi predchádzajúceho zrkadla a pridaním práve získanej číslice na koniec. Opakovaným násobením desiatimi sa prvé získané číslice "odsúvajú" smerom doľava.

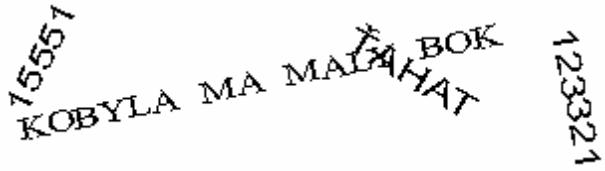
Uveďme si riešenie v tvare funkcie, ktorej parametrom je dané číslo, výsledkom zrkadlo k nemu:

```
Function Zrkadlo(cislo: longint): longint;
var cifra: integer;
begin {Zrkadlo}
 obrathod:= cislo mod 10;
 cislo:= cislo div 10;
 while cislo>0 do
  begin {while}
   cifra:= cislo mod 10;
   cislo:= cislo div 10;
   Obrathod:= (obrathod*10)+cifra;
  end; {while}
 Zrkadlo:= obrathod;
end; {Zrkadlo}
```

Zostavme si tabuľku sledovania výpočtu realizovaného funkciou *Zrkadlo* pre číslo 123.
Funkčná hodnota funkcie *Zrkadlo*(123) sa rovná 321, čo zodpovedá úlohe.

cislo	obrathod	cifra
123	3	
12		2
1	30+2 = 20	
0	320+1 = 321	1

Ak už máme k dispozícii zrkadlo k danému číslu, stačí iba porovnať pôvodné číslo, ktoré sa funkciou nezmenilo (prečo?), so zrkadlo a ak sa rovnajú, je dané číslo zrkadlom.



1. Zostavte program Tachometer podľa uvedeného návodu.
2.Rozšírte riešenie tak, aby zisťovalo, či je dané číslo palindrom pre "ľubovoľne" dlhé číslo. (Návod: Uložte číslice čísla do jednorozmerného poľa a zabezpečte "ľudský" prístup, spomínaný vyššie.)

3. Človek takýmto spôsobom nepostupuje. Pozrie sa na číslice čísla a podľa nejakých pravidiel dokáže určiť, ktoré najbližšie vyššie číslo je palindrom bez toho, aby kontroloval číslo po čísle. Skúste vytvoriť program riešiaci problém Tachometer týmto spôsobom.

Nielen čísla, ale aj ľubovoľné slová môžu byť palindromami, t. j. pri čítaní odpredu aj odzadu dávajú ten istý výsledok – to isté slovo. Napr. "OKO", "RADAR", "ŤAHAŤ". Najkrajší je ten posledný prípad. Hlavne vtedy, keď je nastriekaný farbou na sklenených dverách! Je totiž palindromom nielen zľava-doprava a sprava-doľava ale aj odpredu aj odzadu; je symetrický podľa stredu grafickou podobou. Dokonca aj niektoré vety majú takúto vlastnosť, napr. ak neuvažujeme medzery a dĺžne: "kobyla ma maly bok".

Zostavte program, ktorý pre zadanú postupnosť znakov určí, či je palindrom.

6.8 Niečo o tvare a spôsobe zápisu programu

Skôr, ako sa dostaneme k riešeniu zložitejších problémov, je dobré povedať si niečo o zápise programu. Ako ste si určite všimli, pri zápise programov sú v tejto knihe dodržiavané isté konvencie, teda dohody. Teoreticky môžeme totiž program napísať aj takto a počítač mu bude "rozumieť" a správne ho vykonávať:

```
program F; var i, n, fa: integer; begin readln (n); fa:= 1; for i:=1 to n do fa:=fa*i; writeln(n, fa); readln; end.
```

Horšie to však bude s jeho čitateľnosťou. A neplatí to len pre učenie sa programovaniu, kedy sa na napísaných programoch učíme, ako počítaču správne zadať program a pomocou akých prostriedkov ho donútiť k spolupráci. Dokonca aj autor takéhoto programu sa môže pri jeho vylepšovaní dostať do situácie, kedy už sa v ňom nedokáže orientovať a doplatí na svoju neporiadnosť pri zápise.

Aby sme si nekomplikovali život, uveďme niektoré zásady, ako by mal byť písaný program v pascale.

Základom písania programu je jeho správne rozdelenie na bloky. Blok je logická časť programu ukončená bodkočiarkou, resp. bodkou. (Teda aj celý program je blok.) Často býva blok uzavretý aj pomocou "zátvoriek" begin-end. Každý blok by mal byť v zápise programu viditeľne ohraničený. K tomu nám slúži alebo vhodné odriadkovanie alebo "odsunutie" jeho častí o nejaký počet medzier doprava v riadku. Pretože každý blok môže obsahovať ďalšie bloky, ktoré sú jeho súčasťou, dochádza k postupnému odsúvaniu vnorených podblokov doprava. Medzery prekladač TP ignoruje, a tak sa nič nestane. Koľko ich máme vkladať, je už vec individuálna, nemalo by to byť veľmi veľa, aby sa nám riadky programu zmestili na obrazovku. Riadok u TP môže mať totiž až 128 znakov, ale na obrazovku sa nám ich zmestí iba 80. Vnáranie blokov do seba s odsadením začiatkov môžete vidieť u všetkých tu uvedených programov.

Prehľadne je potrebné narábať aj so spracúvanými objektmi, najčastejšie premennými a metódami ich spracovania – funkciami a procedúrami (podrobnejšie pozri v časti o podprogramoch). Predovšetkým je potrebné odložiť pohodlnosť a objekty označovať tak, aby sme z ich mena zistili ich úlohu. Vhodné skratky dlhých mien sú užitočnou pomôckou. V programoch uvedených v tejto knihe sa okrem toho používajú tieto "finty":

Mená vstupných a výstupných objektov sú tu väčšinou písané veľkými písmenami, napr. *N*, *FAKT* ... , aby sme ich odlíšili od pomocných premenných, ktoré píšeme malými písmenami, napr. *i*, *j*.

Procedúry a funkcie sú v tomto texte (a podobne aj v helpe TP) zapisované tak, že prvé písmená skrátených slov sú veľké, napr. *ReadLn*, *ReadKey*, *Faktorial*, *NnadK*, *Prines*, *Vypis* a pod.

Mená premenných by mali vystihovať ich úlohu v programe. Je to veľmi dôležité pri ďalších úpravách programu, ktoré sa robia prakticky vždy. Prvé, čo by sme mali urobiť, je totiž, aby program bol funkčný a mohli sme otestovať, či naozaj robí to, čo má robiť. Až keď prebehne táto fáza, môžeme ho "vylepšovať" dopĺňaním vhodnej komunikácie s používateľom. Je dobré, ak na začiatku programu volíme konštanty; umožní nám to ich jednoduchú zmenu. Napr. položíme *const max=5*. Ak zistíme, že tento rozsah nie je dostačujúci, zmeníme iba jednu hodnotu na začiatku a zmeny sa prejavia vo všetkých ďalších deklaráciách a celom programe.

Nikdy nepodceňujte vstup, výstup a komunikáciu s užívateľom, ale neberte ho ako alfu a omegu svojej programátorskej činnosti. Kultúrna komunikácia je však základnou podmienkou dobrého programu. Myslite pri jeho písaní aj na používateľa, ktorý sa nemusí v programe (programovaní) tak vyznať ako vy. Každý slušný program by sa mal predovšetkým predstaviť a oznámiť, čo rieši, ďalej vhodne vypýtať požadované vstupné údaje. Mal by tiež primerane oznamovať postup a oznamovať postup a výsledky činnosti. Ale najlepšie bude uviesť (anti-) príklad.

6.9 Medzi Skyllou a Charybdou

Čítali ste Homérove eposy Ilias a Odyssea? Ak nie, máte ešte príležitosť, určite nie sú vypredané. Čítajú ich už vyše dvoch tisíc rokov všetci, ktorí sa chcú zo skúseností iných poučiť tak, aby nerobili tie isté chyby. V Homérovej Odyssei je aj takmer thrillerový príbeh o Skylle a Charybdis.

Odysseovi, jednému z hrdinov trójskej vojny sa domov nechcelo. Preto sa motal po celom Stredomorí, raz sa vysmieval jednookému obrovi Kyklopovi, inokedy si užíval s čarovnou Kirké. Zachcelo sa mu aj kultúry, a preto nasmeroval svoju loď do Messinského prielivu (medzi Talianskom a Sicíliou), ktorý bol známy tým, že v ňom prekrásne spievali morské víly Sirény. Kto počul ich spev, nevedel mu odolať a vrhol sa do mora za nimi. Ten, kto sa náhodou dostal ďalej, nepochodil lepšie – prieliv strážili v tých rokoch dve príšerky nazývané Skylla a Charybdis. Každá z nich mala svoj "stan" na jednej strane úžiny, takže loď, ktorá unikla jednej z nich, stala sa korisťou druhej. Charybdis vytvárala trikrát denne víry (nie vírusy), ktoré vťahovali pod vodu všetko, čo bolo v jej dosahu, potom po konzumácii všetko "vyvrhla" naspäť. Oproti nej stála Skylla – príšera so šiestimi psími hlavami na dlhých krkoch pripomínajúcich dnešné stavebné žeriavy. Ak niekto unikol Charybde, vychutnala si ho Skylla. Stali sa symbolom situácie, z ktorej sa nedá uniknúť ani jednému ani druhému zlu.

Odysseus vyriešil problém nasledovne: Pretože chcel počuť Sirény, dal sa pripútať k sťažňu svojej lode a všetkým námorníkom zalepil uši voskom, aby nič nepočuli. Prikázal im viesť loď stredom úžiny a nebrať jeho výkriky do úvahy. Takto si vypočul prekrásny spev Sirén. Potom dostal loď úspešne cez zvyšok prielivu tak, že ju nasmeroval bližšie k Charybde, a tá si zobrala šesť námorníkov na raňajky.

Ponaučenie? Prípadným nebezpečiam sa nevyhýbaj, ale poriadne sa na ne priprav a vyber si menšie z nich.

Čo to má spoločné s programovaním? Verte tomu, že veľa. Pri tvorbe programov totiž musíme mať na pamäti dve odlišné záležitosti: vnútornú realizáciu programu a jeho komunikáciu s okolím.

Užívateľ programu vôbec nemusí vedieť, čo má byť jeho obsahom a aké má zadávať hodnoty. Proste "surfuje" po adresároch počítača a hľadá spustiteľné súbory. A vtedy môže nájsť a spustiť niečo podobné nasledujúcemu fragmentu programu:

```
Program ABCDEF;
var   a, b, c : real;
Begin
Write ('Zadaj a, b, c '); ReadLn (a, b, c);
...
End.
```

Na obrazovke monitoru sa vypíše oznam *"Zadaj a, b, c"* a kurzor ukazuje na miesto za nimi. Ten, kto nepísal program – obyčajný používateľ a konzument všetkého, čo sa mu podarí na počítači spustiť, môže mať dosť čudné myšlienky, o ktorých programátor ani neuvažoval:

a) Pretože nevie, čo vlastne chce program, napíše poctivo znak po znaku "*a, b, c*" a čaká. Nič sa nedeje a on čaká!

b) Keď už sa dlho nič nedeje, stlačí *ENTER*!

c) Spustí program znovu a zadá s prehľadom "*1, 2, 3*" a už vie, že má stlačiť *ENTER*!

d) Až po príslušnej inštruktáži, že je nemožný, keď nevie, že v pascal sa zadávajú na vstupe údaje oddelené medzerou, zadá "*1 2 3*" a hrdinsky stlačí *ENTER*!

e) Škoda, že nevedel, že *a* má byť väčšie ako *b* a to zase väčšie ako *c*! Jeho chyba, veď je to taký pekný program!

f) A keby ešte vedel, že môže zadávať aj reálne čísla, ale desatinná časť čísla musí byť oddelená bodkou!

g) A keby ešte vedel, čo vlastne tento program robí!...

Uvažujme opačný prípad, kedy sa príliš zameriavame na formu a obsah nie je akosi podstatný:

```
Program TOTO_JE_SUPER;
uses Crt;
var   i,j,k,A,B,C : integer;
      retaz  : string;

Begin
  ClrScr;
  retaz := 'Som Tvoj pocitac, ktory urobi vsetko, co chces!';
  for i:=1 to Length(retaz) do
    begin
      Write(retaz[i]); Sound(2000); Delay(500); NoSound;
    end;
  WriteLn; Write('Zadaj, prosim Ta, dve prirodzene cisla, oddelene medzerou a stlac klaves
  Enter, ktory najdes na klavesnici ');
  ReadLn(A,B);
  C:= A+B;
  WriteLn(A:30,' + ',B:10,' = ',C:10);
  Write('Pre ukoncenie programu stlac Enter!'); ReadLn;
  WriteLn; WriteLn('Velmi ma tesilo, pust si ma niekedy zase!');
  Delay(5000); ClrScr;
End.
```

Zatiaľ nemusíte vedieť, ako presne program pracuje, do konca knihy to však určite zvládnete. Dôležité je všimnúť si predovšetkým počet kurzívou (šikmo) vytlačených riadkov programu. Samé osebe stačia k tomu, aby bol program funkčný. Všetko ostatné je iba komunikácia s užívateľom. Je to síce pritiahnuté za vlasy, ale bohužiaľ, takýchto programov existuje viac, ako by sa mohlo zdať.

7. ZBIERKA (HROMADA) PRÍKLADOV A ÚLOH

V tejto časti si uveďme niektoré riešené, ale aj neriešené úlohy v tvare programov. Pri ich riešení sa budeme snažiť o dôkladný rozbor zadania a o vzorové zapísanie v tvare programu. Pritom však "príkrasy" efektnej (a efektívnej) komunikácie s užívateľom väčšinou neuvádzame a je možné sa im venovať podľa svojho vkusu.

Začnime trochu voľnejšie:

7.1 Pekne sa pozdrav

Zostavte program POZDRAV N-KRÁT inak ako už tu bolo – každý ďalší pozdrav bude na nasledujúcom riadku zobrazený o jednu medzeru ďalej vpravo ako predchádzajúci.

V *i*-tom riadku by sme mohli uvažovať najskôr vypísanie *i* medzier a za nimi pozdrav, napr. Ahoj. Takáto úvaha môže byť hneď realizovateľná v tvare programu:

```
Program Pozdrav_N_krat;
var   i, j, N : integer;

begin
 Write ('Zadaj počet: ')(ReadLn (N);
 for i:= 1 to N do
  begin
   for j:= 1 to i do Write (' ');
   WriteLn ('Ahoj! ');
  end;
 ReadLn;
end.
```

Pretože cyklus s riadiacou premennou *i* obsahuje v tele cyklu dve činnosti, musí mať telo cyklu uzavreté v begin-end. Cyklus pre vypísanie *i* medzier s riadiacou premennou *j* obsahuje v tele iba jeden príkaz, preto tento nemusí byť uzavretý v begin-end.

1. Program má jeden nedostatok, s ktorým sme však rátali: už v 1. riadku je jedna medzera. Upravte program tak, aby v 1. riadku nebola žiadna medzera, v 2. riadku jedna medzera...

2. Skúste upraviť program tak, aby sa pozdrav pohyboval od konca riadku k začiatku. (Zrejme sa bude počet medzier znižovať v závislosti od i.)

3. Ak zadáme väčšie N, od istého momentu sa nám začne program správať "čudne" – pozdrav nejakým čudným spôsobom "preskočí" zase od začiatku. Upravte program tak, aby po tom, čo pozdrav prejde ku koncu (nemusí úplne) riadku, zase sa po medzere "vracal" k ľavému okraju. Potom samozrejme znovu začne pokračovať doprava (akoby sa odrážal od okraja k okraju obrazovky).

7.2 Eratostenovo sito[1]

Aj keď je to zase matematika, venujme sa teraz takejto úlohe: Potrebujeme zistiť prvočísla medzi číslami 1 a N. Načo sú užitočné prvočísla (prirodzené čísla, ktoré sú deliteľné iba jednotkou a sebou samým)? "Predovšetkým, aby sa malo v matematike čo učiť a čím nás trápiť", môže znieť odpoveď. Oblastí využitia je však viac; najzaujímavejšia je tá, že veľké prvočísla sa používajú na kódovanie správ vojakmi a špionážou. Nemenej podstatné je, že deliteľnosť, s ktorou prvočísla úzko súvisia, patrí k základným metematickým metódam skúmania a charakteristík celých čísiel.

Zostavte program na zistenie prvočísiel medzi 1 a N pre zadanú hodnotu N. Úlohu riešte tak, aby všetky prvočísla medzi 1 a N boli uložené v poli.

Sú najmenej dva rôzne algoritmy riešenia:
a) Brať postupne čísla 3,4,...,N a pre každé z nich zistiť, koľko má deliteľov. Ak má práve dvoch (1 a samé seba), potom je prvočíslom a je potrebné ho uložiť do poľa. Takto pokračujeme pre všetky čísla.

b) Použiť tzv. Eratostenovo sito. Je to postup, ktorý môžeme zhrnúť v tomto tvare:

Napíšu sa za sebou všetky prirodzené čísla od 2 po N. Prvočíslo 2 je prvé prvočíslo a vyčiarknu sa všetky jeho vyššie násobky. Prvé nevyčiarknuté číslo je 3, čo je ďalšie prvočíslo. To zostáva a vyčiarknu sa jeho vyššie násobky, pokiaľ už neboli vyčiarknuté (mohli by sa vyčiarknuť "iným smerom"). Takto sa postupuje ďalej a uvažujú sa nevyčiarknuté čísla, pokiaľ sa neprekročí hranica $\sqrt{N}$ (čítaj odmocnina z N). Všetky čísla, ktoré zostanú neprečiarknuté, sú prvočísla.

Pre N=25 je výsledok:

2	3	~~4~~	5	~~6~~	7
~~8~~	9	~~10~~	11	~~12~~	13
~~14~~	15	~~16~~	17	~~18~~	19
~~20~~	~~21~~	~~22~~	23	~~24~~	~~25~~

Skúsme riešiť úlohu využitím možnosti b). Použijeme jednorozmerné pole čísiel s hodnotami indexu od 2 po N, do ktorého vložíme hodnoty 2, ..., N, teda hodnota prvku poľa je rovná hodnote indexu. Ďalej budeme postupne plniť kroky algoritmu. Riešenie môže mať tvar:

```
Program ERAT_SITO_01;
const  MAX = 500;
var    N, i, j, p: word;
       A      : array [2..MAX] of word;

Begin
 ReadLn(N);
 for i:=2 to N do A[i]:=i;
 p:=2;
 while p<= sqrt(N) do
  begin
   for i:=p+1 to N do
    if A[i] mod p = 0 then A[i]:=0;
   repeat p:=p+1 until p<>0; { dalsie nenulove cislo}
  end;
```

[1] Eratostenes z Kyrény (asi 276 až asi 194 pr.n.l) - grécky matematik a astronóm. Z výšky Slnka a zo známej vzdialenosti dvoch miest na tom istom poludníku vypočítal približne obvod Zeme. Zostavil uvedenú metódu na určovanie prvočísiel, známu ako Eratostenovo sito.

```
 {vypis nenulovych cisiel v poli - prvocisiel - doplnit}
End.
```

1. Premenná p slúži na zapamätanie si polohy čísla, ktorého násobky "vyčiarkneme" tak, že ich nahradíme nulami. Trošku je to zamotané s cyklami a podmienkami ich opakovania. Zistite, či je program správny pre všetky možné prípady, ktoré voľbou N môžu nastať. (Hlavne cyklus repeat je "podozrivý"!) Prípadnú chybu, resp. nedostatok pre špeciálne prípady, odstráňte.

2. V programe nie sú uvedené všetky "príkrasy", ale pre vás už nebude problémom doplniť ich.

3. Použitím Eratostenovho sita zostanú v poli aj nuly. Ak by sme potrebovali pole obsahujúce iba prvočísla, musíme ho vytvoriť z poľa A vynechaním núl a "posunutím" ostatných (prvo-)čísel doľava. Skúste to.

4. Pole A môže byť aj poľom logických hodnôt true, false podľa toho, či je číslo charakterizované indexom príslušného prvku prvočíslom alebo nie. Pokúste sa o takéto riešenie.

5. Riešte úlohu nájdenia prvočísel medzi 1 a N s využitím návodu a).

6. Všimnite si pri výpise podobnom ako na obrázku uvedenom vyššie, že prvočísla sa od istého malého čísla môžu nachádzať iba na pozíciach 6k-1 a 6k+1 pre k>=1. Využite túto skúsenosť pri hľadaní!

7. Prvočíselnými dvojičkami sa nazývajú prvočísla, ktorých hodnoty sú od seba "vzdialené o dve"(napr. 5-7, 11-13, 17-19, atď.). Skúste zostaviť program, ktorý bude určovať prvočíselné dvojičky medzi 1 až N. (Nie je dokázané, či ich je nekonečne veľa.)

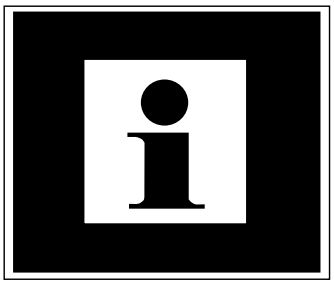
V roku 1983 bolo pomocou počítača určené najväčšie vtedy známe prvočíslo 2^{586243}-1, ktoré má 25962 číslic, v Guinessovej knihe rekordov z roku 1991 sa objavilo 319581.2^{216193}-1. Bez počítačov (samozrejme s iným ako týmto programom) by to nebolo možné.[1]

[1] Ale ľudí netreba podceňovať: V r. 1987 vypočítal "ručne" 45-ročný Japonec Hitachi Tomori za 13 hodín a 6 minút číslo π s presnosťou na 40-tisíc desatinných miest. Už pred ním 23-ročný Ind Radžmán Mahádevanu podobne určil číslo π s presnosťou na 31811 číslic, ale trvalo mu to iba 3 hodiny a 49 minút, čo je v priemere 156,7 číslic za minútu!! A potom, že človek nemôže nahradiť počítače (či naopak?).

7.3 Ktoré písmeno vyhráva?

Rôzne jazyky sa odlišujú okrem iného aj počtom výskytov jednotlivých písmen. Napr. v slovenčine majú najčastejší výskyt v percentách tieto písmená (v prvom stĺpci sú spoluhlásky, v druhom samohlásky):

písm	%	písm	%
n	5,7	a, á	9,7
s	5,0	o, ó	9,1
t	4,9	e, é	8,4
r	4,7	i, í, y, ý	8,4
v	4,7	u, ú	3,1
k	4,0		
l	3,9		
m	3,6		
d	3,4		
p	3,0		



Zostavte program, ktorý bude pre zadaný reťazec znakov zisťovať počet výskytov jednotlivých znakov v tomto reťazci.

Zjednodušme úlohu: Uvažujme, že reťazec bude obsahovať iba malé písmená. Využime spracovanie znakov reťazca a ukladanie počtu ich výskytov do poľa *P*. Jeho indexmi môžu byť znaky, pretože skupina znakov ‘a’..’z’ tvorí kompaktný podinterval množiny znakov. (Kompaktný v tom zmysle, že medzi nimi nie je žiadny iný znak.) Po zistení počtu výskytov vypíšeme tieto počty pre každé písmeno a za tým znázorníme výpis príslušného počtu hviezdičkami (tzv. zjednodušený riadkový histogram).

Najskôr si uveďme návrh programu, ktorý nie je algoritmicky veľmi zložitý:

```
Program VYSKYT_ZNAKOV;
uses  Crt;
var   znak   : ‘a’..’z’;
      ret    : string;
      P      : array [‘a’..’z’] of integer;
      i, r   : integer;

Begin
 Write(‘Zadaj retazec malych pismen a stlac Enter ‘);
 ReadLn (ret);
 for znak:=’a’ to ‘z’ do P[znak]:= 0; {vynulovanie počtov výskytov}
 for i:=1 to Length(ret) do
  begin
   znak:= Copy (ret,i,1);
   Inc (P[znak]);          {zvýšenie počtu výskytov znaku}
  end;
 WriteLn (‘ Pocet vyskytov malych pismen v tvare riadkoveho histogramu: ‘);
 r:= 1;                {pomocná premenná pre výpis riadkov}
 for znak:=’a’ to ‘z’ do
  begin
   r:= r+1;
   Write (‘Znak ‘, znak, ‘ pocet ‘, P[znak], ‘ ‘);
   for i:=1 to P[znak(do Write (‘*’);     {vytlačenie počtu hvizdičiek}
   WriteLn;
   if r mod 15 = 0 then              {pozastavenie programu}
    begin
      Write (‘Pre pokracovanie stlac Enter! ‘); ReadLn;
```

```
    end;
  end;
 ReadLn;
End.
```

Zvýšenie počtu výskytov znaku sa realizuje tak, že zistíme *i*-ty znak reťazca a hodnotu príslušného prvku poľa *P* zvýšime o jedna pomocou procedúry *Copy*. Pri výpise je použitá iba jedna "finta" navyše. Pretože malých písmen je 26 + nadpis, horné znaky s ich početnosťou by rýchlo "vybehli" nahor a neboli by viditeľné. Preto premenná *r* zabezpečuje zastavenie na čas do stlačenia *ENTER*.

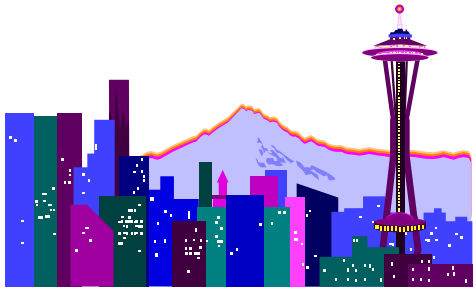
1. Doplňte úlohu tak, aby boli uvažované veľké aj malé písmená a číslice. Výsledky môžete vyjadriť v tvare tabuľky s percentuálnym vyhodnotením počtu výskytov.

2. Doplňte program o zistenie a výpis počtu ľubovoľných znakov z klávesnice. Výsledky prehľadnou formou zobrazte – nemusí to byť riadkový histogram ale napr. stĺpcový alebo tabuľka, ktorá obsahuje percento výskytu jednotlivých znakov.

3. Podobné úvahy mali "jazyčníci" aj pri zisťovaní výskytu slov určitej dĺžky. Zostavte program, ktorý bude zisťovať početnosť výskytu 1-, 2- atď. písmenových slov v danom reťazci znakov. Jednotlivé slová sú od seba oddelené medzerou.

7.4 Svetlá veľkomesta menom unit crt

Predstavte si situáciu, keď z istého odstupu a nadhľadu za peknej noci pozeráte na veľkomesto. Svetlá v ňom sa náhodne rozsvecujú a zhasínajú, majú náhodnú farbu aj polohu. Na takomto princípe je založených viacero tzv. šetričov obrazovky (*screensavers*), ktoré po istom čase nečinnosti počítača "nabiehajú" na obrazovku a zmiznú v momente stlačenia klávesu.



Aké činnosti by sme potrebovali na modelovanie svetiel veľkomesta? Určite zmazanie obrazovky na začiatku, potom do stlačenia ľubovoľného klávesu zobrazovanie nejakého znaku (zase to vyzerá na hviezdičku) v náhodne zvolenej farbe na náhodne zvolenej pozícii. Toto, ako aj ďalšie výhody komfortnejšej obsluhy vstupu aj alfanumerického výstupu nám umožňuje *unit* (jednotka) ***crt***.

Možnosť využívať tzv. *unity* je špecifickou črtou Turbo pascalu. Unity sú samostatne vytvorené (či už výrobcom alebo programátormi) a skompilované programové produkty, ktoré môžu obsahovať často používané nástroje pre tvorbu programov. Ich výhoda spočíva v tom, že tieto nástroje nemusíme vždy znovu a znovu prepisovať do programu, iba si ich "požičiavame" previazaním programu s príslušným unitom. Toto previazanie zabezpečíme uvedením mena potrebného unitu za kľúčovým slovom ***uses***, tak ako to vidno v nasledujúcom programe. Ak používame viac unitov, musia byť ich mená uvedené za *uses*, oddelené čiarkami. Slovo *uses* sa používa vždy hneď za hlavičkou programu.

Unit *Crt* je výrobcom (firmou Borland) dodávaná sada nástrojov na efektívnu (aj efektnú) prácu so vstupom a výstupom v alfanumerickom (textovom) režime. Bez nároku na úplnosť si uveďme niektoré jeho nástroje:

ClrScr – zmazanie obrazovky, vyplní sa farbou podkladu,
TextBackground*(farba)* – nastavenie farby podkladu (0–7),
TextColor*(farba)* – nastavenie farby znaku (0–15),
GoToXY*(X, Y)* – premiestnenie kurzora na súradnice (X, Y)[1],
KeyPressed – funkcia testuje stlačenie nejakého klávesu (logická),
Delay*(cas)* – zastaví program na určený čas (1000 = 1 sekunda)...

S týmito poznatkami už môžeme napísať náš program:

```
Program SVETLA_VELKOMESTA;
uses Crt;
var x, y, farba: integer;

Begin
 SetBackground (black); ClrScr;  {pozadie čierne, zmazanie obrazovky}
 Randomize;
 repeat              {začiatok čakania na stlačenie klávesu}
  x:= Random (24)+1; y:= Random (78)+1;
                          {generovanie náhodných hodnôt}
  farba:= Random (15);
  TextColor (farba);       (nastavenie náhodnej farby znakov}
  GotoXY (x, y); Write('*')
              {presun kurzora do náhodnej pozície a zobrazenie * danej farby}
 until KeyPressed;  {ak bude stlačený nejaký kláves, cyklus končí}
End.
```

Farby, s ktorými môžeme pracovať, sa určujú buď uvedením anglického názvu farby alebo číslom:
Black – 0 (čierna)
Blue – 1 (modrá)
Green – 2 (zelená)
Cyan – 3 (modrozelená)
Red – 4 (červená)
Magenta – 5 (fialová)
Brown – 6 (hnedá)
LightGray – 7 (svetlosivá)
DarkGray – 8 (tmavosivá)
LightBlue – 9 (svetlomodrá)
LightGreen – 10 (svetlozelená)
LightCyan – 11 (svetlomodrozelená)
LightRed – 12 (svetlo–červená)
LightMagenta – 13 (svetlofialová)
Yellow – 14 (žltá)
White – 15 (biela).

[1] Súradnice ľavého horného rohu sú (1,1) a vieme, že máme 25 riadkov (používajte radšej iba 24) po 80 znakoch. Za príkazom GoToXY použitý príkaz Write vypisuje príslušné položky od danej pozície.

Väčšina činností je popísaná komentármi v programe a mali by byť zrejmé. Súradnica *x* je "poddimenzovaná" na 1–24, podobne *y* na 1–78, farba nadobúda hodnoty od 0 po 15. Ak si program spustíte, uvidíte, že sa "veľkomesto" podozrivo rozsvecuje viac a viac. Je to spôsobené skutočnosťou, že nemáme "zhasínanie svetiel". Čiastočne je síce zabezpečené tým, že sa niektorý farebný znak môže premazať čiernou hviezdičkou, ale to je pravdepodobnosť najviac 1:16.

Skúste upraviť program tak, aby približne v jednej tretine generovaní zapísal na dané miesto medzeru. Návod: Buď si zvoľte špeciálny, náhodne generovaný "rozlišovač", alebo skúste zmeniť rozsah farby s nasledujúcim testovaním.

7.5 Chcete byť v televízii?

Kto by nechcel? (Prinajhoršom v relácii "Polícia pátra"). Sľúbiť to nemôžem, ale sami sa môžete postarať aspoň o to, aby sa vaše meno premietalo ako šetrič obrazovky na obrazovke monitoru počítača. Stačí len zostaviť príslušný program.

Zostavme program, ktorý zadané meno (reťazec znakov) bude postupne presúvať v náhodnej farbe a náhodnom riadku pomaly cez obrazovku zľava doprava.

Vstupom do programu bude meno – ľubovoľný reťazec znakov. Po vymazaní obrazovky by sa mal náhodne zvoliť riadok a text sa bude posúvať, kým nedôjde na koniec riadku. Potom sa zvolí znovu náhodný riadok a text sa začne zobrazovať od jeho začiatku. Tieto úvahy vedú k programu:

```
Program MENO_V_POHYBE;
uses  Crt;
const cas = 500;
var   i, x, y, farba, dlzka: integer;
      meno : string;

Begin
 SetBackground (black); TextColor(white);
 Write ('Zadaj meno: '); ReadLn (meno); ClrScr;
 dlzka:= Length(meno);                    {dlzka retazca}
 Randomize; x:= Random (24)+1; y:= 1;
            {generovanie náhodneho riadku 1-24, prvy stlpec}
 repeat              {začiatok čakania na stlačenie klávesu}
  farba:= Random (15)+1;
  TextColor (farba);     {nastavenie náhodnej farby znakov okrem čiernej}
  GotoXY (x, y); Write(meno);
                   {presun kurzora do náhodnej pozície a zobrazenie mena}
  Delay(cas);                          {čakanie stanovený čas}
  TextColor(black); GoToXY(x, y);
  Write(meno);                            (zmazanie mena}
  if y+dlzka 79 then y:= y+1
                   {ak sa reťazec zmestí do konca riadku, tak posun o 1}
               else
                begin x:= Random (24)+1; y:= 1; end;
                   (generovanie iného náhodného riadku 1-24, prvý stĺpec}
 until KeyPressed;  (ak bude stlačený nejaký kláves, cyklus končí}
End.
```

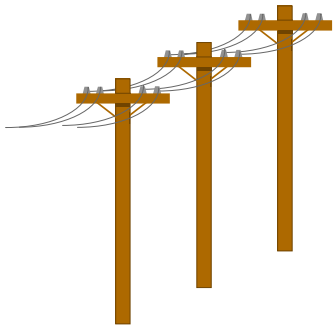
Program je dostatočne popísaný komentármi. Ak si ho spustíte, bude "hrať" všetkými farbami. "Menej je niekedy viac," hovorí ľudová skúsenosť. Zvážte svoje estetické cítenie a upravte program tak, aby mu vyhovoval. Často sa totiž v prvotnom štádiu okúzlenia novými možnosťami stáva, že programy sa len tak "hýria" farbami, "otravujú okolie" neznesiteľnými zvukmi – proste správajú sa ako kohút ráno (t. j. dá svojmu okoliu jasne najavo, že už je hore a že on je tu pánom).

1. Vráťte sa teraz k programu TOTO_JE_SUPER spomínanému v časti o spôsobe zápisu programu a pozrite si úlohu cyklu for. Využite tohoto návodu pre vytvorenie procedúry na výpis reťazca znakov znak po znaku od danej pozície na obrazovke a so zvukovým efektom podobným písaniu na písacom stroji.

2. Zostavte program, ktorý pre zadané meno (reťazec znakov) zo vstupu bude toto zobrazovať na náhodnom mieste na obrazovke a po istom čase ho zmaže a zobrazí niekde inde.

3. Zostavte program, ktorý bude daný reťazec znakov pohybovať po obrazovke "náhodným" smerom. Napr. najskôr sa posúva doprava, potom dole, doľava atď. Možné sú rôzne algoritmy posunu po obrazovke.

7.6 ... - - - ... (Čítaj SOS)



V roku 1844 sa uskutočnil prvý diaľkový prenos správ pomocou telegrafu, ktorý navrhol S. Morse v roku 1837, medzi Washingtonom a Baltimore. Morse tiež zaviedol abecedu z bodiek a čiarok, známu dodnes ako Morseova abeceda. Používala a používa sa najmä pri diaľkovom dorozumievaní sa (nezabudnuteľný je film Přednosta stanice s Vlastom Burianom), dnes napr. pri stretávaní sa lodí na mori, pri tiesňovom volaní a pod.

Morzeovka: Uveďme si iba kombinácie bodiek a čiarok, ktorými sú zakódované písmená. (Existujú aj kódy pre číslice a ďalšie znaky.):

Písmeno	Kód	Písmeno	Kód	Písmeno	Kód	Písmeno	Kód
a	. -	h	....	n	-.	u	..-
b	-...	ch	----	o	---	v	... -
c	-. -.	i	..	p	. --.	w	. --
d	-..	j	. ---	q	--. -	x	-..-
e	.	k	-. -	r	. -.	y	-. --
f	..-.	l	. -..	s	...	z	--..
g	--.	m	--	t	-		

Zostavte program, ktorý bude zo vstupu zadaný text (reťazec písmen) kódovať do postupnosti bodiek a čiarok podľa pravidiel Morseovej abecedy.

Najväčším problémom bude "naučiť" počítač Morseovu abecedu. Na zistenie toho, aký kód majú jednotlivé písmená totiž neexistuje algoritmus, a tak sa musí program kódy "natvrdo naučiť naspamäť". Budeme musieť zaviesť pole kódov *M*, ktoré obsahuje postupne kódy písmen *a..z*. Nepríjemnosti nám môže pri prevode spôsobiť dvojznakové písmeno *ch*, preto ho radšej z našej postupnosti vylúčime. Potom by mal počet prvkov poľa kódov byť 26, a teda zodpovedá postupnosti písmen anglickej abecedy, používanej aj pre počítače. Stačí "iba" zabezpečiť prevod znaku na zodpovedajúci kód uložený v poli *M*.

Ale k tomu môžeme využiť vlastnosti typu údajov *char*. Typ ***char*** je ordinálnym ktorými budú znaky z intervalu '*a*'..'*z*'.

Ďalším problémom je výstup zakódovanej informácie. Napísať iba postupnosť "..-... ----. -..---. " asi nie je najšťastnejším riešením. Potrebujeme nejaký oddeľovač jednotlivých kódov písmen; použime znak "/". Možno by bolo potrebné zaviesť aj oddeľovač slov – medzeru, oddeľovač viet – bodku (stop), ale to už sú ďalšie podmienky na vstup, ktoré zatiaľ nebudeme brať do úvahy.

A čo keby sme výstup na obrazovku sprevádzali zvukovým výstupom, charakteristickým pre morseovku "píííp, pip, píííp... "? Máme síce obmedzené prostriedky výstupu, ale pre tento účel nám vystačia. Použiť môžeme vhodnú kombináciu príkazov *Sound* (zvuk), *NoSound* (nie zvuk) a *Delay* (zastavenie na istý čas). Voľba dĺžok "čiarky" a "bodky", ako aj medzery medzi jednotlivými písmenami je viac menej experimentálna. Preto si ich zvolíme ako konštanty na začiatku programu a v prípade toho, že vám budú veľmi "píliť uši", ich môžete jednoducho upraviť podľa vlastnej potreby. Dokonca by bolo vhodné, aby sa prípadný zvuk dal vypnúť.

Už nám zostáva iba dohovoriť sa, či má program vykonávať iba prevod postupnosti znakov (malých písmen) na zodpovedajúcu správu v Morseovej abecede alebo aj naopak. V opačnom

prípade je problematické jednak zadávanie čiarok a bodiek používateľom, jednak kontrola správnosti vstupu. Preto bude lepšie, keď tu budeme uvažovať iba o prevode reťazca písmen na zodpovedajúcu postupnosť čiarok a bodiek.

```
Program MORSEOVKA;
uses  Crt;
const carka = 700; bodka = 350; medz = 1000; oddel = '/';
var   ret    : string;
      M      : array ['a'..'z'] of string[4];
      dlzka,i: integer;
      znak   : 'a'..'z';

Begin
 M['a']:='.-'; M['b']:='-...'; M['c']:='-.-.'; M['d']:='-..'; M['e']:='.';
 M['f']:='..-.'; M['g']:='--.'; M['h']:='....'; M['i']:='..'; M['j']:='.---';
 M['k']:='-.-'; M['l']:='.-..'; M['m']:='--'; M['n']:='-.'; M['o']:='---';
 M['p']:='.--.'; M['r']:='.-.'; M['s']:='...'; M['t']:='-'; M['u']:='..-';
 M['v']:='...-'; M['w']:='.--'; M['x']:='-..-'; M['y']:='-.--';M['z']:='--..';
 WriteLn ('MORSEOVKA':50);
 Write ('Zadaj spravu pomocou malych pismen '); ReadLn (ret);
 dlzka:= Length (ret);
 for i:=1 to dlzka do
  begin
   znak:= Copy (ret, i, 1);
   for j:=1 to Length(M[znak]) do
    begin
     Sound;
     if Copy(M[znak], j, 1) = '.'  then Delay (bodka)
                                   else Delay (carka);
     NoSound;
    end;
   Write (M[znak], '/'); Delay (medz);
  end;
 WriteLn ('STOP'); ReadLn;
End.
```

Program nie je príliš dokonalý. Medzera medzi slovami chýba. Často sa totiž objavujú vety, v ktorých majú medzery svoje opodstatnenie. Ja poznám takúto: "zvierala som zviera lasom" alebo "mala koza korunu", ale niekedy môže dôjsť ešte k horším prípadom a úplnej zmene významu. Skúste také vety nájsť.

1. Zabezpečte, aby sa spolu so zakódovanou správou vypísala na obrazovke aj prehľadná tabuľka kódov Morseovej abecedy tak, aby si užívateľ mohol skontrolovať správnosť výpisu.

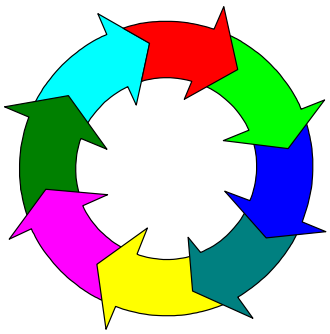
2. V programe nie je zabezpečené zapamätanie si výstupného kódu. Skúste si ho cvične zapamätať vo vhodnom tvare. (Odporúča sa pole, ale aké, to si určte vy.)

3. Doplňte sadu prípustných znakov zo vstupu o medzeru a bodku a ich príslušné zakódovanie do postupnosti kódov. Ďalej zabezpečte, aby iné ako prípustné znaky zo vstupného reťazca neboli alebo uvažované alebo program žiadal nový vstup reťazca. Porozmýšľajte aj o možnosti použitia veľkých písmen – tie totiž majú v Morseovej abecede rovnaké kódy ako malé.

4. Skúste vytvoriť program, ktorý bude podľa potreby aj dekódovať postupnosť "bodiek a čiarok" na text. Zvoľte vhodné podmienky pre vstup a naivne predpokladajte, že sa užívateľ nepomýli (nie je nutná podmienka).

7.7 Vyčítanka

Preč sú časy detských hier. Zostali len nostalgické spomienky na "tie zlaté časy, keď nám bolo krásne" (Elán). Mnohí z vás sa určite hrali hry, v ktorých bol jeden z hráčov vyčlenený na špeciálne účely v dobrom či zlom. Napríklad naháňal alebo hľadal ostatných. Ako určiť obeť? Slúžili k tomu od nepamäti rôzne riekanky – vyčítanky. Všetci hráči sa postavia do kruhu a na koho to slovo padne, ten musí ísť z kola von.



Ale existujú aj iné analogické historické príklady. Či sa naozaj stali, nie je isté, ale sú tiež pekné. Napríklad tento: V židovskej vojne proti Rimanom sa vraj stala takáto udalosť: Rimania obliehali židovskú pevnosť, v ktorej sa statočne bránilo už len niekoľko obrancov. Dali si sľub, že zomrú do posledného muža a nevzdajú sa. Pretože zásoby potravín sa veľmi rýchlo míňali, rozhodli sa, že sa budú "hrať" čudnú vyčítanku: Postavili sa do kruhu a ten, na ktorého padlo isté poradie, sa dobrovoľne vrhol medzi obliehajúcich a... Takto to išlo deň za dňom, až kým neostali iba dvaja – Josephus Flavius a jeho priateľ. Tí už od začiatku plánovali vzdať sa Rimanom, ale báli sa ostatných, aby ich nepredhodili Rimanom ako prvých. Svoj plán vzdať sa nakoniec uskutočnili vďaka tomu, že sa postavili na správne miesto v kruhu.

Predpokladajme, že sa N hráčov postaví do kruhu a je zadané číslo M (možno si ho predstaviť ako počet slabík vyčítanky) hráča, ktorý vypadáva v každom kroku z kruhu. Zostavme program, ktorý vypíše poradie vypadávajúcich hráčov.

Potrebujeme si najskôr "zoradiť" hráčov. K tomu by bolo výhodné jedno-rozmerné pole. To má však lineárne poradie, t. j. prvý a posledný prvok. Nevadí, stačí, ak si zapamätáme, že po poslednom hráčovi máme zase začínať prvým. Ako si označiť hráča? Možností je (ako vždy) viacero, môžu to byť mená, čísla... Náš postup by sme mohli stručne zapísať takto:

1. Vstup počtu hráčov N, ich "mien" a čísla vypadávajúceho v každom počítaní M.
2. Postavenie hráčov do kruhu.
3. Pokiaľ je ešte niekto v kruhu, opakuj:
   a) odpočítanie vypadávajúceho,
   b) jeho "vyhodenie" z kruhu, ďalej sa počíta od nasledujúceho.

Z hľadiska voľby typu prvku je teda dobré uvažovať o tom, ako "vyhodiť" hráča, na ktorého "to slovo padne". Čo keby sme hráčov očíslovali od 1 po N a na miesto toho, ktorý už vypadol, vložíme nulu? Zdá sa to dobré riešenie, takže to skúsime. Musíme si ešte zvoliť "mechanizmus vyčítania": Použijeme premennú *KDE* typu *index*, ktorá bude postupne nadobúdať hodnoty od 1 po M vtedy, ak narazí na nevyhodeného hráča. Bude sa pohybovať medzi 1 až N a po N prejde znovu na začiatok poľa.

Ešte je tu jeden problém: Kedy skončiť? Dalo by sa na to ísť dvomi spôsobmi: jednak zisťovať, či už pole neobsahuje samé nuly, jednak počítať si, koľko už hráčov vypadlo. Jednoduchší je určite druhý variant, pretože vieme, že ich bude práve N. Preto zmeníme cyklus *while* zo zápisu postupu na cyklus *for*.

Tento rozbor môže viesť k riešeniu v tvare programu:

```
Program VYCITANKA;
var   i, N, M, KDE,p: word;
```

```
      H             : array [1..100]of word;
Begin
 Write ('Zadaj pocet hracov a ktory ide z kola von);
 ReadLn (M, N);
 for i:=1 to N do H[i]:= i; H[N+1]:= 0;
KDE:= 0;
 for i:=1 to N do   {N-krát vyraďujeme}
  begin
   p:=0;      {počet "nenulových" hráčov, na ktoré sme narazili}
   repeat           {kým nenájdeme M nenulových hráčov}
    KDE:= KDE+1;
    if H[KDE]<>0 then p:= p+1;
    if KDE>N then KDE:= 0;
   until p=M;       {už ich máme}
   WriteLn (i, '. ty vypadava hrac cislo ', H[KDE])
   H[KDE]:= 0;      {vyradený hráč}
  end;              {koniec vyradení, kruh by mal byť prázdny}
 WriteLn ('KONIEC! ');
 ReadLn;
End.
```

V programe sme pri takomto prístupe museli použiť pomocnú premennú *p*. Pomáha nám pri počítaní nenulových prvkov poľa, na ktoré sme postupnou zmenou indexu *KDE* narazili. Pole obsahuje N+1 prvkov (posledná je nula), aby sme v extrémnych situáciách kontrolovali aj N-tý prvok. Kedy? Overte správnosť programu.

1. V programe je úmyselne urobená najmenej jedna logická a podstatná chyba. Logická chyba je taká, ktorú nenájde počítač pri preklade, ale môže podstatne ovplyvniť správnosť výsledkov. Takéto chyby sa najťažšie hľadajú, a preto: Nájdite ju![1]

*2. Potrebujeme vôbec pole? Nedal by sa program podstatne zjednodušiť tak, že budeme uvažovať iba nejaké násobky čísla M po nejakú hodnotu N*N? Porozmýšľajte o tom.*

3. Zostavte program, ktorý namiesto čísel bude pracovať s menami hráčov a vypíše poradie, v akom budú vypadávať spolu s ich pôvodným umiestnením v kruhu.

4. Upravte program tak, aby postupne znázorňoval, ako prebieha vyčítanie hráčov. Môžete k tomu použiť ďalšie grafické prostriedky, ako je farebné vyznačenie tých hráčov, ktorí ešte zostali v hre, tých ktorí už vypadli, miesto počítania a pod.

Úlohy tohoto typu sa efektívnejšie riešia pomocou tzv. dynamických štruktúr, pretože kruh sa stále zužuje, teda dynamicky mení počas realizácie programu. (pozri Turbo pascal II.)

[1] Ak sa v niektorých programoch uvedených v tejto knižke objavila chyba, je to pravdepodobne chyba tlače. Ak ste žiadnu chybu nenašli a veríte všetkému, je to vaša chyba! Snáď vás život naučí. "Dôveruj, ale preveruj!", hovorili naši predkovia.

7.8 Hlava alebo orol?

Je tradíciou našich končín, že sa pri vážnych rozhodnutiach zodpovednosť zvaľuje na peniaze, a pretože ich je málo - na mincu. (Určite to platí aspoň v politike.) Kedysi rozhodovala hlava alebo orol, potom panna alebo lev, dnes je to asi číslo a znak. (A čo hlava? Už nerozhoduje?) Využime generátor pseudonáhodných čísel, ktorý je súčasťou vybavenia TP na modelovanie hodov mincou.

Generátor pseudonáhodných čísel odovzdáva hodnotu náhodného čísla prostredníctvom volania funkcie ***Random*** používanej v dvoch formách:

Random – výsledkom je náhodné desatinné číslo medzi 0 (0 patrí) a 1 (1 nepatrí)

Random (n) – výsledkom je náhodné celé číslo od 0 po n-1 (teda n hodnôt).

Pretože generátor nie je úplne náhodný – vždy od začiatku vám dáva rovnakú postupnosť čísiel, je potrebné na začiatku programu s jeho použitím zadať príkaz ***Randomize*** – ten nastaví polohu generátora niekde do postupnosti v závislosti od strojového času.

Zostavte program, ktorý bude generovať N hodov mincou a vyhodnotí, koľkokrát padla hlava a koľkokrát orol. (Hlava a orol sú myslené obrazne.)

Postup je pomerne jednoduchý, preto môžeme hneď písať program:

```
Program HLAVA_A_OROL;
var i, N, hod, HLAVA, OROL: integer;

Begin
 Write ('Zadaj počet hodov mincou '); ReadLn (N);
 Randomize;                {nastavenie generatora}
 HLAVA:= 0; OROL:= 0;      {vynulovanie pocitadiel}
 for i:= 1 to N do
  begin
   hod:= Random(2);         {hod nadobudne nahodne hodnotu 0 alebo 1}
   if hod=0  then begin Write ('Hlava '); HLAVA:= HLAVA+1 end
             else begin Write ('Orol '); OROL:= OROL+1 end;
  end;
 WriteLn;
WriteLn ('Z ', N, ' hodov padla hlava ', HLAVA, '-krat, orol ', OROL,
         '-krat');
 ReadLn;
End.
```

Vo vnútri cyklu je zaradený aj výpis, ktorá z možností padla. Všimnite si, že generátor nie je až tak dobrý, ale pre účely, na ktoré sa používa TP, patrí k najlepším. (Veď aj mince sa niekedy správajú čudne, hlavne keď sa od nás kotúľajú.)

1. Skúste vynechať príkaz Randomize a niekoľkokrát po sebe spustiť program pre rovnakú hodnotu N.

2. Napíšte program, ktorý bude vypisovať N náhodných čísiel od 0 po 1 – všimnite si ich tvar.

7.9 Med alebo jed?

Sme na starobylom zámku v dávnych časoch; je pochmúrna noc a pri blikajúcej sviečke sedí za stolom bledý princ, nazvime ho Hamlet. Pred sebou na stole má dva poháre naplnené neznámou tekutinou, nazvime ju víno, a chce mať už život za sebou (alebo zaspať?). V jednom z tých pohárov je primiešaný jed, v druhom med. Hamlet si môže vybrať iba jeden z pohárov a vypiť ho. Ďalší osud je v jeho rukách, preto: "Med alebo jed?".

Skúste zostaviť program, ktorý pomôže Hamletovi vybrať si ten "správny" pohár.

Podobne ako v príklade hodu mincou máme dva možné prípady: Hamlet si vyberie med alebo jed. Zmeňme však postup pri voľbe obsahu pohárov. Dohovorme sa, že Hamlet si vyberá ľavý alebo pravý pohár. Pred jeho výberom sú už v pohároch namiešané med alebo jed. Tieto si môžeme uložiť ako reťazce znakov do „pohárov". Hamlet si vyberá uvedením prvého písmena poháru – *l* ("el") alebo *p*.

Tu je prvý variant riešenia:

```
Program HAMLET;
var   lavy, pravy    : string;
      miesanie       : integer;
      volba          : char;

Begin
 Randomize; miesanie:= Random (2);
 if miesanie=0 then begin lavy:= 'jed'; pravy:= 'med'; end
               else begin lavy:= 'med'; pravy:= 'jed'; end;
 Write ('Vyber si lavy - stlacenim L alebo pravy pohar -
        stlacenim P, potom stlac Enter ');
 ReadLn (volba);
        stlacenim P, potom stlac Enter ');
 ReadLn (volba);
 Write ('Prave si vypil ');
 if (volba = 'l') then WriteLn (lavy)
                  else WriteLn (pravy);
 WriteLn ('Na zdravie a sladke sny!);
 ReadLn;
End.
```

Program v tejto verzii je síce prvým nápadom, ale asi nie najšťastnejším. Predovšetkým mu môžeme vyčítať nie veľmi dobrú komunikáciu s užívateľom. Asi by bolo vhodné vyriešiť ho ako dvojitú alternatívu s náhodou, pridať cyklus, či chce užívateľ (Hamlet) ešte pokračovať, prípadne viazaný na dobrý výsledok predchádzajúceho behu programu.

Pridajte voliteľný počet pohárov, z ktorých bude časť (určená) otrávená, časť nie. Vyberať pomocou "rozpočítania" – náhodného čísla, ktoré môže byť väčšie ako počet pohárov.

7.10 Strihnime si!

Okrem hodu mincou sa niekedy (hlavne, keď mincu nemáme) používa známa hra Kameň-nožnice-papier. Pravidlá sú veľmi jednoduché. Každý z dvoch hráčov ukáže súčasne so súperom rukou jednu z troch možností:

- zovretú päsť (t. j. kameň),
- dva rozovreté prsty (t. j. nožnice),
- natiahnutú dlaň (t. j. papier).

Rovnako jednoducho sa hra vyhodnotí. Ak obaja súperi ukážu rovnaký predmet, potom je hra nerozhodná, ak nie, potom

- kameň otupí nožnice (päsť vyhrá nad dvomi rozovretými prstami)
- nožnice rozstrihnú papier (dva prsty víťazia nad dlaňou),
- papier obalí kameň (plochá dlaň poráža päsť).

Ako vidno z pravidiel, žiadny predmet nemá prevahu nad inými. Každý môže rovnako zvíťaziť aj prehrať, víťazstvo záleží od konkrétnej dvojice predmetov.

Táto hra sa dá jednoducho naprogramovať pre hru človeka s počítačom. Počítač pritom môže ďaleko dôslednejšie ako človek realizovať tzv. optimálnu stratégiu pre lepšie výsledky v hre. Optimálna stratégia je založená na tom, že možnosti sa majú voliť úplne náhodne a s rovnakou pravdepodobnosťou.

(*Poznámka*: Hra je veľmi stará a má pôvod v Číne. Pôvodne sa v nej nevyskytovali kameň, nožnice a papier, ale človek, kohút a červík. Pri vyhodnotení sa v tejto hre hovorilo: "Človek zje kohúta, kohút zďobne červíka a červík zožerie človeka.")

Zostavme program, pomocou ktorého bude hrať počítač hru Kameň-nožnice-papier proti človeku.

Hráč zadá svoju voľbu a počítač si svoju náhodne vygeneruje bez toho, aby "pozeral" na hráčovu voľbu. Najzložitejšie (nie myšlienkovo, ale programovo) je dobre zapísať všetky možnosti vzájomných kombinácií volieb, ktoré môžu nastať. Tu je jedno z možných riešení:

```
Program KAMEN_NOZNICE_PAPIER;
var   tah : char;
      th, tp : string[7];
      i, pvp, pvh, N, h1, h2 : integer;

Begin
 Write ('Zadaj pocet hier '); ReadLn (N);
 Randomize; pvh:= 0; pvp:= 0;
 for i:=1 to N do
  begin
   Write (i, '. hra: Zadaj K-ak volis kamen, N-ak volis
             noznice, P-ak volis papier '); ReadLn (tah);
   tah:= UpCase(tah);
   if tah='K' then begin th:= 'KAMEN'; h1:= 1 end;
   if tah='N' then begin th:= 'NOZNICE'; h1:=2 end;
   if tah='P' then begin th:= 'PAPIER'; h1:= 3; end;
   h2:= Random(3)+1;
   if h2=1 then tp:= 'KAMEN';
   if h2=2 then tp:= 'NOZNICE';
   if h2=3 then tp:= 'PAPIER';
   Write ('Tvoja volba: ', th, ' Moja volba: ', tp);
   if (h1+h2=2) or (h1+h2=6) then WriteLn ('NEROZHODNE! ': 20)                {K-K, P-P}
   if h1+h2=3 then
   if h1<h2 then begin WriteLn('VYHRAL SI! ':20); Inc(pvh) end                {K-N}
      else begin WriteLn ('VYHRAL SOM! ': 20); Inc(pvp) end;                  {N-K}
   if (h1+h2 =4) then
```

```
    if h1=h2 then begin WriteLn ('NEROZHODNE! ': 20)                          {N-N}
       else if (h1>h2) then begin WriteLn (' VYHRAL SI! ':20); Inc(pvh) end    {K-P}
            else begin WriteLn (' VYHRAL SOM! ': 20); Inc(pvp) end;           {P-K}
    if h1+h2=5 then
         if h1<h2 then begin WriteLn ('VYHRAL SI! ': 20) ; Inc(pvh) end       {N-P}
            else begin WriteLn ('VYHRAL SOM! ': 20); Inc(pvp) end;            {P-N}
  WriteLn (' Po ', i, '. hre: TY: ', pvh, ' JA: ', pvp);
 end;
WriteLn ('Z poctu hier ', N);
WriteLn ('si vyhral ', pvh, ', ja som vyhral ', pvp);
WriteLn ('nerozhodne bolo ', N-pvh-pvp, ' hier');
if pvh>pvp then WriteLn (' SI VITAZOM! ': 45)
   else if pvh=pvp then WriteLn (' NEROZHODNE!: 45)
        else WriteLn (' SOM VITAZ! ': 45);
ReadLn;
End.
```

Ťahy hráča a počítača sú uložené v premenných typu reťazec, aj keď hráč zadáva iba znak. Je to preto, aby hráč nemusel zadávať celé slová, pričom by sa mohol pomýliť v jednom znaku a už by boli problémy s testovaním voľby[1].

V programe sú použité prostriedky, ktorými nám TP umožňuje zefektívniť alebo sprehľadniť zápis programu:

1. *Funkcia UpCase(znak)* – premení prípadné malé písmeno na veľké. Je vhodné ju využiť v prípadoch, kedy by sme museli testovať všetky prípadné zadané písmená zvlášť, či sú malé alebo veľké.

2. *Procedúra Inc(premenná)* – zvyšuje hodnotu premennej ordinálneho typu "o jedna". V našom príklade zvyšuje počet výhier hráča (*pvh*) alebo počet výhier počítača (*pvp*) o 1, zodpovedá teda známemu *pvh:=pvh+1* alebo *pvp:=pvp+1*. (Opačný účinok – zníženie hodnoty "o jedna" – má procedúra *Dec*.)

Pre zjednodušenie porovnaní jednotlivých situácií, ktoré môžu nastať, sú zavedené pomocné premenné *h1* a *h2*.

Overte, či dávajú správne výsledky. Je zrejmé, že by sa text programu dal zjednodušiť, aby nemuselo byť toľko príkazov ***if*** *– skúste to. Napr. by ste mohli premenným* ***h1*** *a* ***h2*** *priradiť také hodnoty, aby ich súčet okamžite jednoznačne určil výsledok. Potom by sa komplikované vetvenie v závere cyklu dalo nahradiť príkazom* ***case****.*

Pokúste sa zostaviť program na hru Kameň-nožnice-papier s pamäťou". Pamäťou nerozumieme pamäť počítača, ale iba spôsob hry, pri ktorom si počítač pamätá ťahy súpera, aby ich využil pri svojom rozhodovaní. To je samozrejme možné aj pri skutočnej hre medzi dvomi ľuďmi. Ak si jeden z nich uvedomí, že jeho súper ukazuje jednu z troch možností častejšie ako ostatné dve, môže to pri hre využiť. Volí častejšie ten predmet, ktorý vyhráva nad súperovým.

Návod: Do programu by sa dalo vložiť to, že voľby hráča sa sčítavajú a využívajú pre voľbu ťahu počítača. To sa však dá až vtedy, keď hráč v predchádzajúcich hrách volil všetky z troch možných predmetov. Využiť túto znalosť o početnosti voľby jednotlivých predmetov je možné na počítači jednoduchšie a presnejšie. V každom ťahu môžeme zaznačiť voľbu hráča a pripočítať ju k stavu zo začiatku hry. Tým sa dokonca môžeme viac dozvedieť o početnosti svojich volieb, ako by sme predpokladali.

(Pozor! Program by sa dal zostaviť aj tak, že by sa rozhodnutie počítača priamo riadilo predchádzajúcou voľbou hráča. To by síce bolo možné, ale nefér! Hra by úplne stratila zmysel.

[1] Pri vkladaní reťazcov znakov z klávesnice sa stávajú veľmi nepríjemné, takmer infarktové situácie. Napr. ak niekto zadá do reťazcovej premennej hodnotu 'Kamen ' (za "n" pridá medzeru) a program testuje na 'Kamen'.

Tak ako keby v skutočnosti mal jeden hráč možnosť voliť ťah až potom, ak jeho súper ukázal svoj predmet.)

Uveďme si pravidlá hry Morra: Je to zložitejšia podoba hry Kameň-nožnice-papier. Hru hrajú dvaja hráči. Obaja súčasne ukážu 1 alebo 2 prsty. V rovnakom okamihu súčasne hádajú, koľko prstov ukáže súper. Ak majú obaja pravdu alebo keď sa obaja pomýlia, je hra nerozhodná. Ak má pravdu iba jeden hráč, získa toľko bodov, koľko je súčet prstov, ktoré obaja hráči ukazujú.

Zostavte program hry Morra pre človeka a počítač.

Ako u všetkých hier, kde hrá úlohu náhoda a odhad, neexistuje metóda zaručujúca víťazstvo. Zaujímavé je, že túto hru analyzoval a tzv. zmiešanú stratégiu navrhol aj matematik John von Newmann, zakladateľ matematickej teórie hier, známejší pre nás ako autor základnej logickej schémy počítača, tzv. von Newmannovej schémy.

7.11 Myslím si číslo

Ak už je najhoršie a aj papier sa minul, môžu ľudia v týchto extrémnych podmienkach hrať aspoň hru "Myslím si číslo". Jeden z hráčov si myslí číslo z vopred dohovoreného intervalu, druhý ho háda pomocou otázok, na ktoré sa odpovedá iba "áno" alebo "nie".

Máme simulovať hádanie čísla z intervalu 0–N. Číslo si náhodne generuje počítač a hráč má na čo najmenší počet pokusov uhádnuť myslené číslo. Počítač "napovedá" pomocou možností, napr. ***"hladane cislo je vacsie", "hladane cislo je mensie"*** *a pod.*

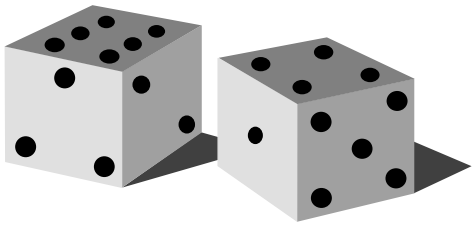
Úloha nie je zložitá. Stačí iba uvážiť vhodnú organizáciu cyklu hádaní hráča. Na test, či hráč uhádol zavedieme logickú premennú *uhadol*, ktorej hodnota umožní ukončiť opakovanie v cykle repeat-until.

```
Program HADANIE_CISLA;
var   i, N, cislo   : integer;
      uhadol        : boolean;

Begin
 Write('Hra Myslim si cislo. Zadaj maximalnu moznu hodnotu ');
 ReadLn(N);
 Randomize; cislo:= Random (N+1);
 uhadol:= false; i:= 0;
 repeat
  i:= i+1;
  Write ('Zadaj cislo '); ReadLn (pokus);
  if pokus = cislo then uhadol:= true
    else if pokus>cislo then WriteLn (pokus, 'je vacsie ako myslene cislo')
           else WriteLn (pokus, 'je mensie ako myslene cislo');
 until uhadol;
 WriteLn ('Hladane cislo ', cislo, 'si uhadol na ', i, ' pokusov! ');
 ReadLn;
End.
```

1. Zostavte program, ktorý umožní hráčovi uhádnuť číslo aj v prípade, že počítač môže – nemusí jedenkrát v odpovedi "zaklamať".

2. Vymeniť úlohy: hráč si myslí číslo a počítač ho čo najoptimálnejšie hľadá – použiť metódu polenia intervalu. Ak je $N = 2^{K-1}$ *tak stačí K otázok.*

7.12 Kocky sú hodené![1]



Možno poznáte, možno nie, hru (Česi majú krásny výraz "vrhcáby", aj keď je z nemčiny), ktorú sa hrávali zbojníci, vojaci a všetci, ktorí potrebovali zabiť "niekoho" (najčastejšie čas). Nosili so sebou kocky, na stenách ktorých bolo 1–6 bodiek; zamiešali ich v pohári, ktorý tiež nikdy nechýbal, a vyklopili na stôl alebo inú rovnejšiu plochu. Kto mal vyšší súčet, vyhral. Mohlo sa hrať viac kôl, mohlo sa hrať postupne o niečo v každom kole alebo uvažovať až celkový súčet zo všetkých hier.

Dnes už sú tieto kocky vzácne. Niektoré vzorné rodiny ich používajú na hranie hry utužujúcej súdržnosť "Človeče, nehnevaj sa!"; väčšinou je však dnes najobľúbenejšou rodinnou hrou náhodné prepínanie televíznych staníc. Využime silu monitora a vráťme sa do zbojníckych čias (aj keď je to už trocha iná romantika).

Zostavte program, ktorý bude náhodne generovať hod M kockami pre N hráčov.

Mohli by sme úlohu riešiť naraz čo najvšeobecnejšie, ale radšej sa zamerajme na toto zjednodušenie:

```
{VST: N hráčov, M kociek}
?
{VÝS: výpis súčtov hodu M kockami pre každého hráča}
```

Budeme potrebovať generátor náhodných čísiel, ak ide len o výpis (nie o zapamätanie). Na prvý pohľad nám stačia dva v sebe vložené cykly:

Pre každého hráča od 1 po N:
"konto":= 0;
Pre každú kocku od 1 po M:
Náhodný hod od 1 po 6;
Jeho pripočítanie na "konto" hráča;
Výpis "konta" hráča

Program (bez "príkras") môže mať tvar:

```
Program KOCKY_1;
var i, j, N, M, KONTO, hod: integer;

Begin
 ReadLn(N, M); Randomize;
 for i:=1 to N do
  begin
   KONTO:= 0;
   for j:= 1 to M do
    begin
     hod:= Random(6)+1; KONTO:= KONTO + hod;
    end;
   WriteLn (i, '. hrac ma sucet = ', KONTO);
  end;
 ReadLn;
End.
```

[1] "Alea iacta est!" – údajný výrok Júlia Caesara, rímskeho vojvodcu, politika a filozofa pri prekročení rieky Rubikon, oddeľujúcej starovekú Itáliu od Gálie r. 49 pred n. l.; tým sa začala občianska vojna v Ríme; prenesene značí: "Závažné rozhodnutie bolo vykonané".

Je rozdiel medzi tým, ako program počíta navonok a ako vo vnútri. Hodnoty jednotlivých kociek totiž vôbec nebolo vidno, len celková suma *KONTO* po pripočítaní M "kociek". Pokojne by pri tomto zadaní stačilo použiť jeden cyklus, v ktorom by sa hneď generovala hodnota *KONTO*. Je potrebné len pomocou M vhodne vyjadriť minimálnu a maximálnu hodnotu, ktorú môže nadobudnúť.

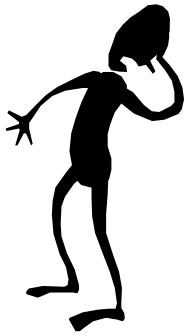
1. Riešte program KOCKY_2 využitím jedného cyklu.

2. Riešte program KOCKY_3 tak, aby pre každého hráča vypisoval všetky hodnoty kociek vrátane celkového súčtu.

3. Zmeníme zadanie: Zostavte program KOCKY_4 tak, aby si zapamätal kontá jednotlivých hráčov a potom vyhodnotil, ktorý z nich vyhral. Pozor na možnosť, že najvyšší súčet môžu mať viacerí. Je možné pre nich vytvoriť "rozoskakovanie", v ktorom o výhru budú hrať už len oni.

7.13 Ruská ruleta

Chlieb a hry sú tým, čo ľudia potrebujú neustále. Kedysi sa za mrazivých nocí v Rusku nedalo (ani nedá) vonku vydržať, a tak sa ruskí dôstojníci museli zabávať vo vykúrených miestnostiach, kde kolovali karty, vodka a pištole, či šable. Nevedno, koho to napadlo, ale určite nie náhodou sa jedna zo "zábav" volá "ruská ruleta". Neodporúčam ju reálne skúšať, predstavme si ju iba ako námet na program:



Pri stole sedí N hráčov, na stole je položená šesťranová pištoľ – kolt s otáčavým zásobníkom. V zásobníku je iba jeden náboj, ostatné miesta sú prázdne. Každý hráč roztočí zásobník, priloží si hlaveň pištole k spánkovej kosti a stlačí spúšť. Pokračovanie je zrejmé, osud neistý.

Zostavme program, ktorý bude simulovať bezpečnú ruskú ruletu.

Možný prístup k riešeniu je takýto: Miesta v zásobníku pištole si predstavíme ako čísla od 1 po 6. Potom do náhodne zvoleného miesta vložíme náboj. Hazardný (až bizardný) hráč si určí, koľkokrát sa má zásobník otočiť. Toto otočenie (ich počet označíme N) spôsobí vygenerovanie ďalšieho náhodného čísla – ako keby miesta v zásobníku, ktoré vyúsťuje do hlavne. Po N otočeniach (generovaniach náhodného čísla) sa vyhodnotí, či sa číslo miesta náboja zhoduje s číslom ústiacim do hlavne. Ak áno, tak „bum", inak „cvak". Algoritmus je hotový, stačí len zohnať počítač a nabiť nasledujúci program:

```
Program RUSKA_RULETA;
var   N, i, naboj, hlaven  : word;

Begin
 WriteLn ('RUSKA RULETA': 55);
 Write ('Zadaj pocet otoceni zasobnika'); ReadLn (N);
 Randomize;
 naboj:= Random (6) + 1;
 for i:= 1 to N do hlaven:= Random (6) + 1;
 if naboj = hlaven  then WriteLn ('BUUUMMM!!! ')
                    else WriteLn ('CVAK! ');
 ReadLn;
End.
```

Príslušnú estetickú a hororickú komunikáciu programu s jeho užívateľom si doplňte podľa svojich predstáv.

1. Čo by sa stalo, ak by užívateľ programu zadal počet otočení N rovné nule? V tomto prípade by v praxi bol samovrahom. V programe to však nemusí platiť! Ak sa cyklus for nevykoná ani jedenkrát, v premennej hlaven je nedefinovateľná hodnota. Upravte program tak, aby s týmto variantom program uvažoval.

2. Poznáme aj iné spôsoby, ako si ohroziť život a nazývajú sa tiež ruleta. „Normálna" ruleta sa hráva v kasínach a dajú sa v nej vyhrať, ale väčšinou prehrať veľké peniaze. Stávky sú trochu zložitejšie, môžete staviť ľubovoľný vklad (kým ho ešte máte) na rôzne prípady, napr. na červené alebo čierne... Podľa pravdepodobnosti prípadu môžete vyhrať istý násobok vkladu. Zistite pravidlá „normálnej" rulety a skúste ju modelovať na počítači.

7.14 Miešanie kariet

Ku kartám patrí aj miešanie, ktoré niekedy trvá dlhšie ako samotná hra. Na počítači nebudeme hrať na psychiku hráčov, využijeme jeho schopnosť rýchlo vymieňať hodnoty podľa určeného algoritmu.

Náhodné pole

Pre dané N namieša počítač do jedno- (dvoj-) rozmerného poľa náhodné čísla zadaného rozsahu, pričom prvky môžu byť aj navzájom rovnaké. Ukážme si riešenie na príklade generovania N náhodných čísiel z intervalu <OD, PO>, teda aj OD aj PO do intervalu patria.

```
Program NAHODNE_POLE;
const max = 100;
var   i, N, OD, PO  : integer;
      A             : array [1..max] of integer;

Begin
 ReadLn (N, OD, PO); Randomize;
 for i:= 1 to N do
  begin
   A[i]:= Random (PO-OD+1)+OD;
   Write (A[i], ‘, ‘);
  end;
 ReadLn;
End.
```

Myslím, že program nepotrebuje komentár.

Majme dané 1-rozmerné pole N hodnôt. Počítač by ich mal premiešať tak, ako by to boli karty.

Predpokladajme, že pole je už zadané a hodnoty jeho prvkov sú uložené v $A[1]..A[N]$. Pritom nie je dôležité, aké hodnoty pole A obsahuje – nazvime ich karty (môžu to byť čísla, reťazce znakov alebo ľubovoľné ďalšie údajové typy).

Možno použiť rôzne spôsoby – algoritmy miešania hodnôt poľa, podobne ako pri miešaní kariet:

a) uskutočniť istý počet výmen *i*-tej a *j*-tej karty v poli,
b) presúvať istú podpostupnosť kariet na koniec poľa,
c) náhodne rozdeliť pole na dve podpostupnosti a “poprekladať” karty medzi nimi.

Porozmýšľajte o iných miešaniach.

Vyberme si možnosť a), teda zamiešanie výmenou dvoch náhodne zvolených kariet istý počet krát. Predpokladajme, že je pole A už obsadené nejakými hodnotami typu *KARTA*, ktorý nebudeme bližšie špecifikovať.

Program by mohol mať tvar:

```
Program MIESANIE_01;
const MAX = 100;
var   i, j, krat, k : integer;
      A      : array [1..MAX] of KARTA;
      pomoc  : KARTA;

Begin
```

```
 {vstup poľa A s počtom prvkov N , resp. nastavenie jeho hodnôt }
 Randomize;
 krat:= Random(50) + 50;          {zamiešať 50..99 - krát}
for k:=1 to krat do
  begin
   i:= Random(N) + 1;
   j:= Random(N) + 1;
   pomoc:= A[i]; A[i]:= A[j]; A[j]:= pomoc;
  end;
 {prípadný výpis zamiešaných hodnôt}
End.
```

Doplňte program tak, aby bol funkčný. (Návod: Aby sme mohli kontrolovať počet a uskutočnenie výmen, môžete do poľa A uložiť hodnoty 1..N a dať vypísať nielen počet generovaných výmen, ale aj pôvodné a výsledné pole.)

K zamiešaniu kariet patrí aj ich rozdanie. Preto už dokončime hazard a zamerajme sa na tento problém.

Rozdanie kariet

Využitím príkladu o zamiešaní kariet môžeme teraz rozdať zo zamiešanej postupnosti každému z *M* hráčov *K* kariet a vypísať ich rozdelenie (*MxK<=N*).

Rozdávať karty je možné viacerými spôsobmi. Vyberme si ten, kedy dávame každému z *PH* hráčovi do kruhu po jednej karte dovtedy, kým ich nebudú mať všetci stanovený počet kariet *PK*.

```
Program ROZDAVANIE;
const MAX = 108;
var   PH, PK, i, j, ktora  : integer;
      A             : array [1..MAX] of KARTA;
      K             : array [1..10,1..10] of KARTA;    {maximálne 10 hráčov po 10 kariet}

Begin
 Write (' Rozdavam karty! Zadaj pocet hracov a pocet kariet
         pre hraca '); ReadLn (PH, PK);
 {Zamiešanie kariet v poli A - doplniť}
 ktora:= 0;
 for i:=1 to PK do               {i-ta karta}
  for j:=1 to PH do              {j-temu hráčovi}
   begin
    ktora:= ktora + 1;           {počítadlo kariet z kopy}
    K[j,i]:= A[ktora];
   end;
 {Výpis rozdaných kariet - podľa potreby doplniť}
 ReadLn;
End.
```

Pole *A* sa skladá z prvkov typu *KARTA*, ktorý je potrebné zadefinovať. Následne je potrebné doň vložiť "karty" a toto pole zamiešať.

1. Doplňte program o zamiešanie a výpis kariet jednotlivých hráčov.

2. Porozmýšľajte o iných spôsoboch rozdávania – napr. každému hráčovi naraz všetky karty, alebo po nejakých skupinách kariet.

3. Rozdajte 32 kariet medzi troch hráčov tak, ako to platí pri mariáši. Tam prvý dostane sedem kariet, potom už stále všetci po päť, kým sa nerozdajú všetky karty. Prvý hráč nakoniec odhadzuje dve karty do "talónu", aby sa počet kariet pri hre vyrovnal.

7.15 Hazard zadarmo – KENO 10

Ľudí (presnejšie zložku, ktorá je hravejšia – či menej zaneprázdnená – mužov) odjakživa lákal hazard. Možnosť vyhrať alebo prehrať za chvíľu toľko, koľko sa poctivou prácou za celý život nedá zarobiť, premení aj najváženejšieho a najpokojnejšieho človeka na šialenca s trasúcimi sa rukami zaťato naháňajúcimi mušku šťastia.

Hazard na počítači má svoje výhody aj nevýhody. Môžete si pokojne staviť alebo zahrať ľubovoľnú hru bez finančnej ujmy, ale na druhej strane sa výhry okrem prípadného dobrého pocitu nedočkáte. Nevadí. Skúsme vytvoriť program, na ktorom si môžete "na sucho" overiť svoju mieru šťastia.

Poznáte stávkovú hru Keno 10? Každý hrací deň z 80 čísiel od 1 po 80 ťahajú 20 čísiel. Tipujúci dáva na tiket 10 čísel.

(Možností voľby počtu čísel aj počtov žrebovaní je viac; tu zadanie zjednodušujeme, ale nič vám nebráni zobrať si na pošte tiket – je zadarmo, preštudovať si pravidlá a vytvoriť "kompletné" Keno 10 "satarmo".)

Pri zvolenom vklade platia tieto pravidlá výhry:

- ak uhádnete 10 čísel - vyhrávate 200 000 – násobok vkladu,
- ak uhádnete 9 čísel - vyhrávate 10 000 – násobok vkladu,
- ak uhádnete 8 čísel - vyhrávate 500 – násobok vkladu,
- ak uhádnete 7 čísel - vyhrávate 20 – násobok vkladu,
- ak uhádnete 6 čísel - vyhrávate 10 – násobok vkladu,
- ak uhádnete 5 čísel - vyhrávate 3 – násobok vkladu,
- ak uhádnete 4–1 čísel - nevyhrávate nič (resp. 0 – násobok vkladu),
- ak neuhádnete nič (alebo 0 čísel) - vyhrávate 1 – násobok vkladu.

Prevádzkovateľ to dobre vymyslel. Pri prvom čítaní tejto ponuky sa pravdepodobnosť výhry zdá dosť vysoká, vkladať sa dajú aj nízke sumy – od 5,- Sk vyššie. O skutočnosti a vašich šanciach vás však môže presvedčiť aj to, že si viackrát spustíte program na simulovanie ťahu tejto hry. Poďme si ho načrtnúť:

```
Vklad a zvolené čísla - 10, nastavenie počiatočných hodnôt
Generovanie ťahu 20 čísiel
Vyhodnotenie (ne-)výhry
```

Dohodnime sa, že budeme používať tieto premenné:

VKLAD – hodnota vkladu hráča,
C[1]..C[10] – čísla zvolené hráčom,
V[1]..V[20] – vylosované čísla,
N[1]..N[10] – násobky vkladu pre výhru,
POCET – počet čísel, ktoré hráč uhádol,
VYHRA – celková hodnota výhry hráča,

Vklad a voľba čísel

Znovu pre zjednodušenie nepoložíme na vklad inú podmienku, len aby to bolo kladné celé číslo – typ *longint*. Trochu môžu robiť problémy aj vysoké hodnoty, ktoré pri výhre môžu prekročiť

rozsah zvoleného typu. Je už "osobnou vecou", nakoľko sa bude programátor venovať ošetrovaniu extrémnych prípadov vstupov[1].

Časť programu riešiaca vstup a nastavenie konštantných hodnôt môže bez zabezpečenia komunikácie vyzerať napríklad takto:

```
begin
 ReadLn(VKLAD);
 for i:= 1 to 10 do ReadLn (C[i]);
 N[10]:= 200000; N[9]:= 10000; N[8]:= 500; N[7]:= 20; N[6]:= 10;
 N[5]:= 3; N[4]:= 0; N[3]:= 0; N[2]:= 0; N[1]:= 1;
 for i:= 1 to 20 do V[i]:= 0;
end;
```

Generovanie ťahu 20 čísiel

Máme náhodne generovať 20 čísiel medzi 1 a 80 tak, aby sa neopakovali, a tieto ukladať do poľa V. Práve problém opakovania bude asi najobťažnejší v tejto časti programu (a aj v celom programe). Možností prístupu môže byť znovu viac (bez nároku na vyčerpanie všetkých):

a) Dopĺňať generované číslo do *V*, iba ak sa nezhoduje s niektorým z predtým vylosovaných; všetky činnosti vykonávať iba s poľom *V*.

b) Dopĺňať generované číslo do *V* na základe použitia pomocného poľa o 80 hodnotách, ktoré bude obsahovať informácie o tom, či už bolo príslušné číslo vytiahnuté.

c) "Vyťahovať" čísla z poľa o 80 hodnotách (1–80). Vytiahnuté číslo vymeniť s posledným a potom ťahať zo 79 čísiel atď. 20-krát.

Najčastejšie sa používa asi prípad a), my si ukážme druhú možnosť riešenia b). Pomocné pole P bude obsahovať logické hodnoty *true*, *false* podľa toho, či bolo číslo zodpovedajúce indexu prvku poľa už vytiahnuté alebo nie. Najskôr musíme nastaviť hodnoty na *false*:

```
for i:=1 to 80 do P[i]:= false;
```

Potom už môžeme začať generovať 20 hodnôt náhodných čísel z intervalu od 1 po 80:

```
begin
 Randomize;
 for i:= 1 to 20 do begin
   repeat
    cislo:= Random(80)+1;
   until P[cislo]= false;
   P[cislo]:= true;
   V[i]:= cislo;
 end;
end;
```

V cykle *for*, ktorý sa dvadsaťkrát zopakuje, sa generuje náhodné číslo od 1 po 80. Ak na jeho mieste v pomocnom logickom poli je *false*, táto hodnota nebola vytiahnutá a možno ju vložiť do poľa vylosovaných čísel *V*. Vtedy je potrebné aj hodnotu v poli *P* zmeniť na *true*. V opačnom prípade (ak už bolo číslo *cislo* vytiahnuté, sa bude cyklus repeat - until opakovať. Podmienku za *until* je možné zmeniť aj na elegantnejšiu: *not(P[cislo])*.

Pokúste sa napísať časti programu, ktoré budú generovanie čísel riešiť pomocou návodov na riešenie uvedených pod a) a c).

[1] Zabezpečenie programu proti úmyselným aj neúmyselným chybám užívateľa tvorí pri "klasickom" programovaní často viac ako polovicu zdrojového textu programu. Je to nevďačná a nikdy nie dokonale zvládnutá úloha, pretože fantázia ľudí a ich schopnosť pokaziť nepokaziteľné sú nevyčerpateľné. Programom, ktoré sa snažia ošetriť všetky možné kolízie, hovoríme fool-proof (čítaj fulprúf, približne preložené "blbotesné"). Od mnohých podobných nepríjemností nás môže odbremeniť iba tzv. udalosťami riadené programovanie, predstaviteľom ktorého je napr. Visual Basic alebo Delphi.

Vyhodnotenie (ne-)výhry

Tento podproblém už zdanlivo nie je ani problémom. Stačí si uvedomiť, že musíme iba zistiť, koľko z čísiel *C*[*1*]..*C*[*20*] sa nachádza medzi číslami *V*[*1*]..*V*[*20*]. Dostaneme hodnotu *POCET* a *VYHRA* sa zistí jednoducho využitím príslušného násobku *N*[*POCET*]. Možné riešenie:

```
begin
 POCET:= 0; VYHRA:= 0;
 for i:= 1 to 10 do
  begin
   pom:= false;
   for j:= 1 to 20 do if C[i]=V[j]then pom:= true;
   if pom then POCET:= POCET + 1;
  end;
 VYHRA:= VKLAD * N[POCET];
 WriteLn (VKLAD, POCET, VYHRA);
end;
```

Pomocná logická premenná *pom* nám hodnotou *true* signalizuje, či sa zvolené číslo *C*[*i*] nachádza medzi vylosovanými číslami *V*[*1*]..*V*[*20*]. Často sa používa aj signalizácia pomocou číselnej premennej, ktorá nadobúda hodnoty 0 alebo 1, ale tento postup je zrozumiteľnejší a prehľadnejší. Hodnota *C*[*i*] sa totiž nemôže v poli *V* objaviť viackrát – všetky hodnoty v ňom by mali byť navzájom rôzne, veď tak sme to chceli. (Či sa to podarilo, je z iného súdka.)

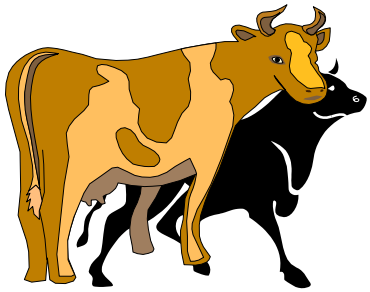
Je samozrejmé, že výstup musí byť realizovaný prehľadnejšie:

– mali by ste doplniť výpisy tipovaných aj vylosovaných čísiel, aj s prípadným napodobnením postupného ťahania čísiel,
– výpis čísel, ktoré sa zhodujú,
– kultúrny výpis výhry s prípadnou pochvalou alebo útechou.

1. Čo sa stane, ak si používateľ zadá 10 rovnakých hodnôt tipovaných čísiel (čo nie je ošetrené) a náhodou bude toto číslo vytiahnuté? Ako sa tomu dá predísť? Opäť je možností viac. Pokúste sa tento nedostatok programu napraviť.

2. Upravte program tak, aby presne simuloval hru KENO 10. Tipujúci si môže voliť aj počet čísel, ktoré tipuje, potom sa ale menia násobky na určenie výhry. Dá sa voliť aj počet ťahov a ďalšie parametre. Stačí si iba zohnať tiket a pustiť sa do programu.

7.16 Mastermind



V 70-tych rokoch tohto storočia bola veľmi obľúbená hra nazývaná Mastermind (“mastermajnd”, doslovne preložené ako “majstrovský rozum”):

Hrajú dvaja hráči. Každý z nich si zvolí štvormiestne číslo, ktoré má navzájom rôzne číslice. Hráči musia uhádnuť súperovo číslo na čo najmenší počet ťahov. Hráči striedavo hádajú číslo súpera, ten oznamuje, koľko číslic z nich bolo uhádnutých takto:

1. ak uhádol nejaké číslice na správnom mieste – oznamuje ich počet, napr. 1B, t.j. bola uhádnuté 1 číslica na správnom mieste.
2. ak uhádol nejaké číslice, ale nie na správnom mieste – oznamuje ich počet, napr. 2K, t. j. boli uhádnuté 2 číslice, ktoré ale nie sú na správnom mieste.

(*Poznámka*: Označenie B znamená “býk”, označenie K “krava” podľa jedného z variantov tejto hry.)

Hra končí tým, že jeden z hráčov uhádne všetky číslice čísla súpera na správnych miestach, ten oznámi 4B.

Je to úloha ako stvorená pre hru s počítačom. (A v skutočnosti ozaj podnietila mnohých odborníkov na programovanie hľadať optimálnu stratégiu tak, aby ju mohol hrať ako rovnocenný hráč aj počítač.) My ju radšej zjednodušíme a počítač sa stane iba prostriedkom, ktorý zvolí číslo a bude vyhodnocovať ťahy človeka.

Zostavme zjednodušenú verziu hry Mastermind na počítači.

Jednotlivé činnosti by mohli vyzerať takto:

1. Generovanie štvormiestneho čísla s navzájom rôznymi číslicami.
2. Kým hráč neuhádne všetky číslice na správnych miestach, prijímať a vyhodnocovať jeho ťahy.
3. Oznámiť skutočné číslo a počet pokusov.

Zostáva len “premeniť na drobné” jednotlivé činnosti:

1. Štvormiestne číslo by sme si mohli zapamätať v tvare postupnosti – jednorozmerného poľa 4 číslic. To by nám pomohlo zjednodušiť vyhodnotenie pozície využitím indexu. Nesmieme zabudnúť pri generovaní na to, že všetky štyri číslice musia byť navzájom rôzne.

Ako generovať číslice tak, aby boli rôzne? Dávať na ľubovoľné miesto aj nulu? Dohodnime sa, že nula medzi číslicami nebude na prvom ani na ďalších miestach. Postup generovania by mohol byť takýto:

a) vygenerujeme náhodne číslicu od 1 po 9,
b) pre *i* od 2 po 4 generujeme *i*-tu číslicu dovtedy, kým nie je rôzna od predchádzajúcich.

2. Situáciu pomôže riešiť cyklus:

```
repeat
 ŤAH_HRÁČA;
 VYHODNOTENIE;
until uhadol;
```

Pre *Ťah_hráča* je problém jedine v spôsobe vkladania. Buď ho zadá celé a program ho “rozseká” alebo bude vkladať číslicu po číslici. Z úcty k užívateľovi by bol vhodnejší prvý prístup.

Vyhodnotenie je zložitejšie, pravdepodobne musí byť “dvojpriechodové”. Najskôr zistíme počet uhádnutých číslic na správnych miestach, potom počet uhádnutých číslic na nesprávnych

miestach. Musíme iba dať pozor na to, aby sme niečo nepočítali dvakrát. Ak hráč uhádol všetky štyri číslice na správnych miestach, nastavíme logickú premennú *uhadol* na hodnotu *true*.

3. Jednoduchá úloha – vypíšeme "myslené" číslo, a ak máme v cykle hádania počítadlo ťahov, aj jeho hodnotu.

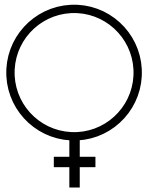
```
Program MASTERMIND;
var   i, pocet, b, k       : integer;
      a, c          : array [1..4] of integer;
      uhadol, uzje  : boolean;

Begin
 Randomize; a[1]:= Random(9)+1; {Generovanie čísla}
 for i:=2 to 4 do
  begin
   uzje:= false;
   repeat
    a[i]:= Random(9)+1;
    for j:=1 to i-1 do if a[i]=a[j]then uzje:= true;
   until not(uzje);
  end;
 pocet:=0; uhadol:=false;
 repeat {Ťah hráča a zistenie správnosti}
  b:=0; k:=0;
  Write ('Zadaj hadane stvorciferne cislo '); ReadLn (cislo);
  pocet:=pocet+1;
  for i:=1 to 4 do {Rozbitie zadaného čísla na číslice do poľa c}
   begin
    c[4-i+1]:= cislo mod 10; cislo:= cislo div 10;
   end;
  for i:=1 to 4 do if a[i]=c[i] then b:=b+1;
                   {Zistenie počtu "býkov"= b a "kráv"= k}
  for i:=1 to 4 do
   begin
    for j:=1 to 4 do if a[i]=c[j] then k:=k+1;
   end;
  k:=k-b;
  WriteLn( pocet, '. tah: Uhadol si ', b, 'cislic na spravnom
         mieste, ', k, 'cislic na nespravnom mieste');
  if b=4 then uhadol:=true;
 until uhadol;
 Write ('UHADOL SI! Hladane cislo je '); {Vyhodnotenie}
 for i:=1 to 4 do Write (a[i]); WriteLn;
Write(' - Pocet tahov = ', pocet);
 if pocet <11 then WriteLn (' a to je velmi dobré! ')
    else if pocet<21 then WriteLn (' a to je slusne! ')
             else WriteLn (', ale si sa nadrel! ');
 ReadLn;
End.
```

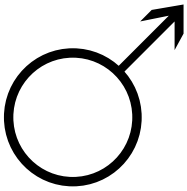
Ak užívateľ programu zadá dlhšie ako štvormiestne číslo, program z neho zoberie do úvahy iba posledné štyri číslice. Isté problémy môžu nastať (aj keď to užívateľ nemusí vedieť), ak niekto zadá štvormiestne číslo, ktoré obsahuje rovnaké číslice. Okrem problémov s nesprávnym počtom číslic uhádnutých na nesprávnom mieste, to však iné asi nevyvolá. (Overte to!) V závere programu je doplnené slovné vyhodnotenie, ktoré môžete podľa svojej úvahy a skúseností patrične upraviť.

1. Overte, či je správne zisťovanie počtu číslic na správnom a nesprávnom mieste. Ako je to so vzťahom premenných b, k?

2. Po niekoľkonásobnom hraní tejto hry je možné uvažovať o stratégii hľadania. Skúste nejakú stratégiu formulovať tak, aby ju mohol realizovať počítač. Potom by ste mohli napísať program hry Mastermind, ktorý bude skutočnou hrou človeka proti počítaču.

7.17 Patríme k sebe, miláčik?





Otázka, ktorú si kladú dvojice (resp. manželské trojuholníky a iné n-uholníky) na celom svete už od nepamäti. Ak by existoval nejaký algoritmus, už by ho určite dávno niekto objavil, pretože táto oblasť ľudskej sebarealizácie patrí k najobľúbenejším, najpreskúmanejším, ale zároveň najhorúcejším a najmenej predvídateľným. Samozrejme, že si zase úlohu zjednodušíme a pokúsime sa svojimi vedomosťami z programovania trošku prispieť k tomu, aby sa ľudia zbadali, kým nebude neskoro.

Zostavte program, ktorý bude perspektívnych partnerov testovať pomocou istého počtu otázok a vyhodnotí ich "súzvuk" v odpovediach na ne.

Predpokladajme, že máme k dispozícii sadu otázok, na ktoré každý z partnerov odpovie *"áno"* alebo *"nie"*. Ďalej predpokladajme, že odpovedajú nezávisle od seba a počet odpovedí, v ktorých sa zhodnú, môže byť istým kritériom pre ich perspektívny "súzvuk duší a tiel".

Pretože odpovede môžu byť iba *"áno"* a *"nie"*, na ich zapamätanie by bolo vhodné použiť pole logických hodnôt. V pascale poznáme štandardný typ ***boolean – logický typ***, ktorý spĺňa tieto podmienky:

1. Obsahuje iba dve hodnoty *false* (nepravda) a *true* (pravda).
2. Tieto hodnoty sú usporiadané *false*<*true*. (V živote je to naopak.)
3. Výhodou logického typu je, že v pamäti zaberá iba jeden bajt a zápis podmienok v podmienených príkazoch a podmienkach cyklov sa jeho prostredníctvom sprehľadňuje. Logickej premennej totiž môžeme priradiť aj hodnotu nejakého logického výrazu (t.j. takého, ktorý po vyhodnotení nadobúda hodnotu *true* alebo *false*).
4. Nevýhodou logického typu je nemožnosť vstupu jeho hodnoty priamo z klávesnice.

Pretože máme dvoch partnerov a potrebujeme si zapamätať ich odpovede na istý počet otázok *PO*, zvolíme si k tomu pole logických hodnôt, ktoré obsahuje dve "PO-tice" odpovedí – pole *ODP* (pozri deklarácie programu *PARTNERI*).

Ďalšou "fintou", ktorú si v tomto programe ukážeme, je používanie typu údajov ***set – množina***.

Typ *množina* je zložený typ, ktorý sa skladá z ordinálnych typov. Množina nesmie mať viacej ako 255 možných hodnôt a ordinálne hodnoty hornej a dolnej hranice základného typu musí byť z intervalu 0..255.

Najpoužívanejšou funkciou pri práci s množinami je zistenie, či sa nejaký prvok v danej množine nachádza. Je to logická funkcia *in*. Má tvar:

```
prvok in množina
```

a jej výsledkom je *true* v prípade, že prvok je prvkom množiny, inak *false*.

Na čo vlastne použijeme množinu? Obaja partneri majú odpovedať na otázky odpoveďou *"áno"* alebo *"nie"*, pričom už bolo povedané, že logická hodnota nemôže byť zadávaná zo vstupu. Pokojne by stačilo, keby sme testovali stlačenie nejakého znaku z klávesnice a určovali, či bolo stlačené "a" (alebo "A"), resp. "n" (alebo "N"). Ostatné stlačené klávesy nebudeme brať do úvahy. To znamená, že si vytvoríme množinu "prípustných" stlačených klávesov obsahujúcu znaky 'a', 'A', 'n', 'N'. Využitím operácie *in* potom jednoducho zistíme, či stlačený kláves bol z tejto množiny.

Ak teda chceme zistiť, či kláves bol z našej množiny, stačí použiť podmienku: *kláves in ['a', 'A', 'n', 'N']*.

Na vlastné riešenie pravdepodobnosti dobrého (zlého) partnerského vzťahu použijeme jednoduché zistenie, v koľkých prípadoch sa odpovede oboch partnerov na jednotlivé otázky zhodovali. Môžeme ich vyjadriť jednak absolútne, t. j. počtom zhodných odpovedí, alebo relatívne, napr. ako počet percentuálnej zhody.

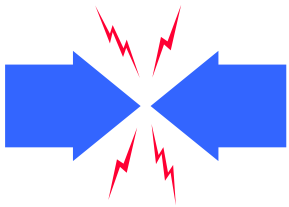
Výsledný program môže mať tvar:

```
Program PARTNERI;
uses   Crt;
const  pocet = 50;
var    i, j, PO, pocet : integer;
       OTAZKA : array [1..pocet] of string;
       ODP     : array [2, 1..pocet] of boolean;
       SUZVUK   : real;
       klaves   : char;

Begin
 {Doplniť otázky do pola OTAZKA - domáca úloha, ich počet bude PO};
 for i:=1 to 2 do
  begin                          {odpovedá 1., resp. 2. partner}
   ClrScr;
   for j:=1 to PO do                     {na j-tu otázku}
    begin
     WriteLn ('Test zhody partnerov, otazka ', j, '. ');
     WriteLn (OTAZKA[j], (Stlac a alebo n ');
     repeat
      klaves:= ReadKey;
     until klaves in ['a', 'A', 'n', 'N'];   {test stlačeného klávesu}
     if klaves='a' or klaves='A' then ODP[i,j]:= true
        else ODP[i,j]:= false;       {priradenie logickej hodnoty}
    end;
   end;                                          {koniec odpovedí}
 pocet:= 0;
 for i:=1 to PO do        {zistenie počtu rovnakých odpovedí na PO otázok}
 if ODP[1,i]=ODP[2,i] then pocet:= pocet+1;
 SUZVUK:= pocet/PO*100;
 ClrScr; WriteLn ('Test zhody partnerov - vyhodnotenie': 60);
 WriteLn;
 WriteLn('Z ', PO, 'otazok ste odpovedali ', pom, '-krat zhodne,
          co je priblizne', Trunc(SUZVUK), '% zhody');
 if SUZVUK>=50 then
         WriteLn ('Nevyzera to s Vami zle! Drzim vam vsetky klavesy! ')
    else
         WriteLn ('Byt Vami by som vazne porozmyslal o inom partnerovi! ');
 ReadLn;
End.
```

V programe je použitá reálna premenná *SUZVUK*. Jej hodnota sa bude pohybovať v rozmedzí 0 až 100, pre výpis je zaokrúhlená nadol pomocou funkcie *Trunc*, ktorá odreže desatinnú časť.

1. Doplňte do programu príslušné (napevno do poľa vložené) otázky a vyskúšajte jeho účinok. Skúste doplniť aj jemnejšiu škálu zhodnotení výsledkov testu.

7.18 Konečne si prečítam noviny!



Pre mnohých čitateľov to asi nie je ešte (pre autora už) aktuálne či akútne, ale vžite sa na chvíľu do pozície rodiča prváčika – druháčika na základnej škole. Prídete po celodennom naháňaní domov, vyložíte nohy podľa možností a rodinných obyčají čo najvyššie a slastne otvoríte noviny. Ale čo to? "Oci, máš ma preskúšať z príkladov na sčítanie z matematiky, povedala pani učiteľka. Aspoň dvadsať alebo radšej milión!" ozve sa pýcha rodu. "Choď za maminou!", je prvá obrana. "Už to tu bolo viackrát a vždy doplácam na svoje matematické vzdelanie!", mrmle si potichu patriarcha. "Ale ONA nemá čas a poslala ma za tebou!", znie rázna odpoveď, ktorá je súčasne príkazom a varovaním z tých najvyšších miest. Rodinná pohoda nadovšetko, dokonca aj nad pohodlie. Vymýšľate príklady, potom pracne zisťujete (s kalkulačkou skrytou pod novinami), či je odpoveď správna, dávate (najradšej variabilné) pochvaly, rady alebo varovania a pohoda sa zmenila na nepopulárnu, časovo i psychicky ubíjajúcu činnosť. Ako na to? Veď máme personálny počítač, ktorý nás má odbremeňovať od rutinných výpočtov a je to komunikatívny "tvor"! Dobrý nápad, skúsme napísať program, ktorý (dúfam, že len v tejto činnosti) nahradí hlavu rodiny.

Úloha už nie je tak jednoduchá a vyžaduje si podrobnejší rozbor. Vieme, že program by mal náhodne generovať N príkladov na sčítanie dvoch čísel. To nie je programovo náročné. Problematickejší je tzv. komunikačný modul, v rámci ktorého musíme trochu spestriť a zatraktívniť pre skúšaného túto nezáživnú činnosť.

Pri rozbore sa môžeme dostať k prvému priblíženiu – akémusi hrubému plánu riešenia. Mohol by mať takýto tvar:

```
ÚVOD;
ZADANIE POČTU PRÍKLADOV;
NASTAVENIE POČIATOČNÝCH HODNOT;
for i:= 1 to pocet do
 begin
  GENEROVANIE A VÝPIS PRÍKLADU;
  VSTUP ODPOVEDE;
  VYHODNOTENIE ODPOVEDE;
 end;
VYHODNOTENIE CELÉHO SKÚŠANIA;
ODHLÁSENIE.
```

Počítačových príkazov je v hrubom pláne riešenia málo, ale nič nám nebráni "rozmieňať" tieto činnosti "na drobné". Urobme si rozbor jednotlivých činností a vytvorme ich v tvare podprogramov alebo sekvencií príkazov. Nesmieme zabudnúť ani na prenos údajov medzi nimi.

Úvod

Mal by obsahovať predstavenie počítača a používateľa – činnosti, ktoré sú nám dôverne známe. Napíšme si jeden z možných variantov v tvare procedúry:

```
Procedure UVOD;

begin
 ClrScr;
 GoToXY(2,20); Write ('Ja som super skusaci stroj 5. generacie');
 GoToXY(4,20); Write ('Skusam scitanie cisel');
 GoToXY(6,20); Write ('A ty si kto? '); ReadLn (meno);
 GoToXY(10,20);Write(meno, 'vitaj! Tesim sa na spolupracu! ');
 GoToXY(20,60); Write ('OCO, 1997');
 Delay(3000); ClrScr;
end;
```

Zadanie počtu príkladov

Počet príkladov môže byť buď konštantný (napr. 10) alebo voliteľný podľa priania používateľa. Oboje má svoje výhody i nevýhody. Pri konštantnom počte príkladov je možné lepšie vyhodnotiť, napr. pri známkovaní, na druhej strane voľba možností dáva používateľovi vždy pocit vlastnej zaangažovanosti. Okrem toho sa aj programátorovi program lepšie odlaďuje.

V tomto prípade však nasaďme pevný počet 10 príkladov. Tento umiestnime do hlavného programu ako konštantu

```
const pocet = 10;
```

a v prípade potreby ju pri ladení programu môžeme zmeniť.

Nastavenie počiatočných hodnôt

Potrebujeme nastaviť počet správnych a nesprávnych odpovedí, rozsah generovaných čísiel, prípadne ďalšie dôležité hodnoty. Zrejme to budú hodnoty, ktoré majú opodstatnenie v celom programe, preto ich zvolíme ako globálne. Môžu to byť premenné:

DOBRE (na začiatku 0), *ZLE* (na začiatku 0), *OD* (pre nás 0), *PO* (pre nás 100); nezabudnúť na *Randomize*!

Generovanie a výpis príkladu

Pre zjednodušenie uvažujme prípad, keď generujeme 10 príkladov na sčítanie prirodzených čísiel od 0 po 100. Zdalo by sa, že riešenie môže byť v tvare:

```
Procedure GENERUJ;

begin
 A:= Random(100); B:= Random(100);                                    {*}
 C:= A + B;
end;
```

Ale to nie je dobre! Môže sa totiž stať, že hodnota A+B prekročí stanovený rozsah do 100, napr. A=97, B=12. Ani náhrada riadku {*} príkazmi

```
A:= Random(50); B:= Random(50);
```

nebude najšťastnejšia, pretože tým nepreskúšame prípady, kedy jedno z čísel je väčšie ako 50.

Používateľ nevidí, čo sa deje vo vnútri programu. Môžeme si preto dovoliť "fintu", ktorá umožní odstrániť nevýhody predchádzajúcich návrhov:

```
A:= Random(100); B:= Random (100);
if A<B then begin p:= A; A:=B; B:= p; end;
                    {nezabudnúť, že p je pomocná premenná – lokálna}
C:= A; A:= C-B;
```

V tomto prípade generujeme dve náhodné čísla od 0 po 99, väčšie z nich bude v premennej *A*, menšie v premennej *B*. Väčšia generovaná hodnota bude hodnotou súčtu *C* (aby nedošlo k záporným výsledkom), menšia bude jeden a rozdiel medzi nimi druhý sčítanec. Nebudeme teda oznamovať používateľovi priamo dve generované čísla, ale až takto zabezpečené hodnoty premenných *A*, *B*.

Výpis zadania príkladu by nemal byť problematický a môže slúžiť ako skúška vášho estetického cítenia. V každom prípade by malo byť oznámené, ktorý príklad ideme riešiť a aké sú sčítance.

Niekoľko námetov:

1. Klasický výpis pomocou kombinácie príkazov GoToXY, Write, WriteLn. (GoToXY môže chýbať.)

2. Každý sčítanec prezentujte pomocou "grafickej" formy, napr. tým, že okrem číselných hodnôt budú sčítance zobrazené aj pomocou nejakého symbolu (odporúčam označenie pomocou symbolu "|", kód #117). (Návod: Na "výpis" príslušného počtu symbolov si vyčleniť pre každý sčítanec (pre používateľom zadaný výsledok) pole 10x10 miest na obrazovke, pričom sa vypisujú po desiatkach, kým sa dá.)

Vstup odpovede

Predpokladajme, že používateľ má na obrazovke monitora zobrazené zadanie (i-teho) príkladu. Vstup odpovede môže byť zabezpečený dvomi spôsobmi:

1. Klasický vstup so zadaním hodnoty a stlačením *ENTER*.
2. Čítanie stlačených klávesov pomocou *ReadKey* (do premennej typu *char* sa vloží stlačený znak - oproti *ReadLn* má tú výhodu, že za každou odpoveďou nepotrebujeme stláčať *ENTER*) a vytváraním výsledného čísla. Tento má oproti predchádzajúcemu výhodu v tom, že môžeme "zablokovať" čítanie iných klávesov ako číselných, je však prácnejší. (Pre labužníkov by však bolo dobré mať takýto nástroj pre programovanie vo svojej sade.)

My si tu uveďme iba klasický vstup:

```
ReadLn (ODP);
```

kde *ODP* je číslo typu *integer* (*word*).

Vyhodnotenie odpovede

Je zrejmé, že odpoveď môže byť správna alebo nesprávna. (Neuvažujme prípad, kedy vstup "zbortí" funkčnosť programu.) Je však viac vecí, ktoré musíme zvážiť:

1. V prípade správnej odpovede stačí pochvala a môže sa prejsť na riešenie ďalšieho príkladu.
2. Horšie je to s nesprávnou odpoveďou. Musíme sa rozhodnúť pre niektorý z nasledujúcich variantov:

a) Skúšanému iba oznámime, že odpovedal nesprávne.

b) Skúšanému oznámime, že odpovedal nesprávne, a uvedieme správny výsledok.

c) Skúšanému oznámime, že odpovedal nesprávne a dáme mu možnosť opravy. Potom ale musíme uvažovať, koľkokrát môže odpovedať nesprávne (teoreticky veľakrát) a ako sa to premietne do celkového hodnotenia.

Nechám na vás, ktorú z možností si vyberiete. Prvé dve reakcie na nesprávnu odpoveď nie sú obtiažne. V treťom prípade sa najčastejšie postupuje tak, že skúšaný má ešte dve možnosti opravy, pričom do konečného hodnotenia už rátame nejaký "trestný bod". Navyše by mohol dostať nejakú nápovedu, napr. pri zadaní 38+43 takúto:

```
"30+40 = ?,         8+3 = ??      ? +?? = ??? "
```

Oznámenia o správnosti výsledku by sme mali taktiež trochu spestriť. Často sú totiž súčasťou motivácie. Môžeme to urobiť tak, že vymyslíme "hromadu" pochvál a varovaní a po každej odpovedi z nich budeme náhodne vyberať. Použime na to príkaz viacnásobného vetvenia *case*.

Tú istú úlohu môže plniť vytvorenie poľa reťazcov znakov pre pochvaly a poľa pre varovania.

```
Procedure VARUJ;
var nahoda: integer;

begin
 nahoda:= Random(10)+1;
 case nahoda of
    1: ret:= 'Co mi to robis? Ved to nie je dobre! ';
    2: ret:= 'Kto ma kontrolovat take nezmysly? ';
    3: ret:= 'To hadas alebo si zo mna robis dobry den? ';
    4: ret:= 'Tak toto ti vobec nevyslo! Trafil si vedla! ';
```

```
    5: ret:= 'Ale, ale! Ty si chybal v skole? ';
    6: ret:= 'Co to z teba bude? Takato chyba! ';
    7: ret:= 'Pomylil si sa! Ale nabuduce uz to vyjde! ';
    8: ret:= 'To hadam nie! Ja sa vypnem!!! ';
    9: ret:= 'Keby to rodicia vedeli, ale by ti dali! ';
   10: ret:= 'A ja som ti tak veril! Je to zle! Sustred sa! ';
 end;              {koniec príkazu case}
 ret:= meno + ', ' + ret;
 ZOBRAZ(ret); {doplňte výpis na príslušné miesto na obrazovke!}
end;

Procedure POCHVAL;
var nahoda: integer;

begin
 {doplňte podobné riešenie ako v procedúre VARUJ }
end;
```

Pozor na realizáciu výstupu reťazcovej premennej *ret* na obrazovku. Možno by bolo najlepšie deklarovať ju ako globálnu v programe. Výhodou využitia viacnásobného vetvenia *case* je možnosť pohodlne dopĺňať ďalšie a ďalšie pochvaly alebo varovania podľa nálady programátora.

Vyhodnotenie celého skúšania

Po skončení skúšania príslušného počtu príkladov nastáva čas zúčtovania. Ako ohodnotiť výsledky? Máme k dispozícii celkový počet príkladov, počet správnych a počet nesprávnych odpovedí. Najčastejšie sa používa hodnotenie známkou. To je veľmi relatívne, napr. v prípade fixného počtu príkladov 10, by to mohlo vyzerať takto: za 10–9 správnych odpovedí známka 1, 8–7 známka 2, 6–5 známka 3, 4–3 známka 4, 2–0 známka 5. Horšie je to, ak je počet príkladov voliteľný. Potom by sa mohlo vnútorne využiť rozdelenie na percentuálnu úspešnosť, napr. podľa vyššie uvedeného. Treba si však uvedomiť, že skúšanému nie je dobré predkladať hodnotenie v percentách – on ešte percentá nevie! Stálo by za úvahu aj zhodnotiť celkový výsledok uvedením počtu správnych a nesprávnych odpovedí a uvedením nejakého slovného hodnotenia podobného procedúram *POCHVAL* a *VARUJ*. Necháme to na vás; pred vytvorením tejto časti programu sa zamyslite nad tým, či má hodnotenie motivovať alebo má tvrdo konštatovať – známkovať.

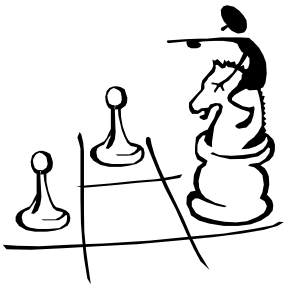
Odhlásenie

Podobná úloha ako *ÚVOD*. Fantázii sa medze nekladú. Stále by ste však mali mať na pamäti, že program má byť "user friendly" (čítaj "júzr frendly" – priateľský k používateľovi). Nemal by jednoducho skončiť, ale možno by bolo dobré spýtať sa na možnosť skúsiť znova, poprípade sa pekne (možno aj efektne) rozlúčiť.

1. Zostavte celý program z tu popísaných fragmentov (častí). Bude to pre vás určite hračka a lahôdka.

2. Zostavte program na generovanie N príkladov pre zvolenú aritmetickú operáciu (+, -, x,:) a zvolený rozsah čísel s vyhodnotením správnosti odpovedí.

7.19 Hráte šach?



Oddávna veľmi populárna hra šach vznikla v 6. storočí v Indii, potom sa rozšírila do Perzie (jej panovník mal titul šach) a Arabi ju priniesli do Európy niekedy v 10.– 11. storočí. Dodnes patrí k najuznávanejším intelektuálnym hrám. Hrá sa na doske so 64 poliami, ktoré sú striedavo biele a čierne. Dvojica hráčov má za cieľ pomocou rôznych figúr bielej a čiernej farby pripomínajúcich vojsko vyhodiť (zmatovať) súperovho kráľa. Nebudeme si tu vytvárať program, ktorý by hral víťaznú stratégiu. Nijaká totiž zatiaľ nebola nájdená. Napriek svojej rýchlosti počítač nedokáže prepočítať všetky možné ťahy tak, aby vždy vyhral[23]. Na druhej strane sa stalo práve hľadanie počítačového programu, ktorý by dôstojne súperil s najlepšími šachistami sveta, inšpiráciou pre mnohé efektívne metódy programovania. (V roku 1997 počítačový program Deeper Blue zohral vyrovnanú sériu zápasov s majstrom sveta v šachu G. Kasparovom)

Zostavme program, ktorý by zisťoval a zobrazoval na zjednodušenej šachovnici, ktoré polia šachovnice ohrozujú figúrky: a) veža, b) strelec, c) dáma, ťahajúce podľa pravidiel šachovej hry

Zjednodušíme zadanie v tom, že neuvažujeme farebné rozlíšenie políčok šachovnice. Ďalej uvažujme, že okrem tejto figúrky nebude na šachovnici žiadna iná. Potom by sa totiž úloha značne skomplikovala. Ak to neviete, potom vedzte, že:

a) *veža* môže ohroziť políčka v tom riadku a stĺpci šachovnice, v priesečníku ktorých sa nachádza,

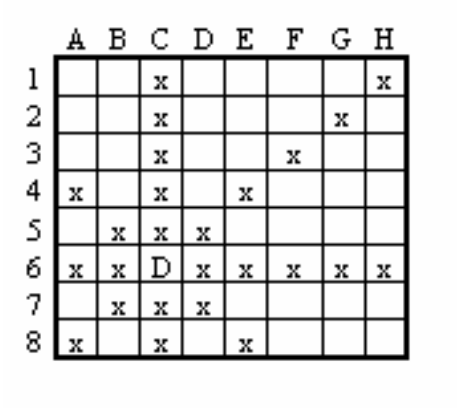
b) *strelec* môže ohroziť všetky políčka, ktoré sa nachádzajú na diagonálach, v priesečníku ktorých stojí,

c) *dáma* má kombinované možnosti veže aj strelca dokopy.

Tu sú príklady možných ťahov (poloha figúry je daná písmenom V–veža, S–strelec, D–dáma, ťahy znakom x):

	A	B	C	D	E	F	G	H
1					x			
2					x			
3					x			
4	x	x	x	x	V	x	x	x
5					x			
6					x			
7					x			
8					x			

	A	B	C	D	E	F	G	H
1		x		x				
2			S					
3		x		x				
4	x				x			
5						x		
6							x	
7								x
8								



Vo všetkých prípadoch budeme potrebovať pozíciu figúrky na šachovnici. Pre zjednodušenie budeme pozíciu zadávať dvojicou čísiel od 1 po 8, kde 1. číslo značí riadok, 2. číslo stĺpec, v priesečníku ktorých sa figúrka nachádza.

Na reprezentáciu výsledkov by sme mohli použiť pole 8x8 znakov, ktoré by v danej pozícii obsahovalo písmeno, označujúce figúrku (V, S, D), políčka, na ktoré figúrka "nedosiahne", označíme bodkou a políčka, ktoré sú "v ohrození" znakom "x". Na začiatku budú všade bodky, na pozícii figúrky jej znak.

Stačí už "len" zvoliť vhodný spôsob zisťovania, ktoré políčko je v ohrození.

Ako príklad uvedieme ten ťažší (!!) - strelca.

Predpokladajme, že je daná pozícia strelca ako súradnice *riadok, stlpec*. Jedno z možných riešení je nasledovné:

Zvolíme si pomocné premenné *r*, *s*, ktoré budú označovať skúmanú pozíciu v poli *A* rozmerov 8x8. Toto budeme posúvať štyrmi "šikmými" smermi a na týchto miestach meniť znaky:

a) doľava hore (kým nedôjdeme na prvý riadok alebo prvý stĺpec),
b) doľava dole (kým nedôjdeme na posledný riadok alebo prvý stĺpec),
c) doprava hore (kým nedôjdeme na prvý riadok alebo posledný stĺpec),
d) doprava dole (kým nedôjdeme na posledný riadok alebo posledný stĺpec).

Zapíšeme iba časť (aj keď dlhšiu) programu v tvare procedúr riešiacich úlohu:

```
Program STRELEC;
var    riadok, stlpec, r, s, i, j  : 1..8;
       moze   : boolean;
       A      : array [1..8,1..8] of char;

Procedure VlavoHore;

begin
 if (r=1) or (s=1) then moze:= false
     else moze:= true;
 while moze do
  begin
   r:= r-1; s:= s-1; A[r,p]:= 'x';
   if (r=1) or (s=1) then moze:= false
      else moze:= true;
  end;
end;

Procedure VlavoDole;

begin
 if (r=8) or (s=1) then moze:= false
    else moze:= true;
```

```
 while moze do
  begin
   r:= r+1; s:= s-1; A[r,p]:= 'x';
   if (r=8) or (s=1) then moze:= false
      else moze:= true;
  end;
end;

Procedure VpravoHore;

begin
 if (r=1) or (s=8) then moze:= false
    else moze:= true;
 while moze do
  begin
   r:= r-1; s:= s+1; A[r,p]:= 'x';
   if (r=1) or (s=8) then moze:= false
      else moze:= true;
  end;
end;
Procedure VpravoDole;

begin
 if (r=8) or (s=8) then moze:= false
    else moze:= true;
 while moze do
  begin
   r:= r+1; s:= s+1; A[r,p]:= 'x';
   if (r=8) or (s=8) then moze:= false
      else moze:= true;
  end;
end;

Begin
 ReadLn(riadok,stlpec);
 for i:=1 to 8 do
  for j:=1 to 8 do A[i,j]:='.';
 A[riadok,stlpec]:='S';
 VlavoHore;
 VlavoDole;
 VpravoHore;
 VpravoDole;
 {Vypis sachovnice}
End.
```

Logická premenná *moze* (čítaj môže) je signalizáciou skutočnosti, či sme nedosiahli okraj šachovnice. Pokojne by mohla byť riešená ako lokálna premenná, podobne ako premenné *r* a *s*. Na prvý pohľad je zrejmé, že uvedené procedúry sú podobné – menia sa "iba" hranice ukončenia cyklu *while* a to, či sa zvyšuje alebo znižuje hodnota *r* alebo *s*. Zrejme by bolo možné uvažovať o spojení všetkých štyroch procedúr do jednej (napr. *Smer*), ktorá však musí mať vhodné parametre. Pokúste sa o to!

1. Napíšte programy, ktoré budú zisťovať políčka ohrozené a) vežou, b) dámou (spojením možných ťahov strelca a veže).

2. Neuvažujme teraz figúrky na šachovnici, ale šachovnicu ako hracie pole. Zostavte program, ktorý pre zadanú hodnotu stĺpca (A–H) a riadku (1–8) zistí, akej farby je dané pole. Predpokladajme, že A1 je pole farby bielej, A2 čiernej, ..., B1 čiernej, B2 bielej...

3. Na základe vyššie uvedeného skúste "nakresliť" šachovnicu (bez figúr) tak, že znakom 'W' (white) označíte jej polia bielej farby, znakom 'B' (black) polia čiernej farby. Môžete to skúsiť aj pomocou farieb pozadia v režime Crt bez znakov – vypísaním medzery na príslušnom mieste obrazovky. Toto sa dá využiť aj v riešeniach ďalších úloh.

4. Spojte riešené úlohy tak, aby užívateľ programu zadával nielen polohu figúrky, ale aj jej typ (D, V, S).

5. Okrem zvolenej figúrky sa na šachovnici objavujú aj ďalšie. Môžeme ich zadať ako "prekážky". Zostavte pre zvolenú figúrku program, ktorý určí ako "prístupné" iba tie políčka šachovnice, ktoré sú po prvú "prekážku" v danom smere. (Môžete uvažovať aj políčko, na ktorom je prekážka. Ale čo ak je to vlastná figúrka?) Vidieť, že aj šach môže byť náročným športom – aspoň pre programátora.

6. Kôň je kráľovské zviera, a preto patrí aj do šachu. Má zvláštny pohyb - preskakuje polia krokom podobným písmenu L. Zistite (ak neviete), ako sa šachový kôň pohybuje po šachovnici, a zostavte program, ktorý vykreslí všetky políčka šachovnice, na ktoré môže kôň z danej pozície skočiť. Aj tu je ďalších úloh nepreberné množstvo – stačí si ich iba vymyslieť a... vyriešiť.

7.20 Koľko je obyvateľov? Sčítanie veľkých čísel

Už bolo spomínané, že veľké čísla sa na počítači správajú nie veľmi dobre. A nás (mňa) by zaujímali aj veľmi veľké a presné výsledky.

Počet obyvateľov veľkých miest je vyjadriteľný iba veľkými číslami. V čase sčítania obyvateľstva je hlavnou úlohou zistiť počet obyvateľov štátu. Sčítajú sa celé čísla, ktoré môžu byť veľmi veľké. Uvažujme preto úlohu zostaviť program, ktorý sčíta čísla väčšie, ako je možné uložiť v premenných štandardných numerických typov.

Človek môže chápať číslo dvomi spôsobmi a takmer medzi nimi nerozlišuje: jednak ako nedeliteľnú číselnú hodnotu, jednak ako postupnosť znakov – číslic. Počítač však rozlišuje iba prvý prípad a "rozbitie" čísla na skupinu číslic ho musíme naučiť. Opäť môžeme objaviť viacero ciest vedúcich k riešeniu:

a) Vložíme a ďalej spracúvame čísla ako postupnosti (polia) číslic. Pritom musíme vhodne zvážiť vstup – ako zabezpečiť to, že čísla majú rôzny počet číslic, vkladať ich v bežnom tvare zľava doprava apod.

b) Vložíme čísla ako reťazce znakov a potom ich rozložíme znak po znaku zľava doprava na číslice do poľa číslic. Tento prípad nevyžaduje, aby užívateľ nejako signalizoval počet číslic čísla. Polia číslic potom spracúvame postupným sčítaním s uvažovaním prenosu.

c) Vložíme čísla ako reťazce znakov a aj ako reťazce ich spracúvame a výsledok ukladáme do výsledného reťazca.

Samotný algoritmus sčítania nie je zložitý. Väčšinu programu by mali tvoriť vloženie čísel do polí, ich sčítanie a výstup vo vhodnom tvare na obrazovku.

Zostavte programy na sčítanie veľkých čísel podľa návodov uvedených v bodoch a), b), c).

Koľko číslic má faktoriál čísla 100?

Potrebovali by sme získať presnú hodnotu "veľmi veľkého" čísla – 100! (čítaj "sto-faktoriál", t.j. 100.99.98.97.96.95.94. ... .3.2.1).

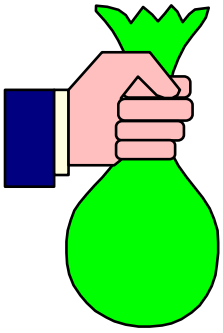
Uvažujme o tom, že vstupom môže byť ľubovoľné prirodzené číslo od 1 po 1000. Jediná možnosť riešenia spočíva v tom, že získaný výsledok rozdelíme na pole číslic a podľa potreby ho budeme násobiť "číslicu po číslici" ďalšou a ďalšou hodnotou – naučíme počítač násobiť ako človeka. Bez ďalších komentárov si preto skúste prečítať a zvládnuť tento program:

```
Program N_FAKTORIAL_POLOM_ver03;
const MAX=1000;
type  rozsah=0..9;
var   i, j, prenos, sucin, naj, N: integer;
      cislica: array[0..MAX] of rozsah;

Begin                                   {hlavny program}
 Write('Zadaj N: '); ReadLn(N);
 repeat
  for i:=0 to MAX do cislica[i]:= 0;    {nulovanie cislic}
  cislica[0]:= 1; naj:= 0;
  for i:=1 to N do begin          {hlavny cyklus i! }
   prenos:= 0;
   for j:=0 to naj do begin             {spracovanie číslic}
    sucin:=cislica[j]*i+prenos;
    prenos:= sucin div 10;
    cislica[j]:= sucin mod 10;
   end;                                 {for j}
   while prenos<>0 do begin             {doplnenie prenosu}
```

```
    naj:= naj+1;
    cislica[naj]:= prenos mod 10;
    prenos:= prenos div 10;
   end; {while}
  end;                                  {for i}
  Write(N, '! = ');
  j:= MAX; while cislica[j]=0 do j:= j-1;
  for i:=j downto 0 do Write(cislica[i]);
  WriteLn;
  WriteLn(' Pocet cislic ', j+1);
  Write('Zadaj N (pre ukoncenie 0): '); ReadLn(N);
 until N=0;  {koniec pri N=0}
 ReadLn;
end.
```

7.21 Máte peniaze? Sem s nimi!

Fy. Jánošík a kolektív – Vaša banka



Toto motto nie je charakteristické iba pre stredovekých zbojníkov. V dnešnom tzv. civilizovanom svete existuje veľa dôkladne premyslených spôsobov ako pripraviť ľudí o peniaze. Medzi špecialistov patria najmä peňažné ustanovizne nazvané banky. Narozdiel od zbojníkov však nepoužívajú len dômyselné nástroje na "zbíjanie" – pôžičky na určitý, nie malý úrok, ale aj zdanlivo (možno skutočne, ale o tom sa musíte presvedčiť sami) výhodné ponuky pre klientov – ukladanie peňazí s možnosťou ich zúročenia. Navyše ich služby (narozdiel od služieb zbojníkov, ktorí vás pristavia na nejakom mieste) nemusíte použiť. Možno je vhodné zamyslieť sa, či by bolo výhodné uložiť si v banke svoje bohaté úspory zo štipendia alebo vreckového a zveľadiť ich do šesťmiestnych hodnôt.

Zostavme program, ktorý bude pre danú vloženú sumu a vopred stanovený úrok za určité obdobie zisťovať výsledok sporenia k istému termínu.

Banky ponúkajú veľké množstvo rôznych foriem sporenia. Preto musíme našu úlohu zjednodušiť. Základom je vždy vklad – množstvo vložených peňazí a percento zúročenia vkladu za vopred stanovené obdobie. Aby sme nekomplikovali zadanie, dohodnime sa, že úrok sa vždy pripíše po skončení stanoveného obdobia (môže to byť mesiac, najčastejšie rok), pritom budeme uvažovať iba celé násobky období. Navyše predpokladajme, že ide o jednorázový vklad.

1. priblíženie: Zostavíme program, ktorý pre daný vklad a úrok bude na príslušný počet rokov určovať, ako sa bude zvyšovať nami vložená suma.

Môžme využiť cyklus so známym počtom opakovaní (počet rokov), na konci každého z nich vypíšeme sumu peňazí, ktorá je momentálne na našom účte. Riešením môže byť takýto program:

```
Program UROKY_01;
var   urok, i, pocet : integer;
      suma, celkovo, prir  : real;

Begin
 ReadLn (suma, urok, pocet);
 celkovo:= suma;
 for i:=1 to pocet do
  begin
   prir:= celkovo*urok/100;
   celkovo:= celkovo + prir;
   WriteLn (i, '. obdobie, prirastok = ', prir, ', celkovo = ', celkovo);
  end;
 ReadLn;
End.
```

Program neobsahuje žiadne komentáre o vstupe. Ani celková suma nie je veľmi šťastne riešená: väčšinou banky zaokrúhľujú na haliere, t. j. na dve desatinné miesta, a to dole. Doplňte toto obmedzenie a upravte aj výpis hodnôt vo vhodnom tvare.

Podľa uvedeného programu počítame jednotlivé hodnoty rok po roku, až kým sa nedostaneme k celkovej sume. Aby bolo možné okamžite určiť, akú celkovú sumu získame po N rokoch, používa sa v bankách jednoduchý vzorec

$$S = S_o\ (1 + p)^N$$

kde S_o je vkladaná hodnota, p – úrok (v desatinných číslach zodpovedajúcich príslušným percentám, teda 0,02 = 2%), N – počet rokov (resp. iných dohovorených časových období).

Ak je N = 1, tak výsledná suma *S* bude dávať hodnotu úspor po 1. roku sporenia. Využitím vzorca môžeme veľmi jednoducho riešiť napr. nasledujúce úlohy:

a. Za koľko rokov pri rovnakej úrokovej miere sa vklad zväčší dvojnásobne, resp. *k*–násobne?
b. Aký percentuálny úrok musí byť, aby sa vklad zväčšil dvojnásobne za N rokov?
c. Aký má byť počiatočný vklad, aby sme po N rokoch získali nejakú vopred zvolenú sumu *S*?

Tieto úlohy nemusíme nechať len pre matematikov. Môžete sa pokúsiť zostaviť programy, ktoré budú nejakým spôsobom umožňovať ich riešenie. Aspoň približne alebo experimentálne.

Všetky tieto úlohy musia predpokladať, že sa môže pracovať aj s veľmi veľkými číslami – miliónmi, miliardami peňažných jednotiek. Preto je nevyhnutné (aspoň v bankách) pre riešenie používať aritmetiku čísiel s veľkým počtom číslic. Pritom musíme uvažovať, že spracúvať sa budú nie celé, ale desatinné čísla zaokrúhlené na dve desatinné miesta (haliere, centy...). Preto skúsme zostaviť program na násobenie veľkých čísel.

Násobenie veľkých čísel

Počítač by nám mal pomôcť pracovať aj s veľmi veľkými číslami. Ale aká je skutočnosť? Presne ráta dokonca na menej platných miest ako lepšia (tzv. vedecká) kalkulačka. A to nie je zvlášť v bankovej sfére, kde každý "halier" hrá svoju úlohu, vhodné. Preto je dobré, ak si uvedieme program na násobenie veľkých čísiel. Násobenie budeme uskutočňovať tak, že budeme "násobiť" dva reťazce číslic.

V programe využijeme sčítanie veľkých čísel z časti Sčítanie obyvateľov.

```
Program VELKE_CISLA_NASOBENIE;
uses  Crt;
var   a, b, product, item, s: string;
      na, nb : array [1..500] of byte;
      i, j, code, La, Lb, c : integer;
      Lmax, Lmin, rp : integer;

Function Long_Adding (a,b:string):string;
 {doplniť zo Sčítania obyvateľov}

begin
 {doplniť zo Sčítania obyvateľov}
end;

Begin
 ClrScr; ReadLn (a, b);
 La:= Length (a); Lb:= Length (b);
 if La>Lb then
  begin
   for i:=La downto 1 do Val (a[i],na[La-i+1],code);
   for i:=Lb downto 1 do Val (b[i],nb[Lb-i+1],code);
   Lmax:= La; Lmin:= Lb;
  end
  else
  begin
   for i:=La downto 1 do Val (a[i],nb[La-i+1],code);
   for i:=Lb downto 1 do Val (b[i],na[Lb-i+1],code);
   Lmax:= Lb; Lmin:= La;
  end;
 product:= '0'; item:= ''; rp:= 0;
 for i:=1 to Lmin do
  begin
   for j:=1 to Lmax do
    begin
     c:= na[j]*nb[i]+rp;
     rp:= c div 10;
     Str (c mod 10, s);
```

```
      item:= s + item;
     end;
    if rp<>0 then
     begin
      Str (rp, s);
      item:= s + item;
     end;
    for j:=1 to i-1 do item:= item + '0';
    product:= Long_Adding (product, item);
    rp:= 0; item:= '';
   end;
 WriteLn; Write (product);
 ReadLn;
End.
```

V programe sú použité tieto premenné:
a, *b*, *product* – činitele a súčin v tvare reťazcov číslic,
item – premenná pre uchovanie medzivýsledných súčinov,
na, *nb* – polia, ktoré obsahujú dané čísla číslicu po číslici,
La, *Lb* – dĺžky činiteľov,
Lmin, *Lmax* – dĺžky činiteľov, ktoré majú menší a väčší počet číslic,
rp – hodnota prenosu,
i, *j*, *code*, *c* – pomocné premenné.

Po vstupe činiteľov sa do poľa *na* číslicu po číslici vkladá ten z nich, ktorý má väčší počet číslic. Druhý činiteľ sa ukladá do poľa *nb*.

Potom sa číslicu po číslici vykonáva násobenie dlhšieho čísla s kratším po číslociach s uvažovaním prenosu. Získané medzivýsledky (ich počet je rovný počtu číslic v kratšom činiteli) sa sčítavajú s výsledkom súčtov z predchádzajúcich krokoch. Sčítanie sa vykonáva pomocou funkcie *Long_Adding* ("dlhé sčítanie"). Pred sčítaním musíme k *j*-temu činiteľovi pridať sprava *j-1* číslic '0'. Takým istým spôsobom sa zabezpečí nevyhnutný posun postupných medzivýsledných súčinov.

1. Ako súvisí toto násobenie s možnosťou násobiť desatinné čísla? Ak budeme uvažovať o tom, že vstupné čísla majú na posledných dvoch miestach desatinné číslice, potom výsledok bude mať štyri desatinné miesta a tie je potrebné zaokrúhliť, resp. "odrezať" na výsledné dve desatinné miesta. Upravte program tak, aby pre dve zadané čísla, kde posledné dve číslice sú uvažované ako desatinné, dával výsledok ich súčinu s dvomi desatinnými miestami.

2. Zostavte program, ktorý bude pracovať ako kalkulačka na operácie s veľkými číslami. Stačí, ak bude počítať iba základné aritmetické operácie s celými číslami – sčítanie, odčítanie, násobenie a delenie (celočíselné alebo reálne) a voľby užívateľa, vstup čísiel a výstup výsledkov vhodne prezentovať na obrazovke.

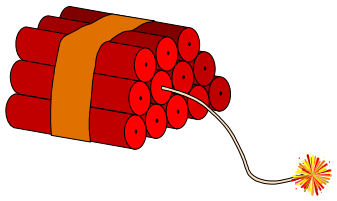
7.22 Horíí!

Hrať sa so zápalkami nám vždy zakazovali. Teraz máme jedinečnú možnosť, dokonca povinnosť skúsiť to. Hry typu NIM (z nemeckého nehmen = brať) sú totiž založené na odoberaní zápaliek z kôpky. Existuje veľa verzií tejto hry, my si ukážeme len niektoré.

Hra začína zvoleným počtom zápaliek na kôpke na stole. Potom sa pravidelne striedajú dvaja hráči v odoberaní zápaliek z kôpky. Pravidlá, ktoré sa tiež volia, určujú, koľko zápaliek môže každý hráč z kôpky odoberať naraz. U niektorých verzií tejto hry záleží počet odoberaných zápaliek tiež na počte, ktorý hráč odobral v predchádzajúcom ťahu, alebo na počte, ktorý odobral súper, apod. Tu si uveďme iba jednoduchšiu možnosť, kedy sa zadáva len to, koľko zápaliek maximálne, prípadne koľko minimálne môže hráč odobrať v jednom ťahu.

Víťazstvo alebo prehra v hre sú závislé od toho, kto odoberie poslednú zápalku. Aj tu sú dve opačné možnosti: víťazí ten, kto poslednú zápalku odoberie alebo ten, kto ju neodoberie.

Pre hru NIM sa dá väčšinou nájsť stratégia, ktorá vedie vždy k víťazstvu. Takúto stratégiu môže mať "zabudovanú" vždy počítač, narozdiel od hráča, ktorý optimálnu stratégiu môže, ale tiež nemusí poznať. Avšak vo všetkých programoch môže mať hráč výhodu prvého ťahu. Bohuži aľ, ani táto výhoda nemusí vždy viesť k výhre aj pri znalosti vyhrávajúcej stratégie. Závisí totiž aj od počtu zápaliek na kôpke a na zvolenom odoberanom počte zápaliek v jednom ťahu.



Predpokladajme, že hráč, ktorý zoberie poslednú zápalku, vyhráva. Ďalej budeme predpokladať, že je možné odoberať v jednom ťahu jednu až vopred stanovený počet zápaliek, ktorý bude zadaný používateľom zo vstupu. Rovnako je možné zadávať aj počet zápaliek na kôpke – buď zo vstupu alebo náhodne generovaný.

Stanovme *stratégiu počítača*: Ak má vyhrať, musí odobrať vždy toľko zápaliek, aby ich na stole zostal nejaký násobok počtu maximálne odoberaných zápaliek zvýšeného o jedna. Napr. ak je počet zápaliek pred ťahom počítača rovný 13 a maximálny počet odoberaných zápaliek je 3, realizácia stratégie vyzerá takto: Počítač sa musí dostať na násobky 4 (= maximálny počet odoberaných zápaliek 3 +1):

počet na kôpke	hráč berie	zostáva	počítač berie	na kôpke zostáva
13	už bral	13	1	12
12	1	11	3	8
	2	10	2	8
	3	9	1	8
8	1	7	3	4
	2	6	2	4
	3	5	1	4
4	1	3	3	0
	2	2	2	0
	3	1	1	0

V prípade, že hráč pozná stratégiu, počítač môže iba náhodne ťahať a čakať na chybu hráča, prípadne iba priznať prehru a pochváliť hráča za bezchybnú stratégiu.

Program "bez príkras" môže mať tvar:

```
Program NIM_0001;
var   POCET, MAX, pomoc, hrac, poc, i: integer;
```

```
      koniec                        : boolean;
      vitaz                         : string;

Begin
 Write (‘ Hra NIM, zadaj pocet zapaliek na kope a maximum pre tah ‘);
 ReadLn (POCET, MAX);
 koniec:= false; i:= 1; Randomize;
 WriteLn (‘Zaciname s ‘, POCET, ‘ zapalkami, najviac mozno odobrat ‘, MAX);
 repeat
  Write (i, ‘ tah: Beries ‘) ReadLn (hrac);
  if POCET=0 then begin koniec:= true; vitaz:= ‘hrac’; end
                                                  {výhra hráča}
     else begin
           pomoc:= (POCET div (MAX+1)) * (MAX+1);
                    {zistenie najbližšieho nižšieho násobku}
           if POCET-pomoc<=MAX then poc:= POCET-pomoc
                                  {počítač ide na výhru}
              else poc:= Random(MAX)+1;
                           {náhodný ťah a čaká na chybu}
             WriteLn (‘Ja beriem ‘, poc, ‘ Zostalo ‘, POCET, ‘ zapaliek ‘ );
           if POCET=0 then
              begin koniec:= true; vitaz:= ‘pocitac’; end;
                                          {výhra počítača}
         end;
  i:= i+1;                      {počítadlo ťahov}
until koniec;
 WriteLn; Write (‘Tuto hru vyhral ‘, vitaz);
 if vitaz = ‘hrac’ then WriteLn (‘ Gratulujem! ‘)
    else WriteLn (‘ Som proste lepsi! ‘);
 ReadLn;
End.
```

Premenná *pomoc* vyzerá čudne; v normálnej reči však zisťuje najbližší nižší násobok hodnoty *MAX* zvýšenej o jedna.

Napr. ak *POČET=10*, *MAX=3*, potom *pocet=(10 div 4)*4= 2*4 = 8*. Počítač “zacíti šancu na výhru” a už ju nepustí. Ak by hráč hral vyhrávajúcu stratégiu, potom počítač ťahá náhodne z intervalu *1..MAX*.

1. Program nezabezpečuje kontrolu toho, či hráč dodržiava pravidlá. Nekontroluje ani neférzadanie, kedy si hráč zvolí počet zápaliek na kope hneď menší ako maximálny ťah. Odstráňte tieto nedostatky. Môžete tiež upraviť začiatok hry na náhodné generovanie rozsahu premenných POCET a MAX a náhodné zvolenie toho, ktorý hráč začína ako prvý.

2. Zostavte program, ktorý bude hrať “opačnú” hru, teda vyhráva ten, kto prinúti súpera zobrať poslednú zápalku. Premyslite stratégiu počítača, aby vyhrával, ak mu to hráč dovolí svojim (chybným) ťahom. Skúste si analyzovať situáciu podľa predchádzajúceho riešenia a na niekoľkých príkladoch.

3. Skúste zostaviť program, ktorý pripúšťa aj nerozhodný výsledok v hre NIM v prípade, že okrem počtu zápaliek a maximálneho možného počtu odoberaných zápaliek sa udáva aj minimálny počet odoberaných zápaliek. Budeme predpokladať, že hráč, ktorý berie poslednú zápalku, vyhráva. Táto hra môže skončiť aj nerozhodne a to vtedy, keď na stole zostane menší počet zápaliek, ako je počet minimálne odoberaných.

7.23 Keby to Pascal vedel?

Jeden zo zaujímavých a všeobecne známych matematických problémov sa nazýva pascalov trojuholník[1]. V grafickom tvare nám umožňuje zistiť tzv. binomické koeficienty, resp. kombinačné čísla. Nebudeme sa podrobne zaoberať ich matematickým využitím, skúsme zostaviť program, ktorý bude pascalov trojuholník vypočítavať za nás.

Ak si dobre pamätáte zo školy, pascalov trojuholník má tvar:

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5  10  10   5   1
```

Na obvode trojuholníka sú jednotky. Každé ďalšie číslo sa vypočíta ako súčet dvoch nad ním ležiacich čísiel. Pretože asi ťažko môžeme trojuholník v počítači reprezentovať tak ako na papieri, rozhodnime sa pre dvojrozmerné pole v tvare "pootočeného" trojuholníka:

```
1   1   1   1   1   1
1   2   3   4   5   .
1   3   6  10   .   .
1   4  10   .   .   .
1   5   .   .   .   .
1   .   .   .   .   .
```

Kde sú uvedené bodky, tam čísla v trojuholníku nepoznáme. (Poznáme, ale už by to nebol trojuholník.)

Môžeme napísať program pre výpočet a zobrazenie prvých šiestich riadkov Pascalovho trojuholníka. Komentármi sú dostatočne opísané jeho jednotlivé činnosti. Výpočet prvkov Pascalovho trojuholníka sa vykonáva pomocou tzv. rekurzívneho vzorca[2]:

$$a_{i,j} = a_{i-1,j-1} + a_{i-1,j}$$

```
Program PASCAL_TROJUHOLNIK;
const n = 6;
var   a : array [1..n,0..n] of integer;
      i, j: integer;

Begin
 for i:=1 to n do           {začiatok pomocného cyklu}
  begin
   a[i,0]:=0;
   a[i,i+1]:=0;
  end;                      {koniec pomocného cyklu}
 a[1,1]:=1;                 {priradeni hodnoty "vrcholu"}
 WriteLn (a[1,1]:4);        {výpis hodnoty "vrcholu"}
for i:=2 to n do            {začiatok cyklu pre výpočet trojuholníka}
```

[1] Pascalov trojuholník – schéma pre výpočet binomických koeficientov, ktorú zverejnil v r. 1653 B. Pascal, ale bola známa už matematikom v Číne okolo r. 1300, ba dokonca aj v Indii okolo r. 1150. (A potom aby človek vymýšľal niečo nové! Stačí len poriadne pozerať okolo.)

[2] Rekurzívny vzorec na zistenie istej hodnoty je predpis, kedy je táto hodnota vypočítavaná "pomocou samej seba". V tomto prípade je prvok pascalovho trojuholníka zisťovaný ako súčet dvoch "vyšších" prvkov toho istého trojuholníka. Takýto prístup k riešeniu problémov sa nazýva rekurzia a aj keď jej využitie je najčastejšie v matematike, existujú aj prípady z bežného života, ktoré majú veľa spoločného s rekurziou. Poznáte takúto rozprávku? "V jednej jaskyni sedeli pri ohni zbojníci. A zbojnícky kapitán začal rozprávať: "V jednej jaskyni sedeli pri ohni zbojníci. A zbojnícky kapitán začal rozprávať: "V jednej... """

```
 begin
  for j:=1 to i do
   begin
    a[i,j]:= a[i-1,j-1]+a[i-1,j];      {výpočetnasledujúceho prvku}
    Write (a[i,j]: 4);                    {výpis hodnoty prvku}
   end;
  WriteLn;                             {prechod na nový riadok}
 end;                          {koniec cyklu pre výpočet troujuholníka}
 ReadLn;
End.
```

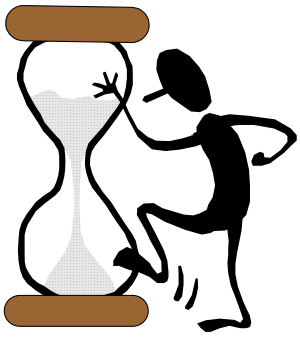
Doplňte program tak, aby vypočítaval a vypisoval hodnoty prvkov Pascalovho trojuholníka pre používateľom zvolený počet riadkov N, kde N<20.

Skúste zabezpečiť, aby sa pascalov trojuholník zobrazil tak, ako má v skutočnosti vyzerať.

Myšlienka výstupu na obrazovku je založená na tom, že prvé číslo sa zobrazí v strede riadku obrazovky a všetky nasledujúce riadky trojuholníka sa zobrazujú s posunom vľavo o 2 pozície.

```
Program PASCAL_TROJ_02;
const m=25;
var a: array[1..m, 0..m] of integer;
      i, j, N, k, r: integer;

Begin
 Write ('Zadaj N '); ReadLn (N);
 for i:=1 to N do
  begin
   a[i,0]:=0; a[i,i+1]:=0;
  end;
 a[1,1]:=1; r:=40;            {prvá hodnota r=40, stred riadku}
 for k:=1 to r do Write (' ');       {na obrazovke bude 40 medzier}
 r:=r-2;                             {r sa znižuje o 2}
 WriteLn (a[1,1]: 4);{zobrazenie prvého prvku v strede riadku}
 for i:=2 to n do
  begin
   for k:=1 to r do Write (' '); {znižujúci sa počet medzier}
   r:=r-2;
   for j:=1 to i do
    begin
     a[i,j]:= a[i-1,j-1] + a[i-1,j];
     Write(a[i,j]: 4);
    end;
   WriteLn;
  end;
 ReadLn;
End.
```

7.24 Čas prideľuje každému rovnako



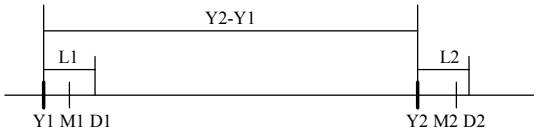
Gregoriánsky[1] kalendár, ktorý dnes používame, má veľa nevýhod. Mýlia sa prestupné a neprestupné roky, mesiace majú "čudný" systém počtu dní, zistiť počet dní (mesiacov, rokov) medzi dvomi dátumami je často problém. A práve tieto problémy nás zaujímajú...

Úlohy, v ktorých používame kalendár, stretávame na každom kroku:

- Koľko dní zostáva do konca školského roku alebo do Nového roku?
- Koľko dní nás už delí od nášho dôležitého rozhodnutia v minulosti?
- Koľko dní je medzi dvomi dátumami alebo aká je dĺžka exkurzie či dovolenky?
- Aký deň v týždni bude v istý dátum v budúcnosti? Ktorý deň v týždni pripadne na 2. 2. 2222?
- O koľko rokov bude kalendár terajšieho roku, napr. 1998, použiteľný z hľadiska toho, že zodpovedajú dátumy a dni v týždni?

Počet dní medzi dvomi dátumami

Pokúsme sa zistiť počet dní, ktoré uplynuli medzi dvomi udalosťami. Pozrime si nasledujúcu schému riešenia:



Označenia na obrázku:
Y1 – rok, v ktorom sa uskutočnila prvá z udalostí,
M1 – mesiac, v ktorom sa uskutočnila prvá z udalostí,
D1 – deň, v ktorom sa uskutočnila prvá z udalostí,
Y2, *M2*, *D2* – tie isté údaje pre druhú z udalostí.

Uvažujme nasledovne:

Nech *L1* je počet dní, ktoré uplynuli medzi prvou udalosťou a začiatkom roka *Y1*; *L2* počet dní medzi druhou udalosťou a začiatkom roka *Y2*.

Jednoducho zistíme rozdiel *Y2*-*Y1*, ktorý udáva počet rokov.

Pri riešení musíme uvažovať, že počet dní v mesiacoch je rôzny. V súvislosti s tým je potrebné vytvoriť pole *M*[*1..12*], ktoré bude obsahovať počty dní v jednotlivých mesiacoch. Napr. *M*[*3*]=*31* znamená, že v treťom mesiaci – marci – je 31 dní.

Ďalej musíme uvažovať, či sú roky *Y1* alebo *Y2* prestupné. Môžeme to urobiť nasledovne: Nájdeme zvyšok po celočíselnom delení *Y1* štyrmi. Ak sa rovná nule, je *Y1* prestupný rok a teda február má 29 dní. Podobne je to s *Y2*.

Teraz už môžeme vypočítať *L1*. K tomu potrebujeme sčítať počty dní v mesiacoch od prvého po *M1* a k výsledku pripočítať *D1*. Podobne je to aj s *L2*.

Zistenie počtu dní medzi dvomi udalosťami S, ktoré oddeľujú začiatok *Y1* od začiatku *Y2*, sa vykoná v dvoch krokoch: najskôr sa zistí, koľko prestupných rokov sa nachádza medzi *Y1* a *Y2*. K tomu je potrebné celočíselne deliť rozdiel *Y2*-*Y1*. Zistenú hodnotu *B* si zapamätáme. Potom

[1] Gregoriánsky kalendár bol zavedený pápežom Gregorom XIII. r. 1582. Tým, že bolo vypustených 10 dní (5. až 14. 10. 1582), bol odstránený rozdiel medzi juliánskym kalendárom (zavedeným J. G. Caesarom) a skutočným striedaním ročných období. Prestupný zostal len rok storočia deliteľný 400. Priemerný rok v tomto kalendári trvá 365,2425 dňa, tropický rok 365,2422 dňa, odchýlka je iba 22 sekúnd.

S = (Y2 - Y1) . 365 + B
Hľadaný počet dní, ktoré uplynuli medzi dvomi udalosťami, teda zistíme
P = S + L2 - L1
Tu je program v pascale:

```
Program POCET_DNI_MEDZI_DVOMI_DATUMAMI;
const m: array [1..12] of integer = (31, 28, 31, 30, 31, 30,
                                     31, 31, 30, 31, 30, 31);
var Y1,Y2,M1,M2,D1,D2,c1,c2,L1,L2,B,S,P,i: integer;

Begin
 Write ('Zadaj prvy datum v tvare den, mesiac, rok');
 ReadLn(D1, M1, Y1);
 Write ('Zadaj druhy datum v tvare den, mesiac, rok');
 ReadLn(D1, M1, Y1);
 c1:=0;
 for i:=1 to M1-1 do c1:=c1+m[i];
 if (Y1 mod 4 = 0) and (M1>2) then c1:=c1+1;
 L1:=c1+D1;
 c2:=0;
 for i:=1 to M2-1 do c2:=c2+m[i];
 if (Y2 mod 4 = 0) and (M2>2) then c2:=c2+1;
 L2:=c2+D2;
 B:=(Y2-Y1) div 4;
 P:=S+L2-L1;
 WriteLn('Pocet dni medzi tymito datumami = ', P);
 ReadLn;
End.
```

V programe je jedna z možných foriem ako umiestniť do poľa konštantné hodnoty. (Iný prípad bude uvedený v nasledujúcom programe.) Inak je program iba prepisom vyššie naznačeného postupu. Pozor aj na to, že počet dní od začiatku roku obsahuje iba súčet počtov dní už ukončených mesiacov (1.,..M1-1, resp. M2-1) a k nim pripočítaný príslušný dátum.

Vytlačme si kalendár

Zostavme program, ktorý vytlačí kalendár na daný mesiac v roku. Pritom predpokladáme, že poznáme, ktorým dňom v týždni tento mesiac začína. (Ak nie, pozri problém o zistení dňa v týždni pripadajúceho na daný dátum.)

Najskôr si musíme uvedomiť, čo má byť vstupom, ďalej si prípadne nakresliť, ako má vyzerať výstup programu.

Vstup: Mali by sme zadať textový údaj (reťazec znakov) s názvom mesiaca a poradové číslo dňa v týždni, ktorý pripadá na 1. deň daného mesiaca. U nás je zvykom, že prvým dňom v týždni je pondelok (nie je to tak všade, napr. v Anglicku je 1. dňom týždňa nedeľa). Teda vstupom pre určenie dňa v týždni bude:
1 – pondelok, 2 – utorok, 3 – streda, 4 – štvrtok, 5 – piatok, 6 – sobota, 7 – nedeľa.

Výstup: Napr. pre mesiac november 1997 by mal výsledok na obrazovke vyzerať takto:

November 1997

Po	Ut	Str	Stv	Pia	So	Ne
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16

... ...

Potrebujeme vhodne zobraziť to, čo je stále rovnaké, t. j. názov mesiaca a označenie dní v týždni; potom začať na správnom mieste číslom 1 a pokračovať, kým nebude naplnený príslušný počet dní v mesiaci. Tie naše mesiace nie sú rozvrhnuté najšťastnejšie, majú rôzne a nepravidelné počty dní. Preto by bolo asi nateraz lepšie, keby užívateľ programu zadal na vstupe aj počet dní v príslušnom mesiaci.

Program riešiaci našu úlohu môže mať tvar:

```
Program VYPIS_KALENDÁRA_NA_MESIAC;
var   MESIAC : string;
      i, p, D, PD    : integer;

Begin
 Write('Zadaj meno mesiaca,ktorym dnom zacina,kolko ma dni:');
 ReadLn (MESIAC, D, PD);
 WriteLn; WriteLn;
 WriteLn (MESIAC: 32);
 WriteLn ('Po':7, 'Ut':7, 'Str':7, 'Stv':7, 'Pia':7, 'So':7, 'Ne':7);
 for i:= 1 to D-1 do Write(' ': 7);
        {medzery a posun v riadku, kým nebude 1. deň na správnom mieste}
 p:= D; {premenná p pomáha zistiť, kedy bude v riadku 7 údajov}
 for i:=1 to PD do   {pre daný počet dní v mesiaci PD}
  begin
   Write(i:7);{výpis dňa na príslušnú o 7 posunutú pozíciu v riadku}
   if p mod 7 = 0 then WriteLn;
            {pomocná premenná p pomáha pri odriadkovaní po "nedeli"}
   p:=p+1;
  end;
 WriteLn; ReadLn;
End.
```

1. Program je dostatočne popísaný poznámkami. Skúste overiť, či je dobrý.

2. Doplňte program o primeranú grafickú úpravu tak, aby sa kalendár zobrazil približne v strede obrazovky.

Koľký deň v roku?

Ak začne človek rozmýšľať o počasí alebo o podobných problémoch, zrejme sa nudí alebo potrebuje niekoho iného zamestnať. Preto asi vznikla aj úloha zistiť, koľký deň v roku je daný dátum.

Mohli by sme použiť predchádzajúce riešenie s nepatrnými úpravami. My si však ukážme iný spôsob (aj keď nie veľmi odlišný) vloženia príslušných hodnôt do poľa a nasledovného zistenia počtu dní od začiatku roku:

```
Program KOLKY_DEN_V_ROKU;
var   d: 1..31; m: 1..12; rok: 1901..2000;
      kolko, i : integer;

Begin
 ReadLn(d, m, r);
 a[1]:=31; a[2]:=28; a[3]:=31; a[4]:=30; a[5]:=31; a[6]:=30;
 a[7]:=31; a[8]:=31; a[9]:=30; a[10]:=31; a[11]:=30; a[12]:=31;
 if rok mod 4 = 0 then a[2]:=29;
 kolko:= 0;
 for i:= 1 to m-1 do kolko:= kolko + a[i];
 kolko:= kolko + d;
 WriteLn ('Den ', d, '. ', m, '. je ', kolko, '. v roku ', r);
 ReadLn;
End.
```

Komentár k programu asi ani nie je veľmi potrebný.

Ktorý deň v týždni?

Na aký deň v týždni pondelok, utorok... nedeľa pripadne 1.1.1999? Ďalšia z úloh, ktoré môžu vymýšľať iba tí, ktorí nemajú čo robiť alebo sú učiteľmi. V ktorý deň v týždni ste sa narodili? To už by mohlo byť zaujímavejšie. Skúsme túto úlohu riešiť.

Algoritmov riešenia tohoto problému je veľa. My si ukážme trochu netradičný postup. Kto bol jeho autorom nie je známe. Vieme iba to, že patrí k najstarším a používa takmer kúzelnícke výpočty, ktoré udivujú všetkých, ktorí sa zaujímajú o rýchle zistenie dňa v týždni podľa dátumu. Predovšetkým je potrebné mať k dispozícii pomocnú tabuľku, v ktorej je každému mesiaci v roku pridelené isté číslo. Tabuľka má tvar:

január	február	marec	apríl	máj	jún	júl	august	september	október	november	december
1	4	4	0	2	5	0	3	6	1	4	6

Do počítača sa vkladá dátum v tvare deň *D*, mesiac *M*, rok *Y*.

Algoritmus pozostáva zo štyroch krokov:

1. Odčlenia sa dve posledné číslice roku *Y* a dostaneme dvojmiestne číslo *Y1*. Toto číslo sa delí celočíselne 12. Podiel označme *T1*, zvyšok po delení *R1*. Potom sa zvyšok *R1* delí 4 a uvažuje sa celá časť *R2*. Potom sa zistí súčet troch sčítancov: *R1+R2+T1*. Tento je potrebné celočíselne deliť 7 a získať zvyšok *R3*.
2. K *R3* sa pripočíta "kľúčové číslo mesiaca" – podľa tabuľky. Tento súčet sa delí celočíselne 7. Zvyšok po delení označíme *R4*.
3. K *R4* sa pripočíta deň *D* a získame *S2*. *S2* sa delí celočíselne 7 a berie sa zvyšok *R5*. Podľa hodnoty *R5* sa určuje deň v týždni takto:

	R5=0	R5=1	R5=2	R5=3	R5=4	R5=5	R5=6
	sobota	nedeľa	pondelok	utorok	streda	štvrtok	piatok

4. Ak je rok *Y* prestupný a mesiac je buď január alebo február, potom musíme od *R5* odčítať 1 a ako výsledok uvažovať získané číslo. Tento spôsob určenia dňa v týždni platí pre 20. storočie (s korekciami uvedenými pomocou premennej *dif* v programe aj pre 19. a 21. storočie. Overte!)

Príklad:

	dátum: 28-01-1953	Y=53, M=1, D=28
	53/12=4, zvyšok 5	T1=4, R1=5
	5/4=1, zvyšok 1	R2=1
	S1=T1+R1+R2=4+5+1=10	S1=10
	S1/7=3, zvyšok 3	R3=3
	Kľúčové číslo V januára je 1 (R3+V)/7=(3+1)/7=0 a zvyšok 4	 R4=4
	S2=R4+D=4+28=32 S2/7=32/7= 4, zvyšok 4	R5=4 streda

Mesiac je síce január, ale rok 1953 nebol prestupný, preto nie je potrebné robiť korekciu.

Skúste si takýmto spôsobom overiť niektorý vám známy dátum.

Výsledný program, v ktorom sa vo veľkej miere používajú celočíselné delenie (*div*) a zvyšok po celočíselnom delení (*mod*), môže mať tvar:

```
Program DEN_V_TYZDNI;
const key:array[1..2]of integer = (1,4,4,0,2,5,0,3,6,1,4, 6);
     day:array[0..6]of string = ('sobota', 'nedela', 'pondelok',
                    'utorok', 'streda', 'stvrtok', 'piatok');
var   Y, M, D, dif, y1, r1, r2, r3, r4, r5, t1, s1:integer;
      prestupny: boolean;

Begin
 Write ('Zadaj den, mesiac, rok '); ReadLn(D, M, Y);
 dif:=y div 100;
 if dif = 18 then dif:=2
             else if dif = 20 then dif:=-1
                              else dif:=0;
 y1:=Y div 100;
 t1:=y1 div 12;
 r1:=y1 mod 12;
 r2:=r+ div 4;
 s1:=t1+r1+r2;
 r3:=s1 mod 7;
 r4:=(r3+key[M]) mod 7;
 r5:=(r4+D) mod 7;
 r5:=r5+dif;
 prestupny:=(Y mod 100<>0) and (Y mod 4=0) or (Y mod 400=0);
 if (m=1) or (m=2) and prestupny then r5:=r5-1;
 WriteLn(' Dany datum je ', day[r5] ', r5);
 ReadLn;
End.
```

Premenná *dif* je použitá na prípadné diferencie storočí. Logická premenná *prestupny* dáva hodnotu *true*, ak je zadaný rok prestupný, hodnotu *false*, ak nie je prestupný.

Zistite, na aký deň v týždni pripadne 1. január 2001 – oficiálny začiatok 21. storočia.

V jednej múdrej knihe som našiel aj iný predpis s názvom "Večný kalendár" (má ho vraj na svedomí už spomínaný F. Gauss):

Návod na rýchle určenie dňa v týždni, na ktorý pripadá istý dátum (deň, mesiac, rok):

Ak dátum leží medzi rokmi 1582 (gregoriánsky kalendár, pozri poznámku) po 4902 (Prečo?, Nový kalendár alebo ... ? , pozn. autora), potom číslo dňa v týždni (nedeľa=0, pondelok=1, utorok=2 ... sobota=6) je rovné zvyšku po celočíselnom delení číslom 7 výrazu

$[2.6m-0.2]+d+y+[y/4]+[c/4]-2c$

kde *d*–číslo dňa v mesiaci, m–číslo mesiaca (začína sa od marca: marec=1, apríl=2, máj=3,... , december=10, január a február majú čísla 11 a 12 predchádzajúceho roka); *y*–číslo tvorené poslednými dvomi číslicami roka, *c*–číslo tvorené prvými dvomi číslicami roku (15..49), [*x*]–celá časť z *x*, odrezanie desatinnej časti.

Napíšte program, ktorý bude zisťovať na základe uvedeného dňa, mesiaca a roku, o ktorý deň v týždni ide.

Zostavte program, ktorý bude pre zadaný mesiac a rok vypisovať kalendár na tento mesiac v prehľadnom tvare na obrazovke.

Nestarnúci kalendár

V zásuvke písacieho stola sa mi pred koncom (a po začiatku) každého kalendárneho roka zhromažďujú "drobné úplatky" vo forme kalendárov alebo diárov rôznych firiem. Sú medzi nimi niektoré, ktoré patria skôr už medzi starožitnosti. Čo s nimi? Vyhodiť niečo je človeku ľúto – ako keby vyhadzoval kus seba (hlavne keď je to zadarmo). Kedy bude zodpovedať to, že rok začína tým istým dňom v týždni ako v kalendári (diári)? Nedalo by sa na toto určenie použiť počítač?

Zostavte program, ktorý bude pre zadaný rok zisťovať, ktorý najbližšie vyšší rok bude začínať tým istým dňom v týždni.

Skúsme najskôr uviesť program, potom ho komentovať:

```
Program STARY_NOVY_KALENDAR;
uses Crt;
var YEAR, Y1, i, N, s : integer;

Function Prestupny(rok: integer): integer;

begin
 if (rok mod 100<>0) and (rok mod 4=0) or (rok mod 400=0)
                     then Prestupny:=2
                     else Prestupny:=1;
end;

Begin
 Write ('Na ktory rok mas kalendar? '); ReadLn(YEAR);
 Write ('Kolko opakovani? '); ReadLn(N);
 for i:=1 to N do
  begin
   Y1:=YEAR; s:=0;
   repeat
    Y1:=Y1+1; s:=s+Prestupny(Y1);
   until (s mod 7 = 0) and (Prestupny(YEAR)=Prestupny(Y1));
  WriteLn (i, '. rok = ', Y1);
 end;
 ReadLn;
End.
```

Ak program "spojazdníte" na počítači, môžete si preveriť, že kalendár na rok 1873 vyhovuje aj pre roky 1997, 2003, 2014.

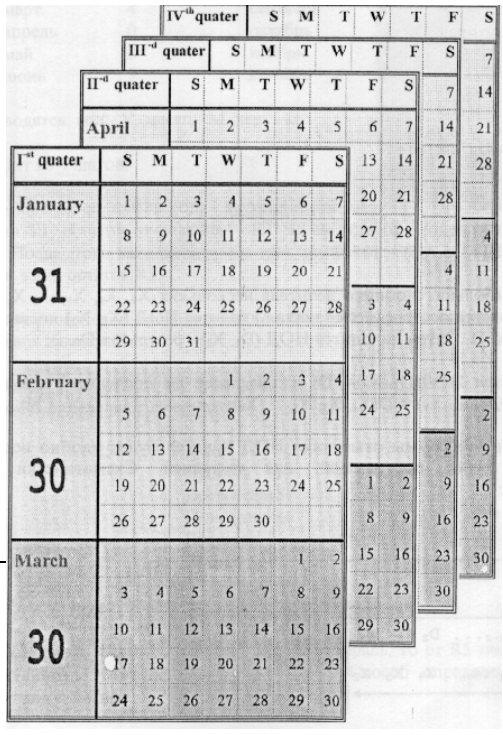
V programe sú použité tieto premenné:

YEAR – premenná pre uchovanie hodnoty roku, ktorého máme kalendár,

Y1 – uchováva rok, pre ktorý zisťujeme, či zodpovedá podmienkam,

s – hromažďuje sa "posúvanie" pre deň týždňa 1. januára roku *Y1*,

i, n – pomocné premenné.



Základnou myšlienkou algoritmu je, že hľadáme rok, v ktorom deň týždňa pre 1. január je ten istý ako u zadaného roku *YEAR*. Ak taký rok nájdeme, musí sa preveriť ešte ďalšia podmienka, či sú oba roky súčasne prestupné alebo neprestupné.

Na určenie roka, v ktorom bude 1. január pripadať na rovnaký deň v týždni, používame skutočnosť, že v prípade neprestupného roka sa deň v týždni pre 1. január "zvyšuje" o jeden deň. Napr. 1. 1. 1995 bola nedeľa, 1. 1. 1996 pondelok. Za prestupný rok deň v týždni "preskakuje" o dva dni. Preto 1. 1. 1997 bola streda.

V premennej *s* sa teda "napĺňa" určenie 1. januára preverovaného roku. Keď premenná s obsahuje hodnotu deliteľnú 7, dosiahli sme posun o 1 týždeň a 1. januára pripadá na ten istý deň v týždni, ako má rok *YEAR*. V tomto prípade ešte musíme preveriť, či oba roky *YEAR* a *Y1* sú súčasne prestupné alebo neprestupné. Ak áno, našli sme rok spĺňajúci podmienky. Toto sa opakuje N-krát.

Pre overenie, či je rok prestupný, je v programe použitá funkcia *Prestupny*. Jej výsledkom je hodnota 2, ak je rok prestupný, resp. hodnota 1, ak je neprestupný.

Je zrejmé, že podobným spôsobom môžeme "ísť" aj do minulosti. Skúste upraviť program tak, aby zisťoval daný počet rokov v minulosti, ktoré mali rovnaký kalendár ako zadaný rok.

Dá sa veriť tvrdeniu, že stačí mať 14 kalendárov, aby sme mali vždy na každý rok vhodný? Asi áno. Stačí, ak uvážime, že rok môže začínať iba 7 možnými dňami v týždni – pondelok ... nedeľa. To je 7. Okrem toho môže každý rok byť buď prestupný, alebo neprestupný, teda 7x2 = 14.

Zdalo by sa, že história kalendára je vyriešená – už dávno bol zostavený ideálny kalendár. Ale nie je to pravda. Určite viete, že rôzne národy a komunity používajú rôzne kalendáre. Všeobecne je v medzinárodnom styku používaný "náš" kalendár, ale je ideálny?

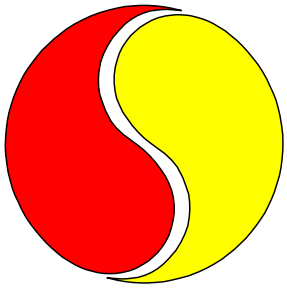
Toto vŕtalo v hlavách expertov OSN (Organizácia spojených národov) a UNESCO (Organizácia pre vedu, kultúru a vzdelávanie). A tak sa stalo, že bol navrhnutý veľmi zaujímavý, aj keď zrejme utopistický projekt celosvetového kalendára. (Že zrejme vďaka konzervativizmu a tradícii neprešiel, asi viete.) Skúsme si ho popísať, pretože tiež (podľa mňa) patrí ku krásnym ukážkam toho, ako človek môže rozumom nabúrať už tradičné metódy prístupu k riešeniu problémov.

Podľa tohto projektu sa rok delí na 4 kvartály. V každom kvartáli po 3 mesiace, resp. 13 týždňov. Prvý z troch mesiacov kvartálu má 31 dní, zvyšné dva mesiace po 30 dní. Takto získame 364 dní. Prvý mesiac každého kvartálu sa začína nedeľou, druhý mesiac každého kvartálu stredou a tretí mesiac piatkom.

Posledný 365. deň roka navrhli experti nazvať "biely deň" (u nás tzv. modrý pondelok) a nedávať mu ani číslo ani deň v týždni. Vsunul by sa medzi 30. decembra starého a 1. januára nového roku, bol by voľným dňom a slúžil by (ako náš Silvester) iba na bilancovanie starého a oslavy nového roka. Deň, ktorý sa dopĺňa v prestupnom roku (29. február), navrhli uvažovať ako 31. jún a bol by určený pre otvorenie olympijských hier – zase ako voľný a bez určenie dňa v týždni. Pekná idea, však? Určite by odpadlo veľa problémov, ktoré sme doteraz v súvislosti s kalendárom spomínali. (Ale pribudli by určite nové, ľudská fantázia je nevyčerpateľná.)

Napíšte program, ktorý vytlačí (zobrazí) kalendár na celý zadaný rok. Program by mal fungovať pre ľubovoľný rok od 1600 po 2100. Zvoľte vhodný a prehľadný tvar výsledného zobrazenia, aby bolo zrejmé, ako je rozdelený rok na mesiace, koľko dní je v každom mesiaci a na aký deň v týždni pripadá ktorý dátum.

7.25 Vieš, čo ťa čaká?



Už sme tu spomínali hviezdy, teraz si však vezmime na pomoc tie skutočné, ktoré svietili pred tým, ako sa prvýkrát rozsvietili obrazovky počítačov, a budú svietiť, aj keď obrazovky zhasnú. Osud máme vraj ukrytý vo hviezdach, stačí iba vedieť ho odtiaľ správne prečítať. Astrológiou sa zaoberať nebudeme, horoskopy sa asi tiež neprogramujú jednoducho. Skúsme jednoduchšiu úlohu. Každý už asi počul o kondiciograme. Dodáva sa mu viac či menej vedeckosti, no všetko závisí od toho, či niečomu chcete alebo nechcete veriť. Popíšme úlohu podrobnejšie, samozrejme, s istými zjednodušeniami tak, aby sme ju mohli realizovať na počítači v tvare programu.

Tvrdí sa, že ľudský organizmus má svoje cykly, v ktorých dochádza k rovnakým situáciám. Nazhromaždená energia sa spotrebuje, potom musí nastúpiť obdobie regenerácie, medzi tým existujú stavy, ktoré nie sú definovateľné. Fyzický cyklus sa vraj opakuje po 23, citový po 28 a inteligenčný po 33 dňoch. V prvej polovici cyklu máme v danej oblasti kladné výsledky, vychádza, do čoho sa pustíme. V polovici dochádza k zlomu, tzv. kritickému dňu, počas ktorého sa nedá nič vopred určiť a môže dôjsť k preceneniu vlastných schopností. Druhá polovica cyklu obsahuje rad dní, kedy sa nám nedarí a len postupne "dobíjame baterky". Prechod do nového cyklu tvorí tzv. polokritický deň, ktorý má podobnú charakteristiku ako kritický. Všetky cykly sa začínajú polokritickým dňom, ktorý je dňom narodenia každého.

Skúste zostaviť program, ktorý pre daný dátum narodenia a deň, na ktorý chceme kondiciogram, určí, aké máme dispozície v jednotlivých oblastiach – fyzickej, citovej a inteligenčnej.

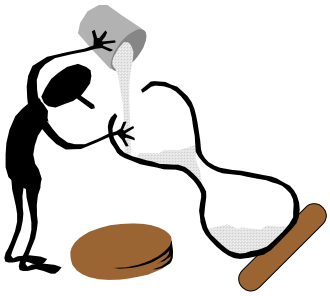
Máme zadané dva dátumy, prvý, dátum narodenia, je skorší. K tomu, aby sme zistili kondiciogram na daný deň potrebujeme vlastne zistiť, koľko dní uplynulo medzi týmito dátumami. A to tu už bolo! Môžeme teda využiť program POCET_DNI_MEDZI_DVOMI_DATUMAMI. Jeho výsledkom bola hodnota *P* – počet dní.

Zostáva nám iba vyriešiť zaradenie dňa do príslušnej skupiny (kladné, záporné dni, kritický alebo polokritický deň) a výsledok pre jednotlivé cykly oznámiť. K tomu si určime tieto skupiny:

Fyzický cyklus 23 dní:
1. deň – polokritický – zvyšok po celočíselnom delení počtu dní 23 je 0
2.–11. deň – kladné – zvyšok po celočíselnom delení počtu dní 23 je 1–10
12. deň – kritický deň– zvyšok po celočíselnom delení počtu dní 23 je 11
13.–23. deň – záporné – zvyšok po celočíselnom delení počtu dní 23 je 12–22

Citový cyklus 28 dní:
1. deň – polokritický – zvyšok po celočíselnom delení počtu dní 28 je 0
2.–14. deň – kladné – zvyšok po celočíselnom delení počtu dní 28 je 1–13
15. deň – kritický deň – zvyšok po celočíselnom delení počtu dní 28 je 14
16.–28. deň – záporné – zvyšok po celočíselnom delení počtu dní 28 je 15–27

Inteligenčný cyklus 33 dní:
1. deň – polokritický – zvyšok po celočíselnom delení počtu dní 33 je 0
2.–16. deň – kladné – zvyšok po celočíselnom delení počtu dní 33 je 1–15
17. deň – kritický deň – zvyšok po celočíselnom delení počtu dní 33 je 16
18.–33. deň – záporné – zvyšok po celočíselnom delení počtu dní 33 je 17–32



(Poznámka: Určenie presnej polovice cyklov je otázne.)

Hlavná časť výpočtu kondiciogramu je teda hotová, stačí ju iba prepísať do programu a vhodne zabezpečiť výpis pre užívateľa. Môžete napr. použiť výpis v tvare “+”, “-”, “P”, “K” a vhodný komentár. Trochu zložitejšie je iba rozvetvenie jednotlivých možností.

1. Podľa uvedeného rozboru zostavte program na určenie kondiciogramu.

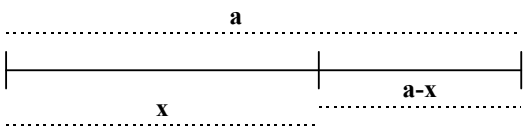
2. Skúste určiť, po koľkých dňoch (rokoch) budeme mať rovnaký kondiciogram. (Napr. kedy od narodenia znovu nastane situácia, že vo všetkých cykloch bude polokritický deň.)

3. Zostavte program, ktorý bude vypočítavať kondiciogram napr. na týždeň alebo mesiac od daného dátumu.

4. Zostavte program, v ktorom sa kondiciogram bude určovať “percentuálne” alebo nejakou inou číselnou hodnotou. Návod: Cykly si môžeme predstaviť ako sinusoidy s rôznou periódou. Alebo sa podľa čísla dňa v cykle dá priradiť mu istú kladnú, nulovú alebo zápornú hodnotu podľa vášho uváženia.

7.26 Zlatý rez

Zlato vždy hralo u ľudí významnú úlohu. Preto sa nám zachovalo veľa názvov, v ktorých sa zlato objavuje v rôznej podobe, napr. “zlatá žila”, “zlaté srdce”, “zlato moje”... Medzi takéto názvy patrí aj pojem zlatého rezu.



Už starovekí Gréci poznali zákonitosť, nazývanú neskôr zlatý rez. Je to pomer rozdelenia úsečky na 2 časti tak, aby dĺžka úsečky *a* k úsečke *x* bola v rovnakom pomere ako dĺžka *x* k zvyšku úsečky, teda *a-x*.

Matematicky vyjadrené:

φ= 1/2 * (1 + sqrt(5))

kde *sqrt* je označenie pre druhú odmocninu, v tomto prípade čísla 5.

φ ≅ 1, 61803 39887 49894 84820 45868 .....

φ je tiež jediné číslo, ktoré má tú vlastnosť, že 1/φ je rovné desatinnej časti φ, teda

1/φ ≅ 0, 61803 39887 49894 84820 45868 .....

Euklides ho nazval “vzťahom krajného a stredného”, v období renesancie tento pomer nazvali “božskou proporciou” alebo “zlatým rezom”. V umení sa zlatý rez považuje za esteticky najkrajšiu proporciu, napr. najznámejší starogrécky sochár Feidiás (5. stor. pr. n. l., autor jedného z tzv. siedmich divov sveta – sochy boha Dia v Olympii; podpisoval sa φ) ho využíval pri tvorbe svojich sôch.

Skúste vypočítať hodnotu φ a 1/φ na počítači alebo na kalkulačke a porovnať hodnoty desatinných častí. Zhodujú sa? Zrejme presnejšie už hodnotu podľa takéhoto predpisu nedostaneme. (Iba ak by sme dokázali získať presnejšiu hodnotu odmocniny piatich a niečo vymysleli na delenie s veľkou presnosťou – pozri ďalej.)

My sa však pokúsime využiť ďalší “zázrak”, ktorý nám pripravili matematici. Zistili totiž, že podiel dvoch po sebe nasledujúcich Fibonacciho čísel sa viac a viac blíži k hodnote φ.

Platí: `5/3 = 1.6666..., 34/11 = 1.619...,  89/55 = 1.618...`

Fibonacciho čísla dokážeme bez problémov vypočítať (pozri problém Králici ála Italiano). Ak by sme mali k dispozícii aj algoritmus na výpočet podielu dvoch čísel s veľkou presnosťou, možno by sa nám podarilo podstatne sa priblížiť hodnote φ. Skúsme to.

Delenie s veľkou presnosťou

Zostavme program, ktorý pre zadané dve čísla A a B zistí ich podiel s presnosťou na N desatinných miest.

Skúsme si spomenúť na to, ako sme kedysi delili dve čísla pomocou písomného delenia:

Nech A = 21, B = 23. Zápis postupu vyzerá takto:

21: 23 = *0, 9130...*
-0
210: 23 = *9*
-207
30: 23 = *1*

-23
70: 23 = *3*
-69
10: 23 = *0* atď.

Algoritmus nemusíme zvlášť rozpisovať. Stačí, ak si uvedomíme, že výsledok môžeme alebo vytlačiť priamo na obrazovku – vtedy si ho nepamätáme, alebo uložiť "číslicu po číslici" do poľa. Tu použime uloženie do poľa *C*, kde do prvku *C*[*0*] uložíme hodnotu celej časti výsledku (preto sú úvodzovky okolo číslic) a do prvkov *C*[*1*]... C[N] postupne jednotlivé číslice desatinnej časti výsledku. Pretože hodnota *A* sa počas výpočtu mení, odložíme si ju do pomocnej premennej *pomA*.

```
Program VELKE_DELENIE;
var   A, B, N, i, pomA : longint;
      C : array [0..100] of integer;

Begin
 Write (‘Zadaj delenca, delitela a pocet desatinnych miest podielu ‘);
 ReadLn (A, B, N);
 C[0]:= A div B; pomA:= A;
 A:= A mod B;
 for i:=1 to N do
  begin
   C[i]:= (A*10) div B;
   A:= A mod B;
  end;
 Write (pomA, ‘: ‘, B, ‘ = ‘, C[0], ‘, ‘);
 for i:=1 to N do Write (C[i]);
 WriteLn; ReadLn;
End.
```

1. Overte si správnosť programu vytvorením tabuľky sledovania výpočtu pre niekoľko prípadov.

2. Ak je v istom momente hodnota A rovná 0, nemá už zmysel vo výpočte ďalších číslic pokračovať. Upravte program, aby v takomto prípade skončil.

3. Niektoré výsledky delenia dvoch prirodzených čísel dávajú racionálne čísla, t. j. také, u ktorých sa od istej číslice počnúc opakuje nejaká periodická zostava číslic. Napr. 2 : 3 = 0,~~6~~, kde ~~6~~ značí periódu, 7 : 11=0,~~63~~; 11 : 13=0,~~846153~~, u niektorých podielov sa perióda vyskytuje až po niekoľkých neperiodických čísliciach, napr. 19 : 36=0,52~~7~~, a pod. Pokúste sa zostaviť program, ktorý v momente, keď zistí periódu, oznámi ju a ukončí výpočet.

Naspäť k zlatému rezu

Prakticky nám už nič nebráni vytvoriť program, ktorý overí predpoklad, že podiel dvoch po sebe nasledujúcich Fibonacciho čísel *F[i] : F[i-1]* sa postupne blíži (konverguje) k hodnote zlatého rezu φ. Stačí len vhodne skombinovať už uvedené programy na zistenie postupnosti Fibonacciho čísel, delenie s veľkou presnosťou a porovnanie výsledku.

1. Zostavte program, ktorý bude postupne brať N dvojíc Fibonacciho čísel a ich podiel porovnávať s hodnotou zlatého rezu. Program by mal udávať, na koľkých desatinných miestach sa výsledok zhoduje s hodnotou φ.

2. Zostavte program tak, aby vykonával výpočet dovtedy, kým sa porovnávané hodnoty nebudú zhodovať v prvých M číslicach.

ČO NA ZÁVER?

Ak ste sa nielen listovaním dopracovali až sem, gratulujem a teším sa. Pôvodne mala byť táto knižka skôr úvodom do pascalu nahrádzajúcim beznádejne vypredané tituly klasického typu s cenou prístupnou aj študentovi. Či sa to podarilo, musíte posúdiť sami. Často sa totiž stáva, že keď dlho pracujete s nejakým "nástrojom" a získate patričnú rutinu, nedokážete pochopiť, že to niekto iný nevie.

Pokúsil som sa ukázať riešenie problémov na počítačoch, ako ich riešim ja. Snáď vám to pomôže, aj keď nikto nemá patent na rozum. Ale utešuje ma, že - ako všetci dobre poznáme z každodenného života – niekedy aj zlý príklad je príkladom. Všetci sa učíme (najlepšie) na vlastných chybách, tí múdrejší (niekedy-zriedka) aj na chybách cudzích. Určite vás napadnú oveľa krajšie a efektívnejšie riešenia riešených i zadaných problémov, na ktoré budete primerane a právom hrdí. Ak sa o to pokúsite a podarí sa vám to, potom

for i:=1 to MaxInt do TesimSaSVami.

Možno viacerí budete nespokojní – nie sú tu všetky systémové premenné, nepoužívajú sa tajné programátorské finty. Nepodarilo sa vynechať matematiku a dokonca sa sem poriadne "zamontovala" aj história a "primitívne" hry, ktoré nemajú grafiku, o virtuálnej realite ani nehovoriac. Napriek tomu si však myslím, že to až tak nebolelo, a má to svoje dôvody. Počul som už viackrát, že programátor nepotrebuje nič iné vedieť ako dobre poznať počítač a programovací jazyk. Ako však, dúfam dostatočne, ukazujú príklady, v ktorých sa pokúšame realizovať na počítači reálne situácie, väčšinou musíme vedieť podstatne zjednodušiť úlohu a vytvoriť model jej riešenia. A keď nie je matematický, tak je aspoň založený na presných pravidlách, aké nás učí hľadať práve matematika (a v spolupráci s ňou informatika). A prečo história? Od koho sa možeme najlepšie naučiť myslieť – riešiť problémy, ako od tých, ktorých mená sú vytesané veľkými písmenami na strmých schodoch cesty ľudstva k poznaniu?

Čo to je programovanie? Technika alebo spôsob myslenia, zábava či umenie? Na tieto otázky si musíte s primeranou znalosťou veci opovedať sami. Pri rozhodovaní vám nepomôže naučiť sa naspamäť všetky doteraz známe algoritmy a programovacie jazyky sveta. Chce to "iba" spojiť realitu s fantáziou a vo vhodnom pomere ich namixovať ako koktejl na počítači.

Na zdravie!

OBSAH

ÚVOD ... 1
1 PROBLÉM A RIEŠENIE PROBLÉMU (POSTUP – ALGORITMUS – PROGRAM) ... 3
Ako začať? ... 3
1.1 Terminológia ... 3
1.2 Mám problém, čo s ním? ... 5
1.3 Algoritmus ... 5
Algoritmizácia ... 10
Algoritmus a program ... 10
Programovanie ... 10
2. JAZYK, ALGORITMICKÝ JAZYK, PROGRAMOVACÍ JAZYK ... 12
3. ZÁKLADNÉ ALGORITMICKÉ KONŠTRUKCIE A ICH ZÁPIS POMOCOU ŠTRUKTÚROGRAMOV .. 14
3.1 Príklady algoritmov ... 16
4. RÝCHLO DO PASCALU – ZAPNITE TURBO ... 24
4.1 Na začiatok trochu histórie a teórie ... 24
4.2 Kedy už budeme programovať? ... 25
Tvar programu v pascale ... 25
4.3 Príkazy vstupu a výstupu v pascale ... 25
4.4 Prepis štruktúrogramov do pascalu ... 26
4.5 Deklarácie premenných ... 27
4.6 Prostredie Turbo pascalu ... 28
Postup pri vytváraní nového programu ... 29
Otváranie, editovanie a spúšťanie hotového programu ... 30
5 TYPY ÚDAJOV ... 32
5.1 Celé čísla ... 32
Čoho je viac – zrniek piesku alebo hviezd? ... 33
Malá násobilka ... 37
Návšteva z Bilandu ... 38
Skončí to niekedy? ... 40
Celé čísla v problémoch ... 41
5.2 Reálne čísla ... 42
5.3 Znaky ... 45
5.4 Reťazce znakov ... 47
5.5 Logický typ ... 48
5.6 Ordinálne typy údajov – interval a vymenovaný typ ... 49
5.7 Štruktúrovaný typ pole ... 50

Králici á la italiano 50
6. PODPROGRAMY 53
6.1 Procedúra bez parametrov 54
6.2 Procedúra s parametrom volaným hodnotou 55
6.3 Procedúra s parametrami volanými odkazom (referenciou) 56
6.4 Funkcia s parametrami 57
6.5 Využitie logickej funkcie 57
6.6 Deklarácia podprogramu pomocou poprednej deklarácie *forward* 58
6.7 Povedz mi, zrkadielko... 58
6.8 Niečo o tvare a spôsobe zápisu programu 60
6.9 Medzi Skyllou a Charybdou 61
7. ZBIERKA (HROMADA) PRÍKLADOV A ÚLOH 63
7.1 Pekne sa pozdrav 63
7.2 Eratostenovo sito 64
7.3 Ktoré písmeno vyhráva? 66
7.4 Svetlá veľkomesta menom unit crt 68
7.5 Chcete byť v televízii? 70
7.6 ... - - - ... (Čítaj SOS) 71
7.7 Vyčítanka 73
7.8 Hlava alebo orol? 75
7.9 Med alebo jed? 76
7.10 Strihnime si! 77
7.11 Myslím si číslo 80
7.12 Kocky sú hodené! 81
7.13 Ruská ruleta 83
7.14 Miešanie kariet 84
Náhodné pole 84
Rozdanie kariet 85
7.15 Hazard zadarmo – KENO 10 86
Vklad a voľba čísel 86
Generovanie ťahu 20 čísiel 87
Vyhodnotenie (ne-)výhry 88
7.16 Mastermind 89
7.17 Patríme k sebe, miláčik? 91
7.18 Konečne si prečítam noviny! 93
Úvod 93
Zadanie počtu príkladov 94

Nastavenie počiatočných hodnôt ... 94
Generovanie a výpis príkladu ... 94
Vstup odpovede ... 95
Vyhodnotenie odpovede ... 95
Vyhodnotenie celého skúšania ... 96
Odhlásenie ... 96
7.19 Hráte šach? ... 97
7.20 Koľko je obyvateľov? Sčítanie veľkých čísel ... 101
7.21 Máte peniaze? Sem s nimi! ... 103
Násobenie veľkých čísel ... 104
7.22 Horíííí! ... 106
7.23 Keby to Pascal vedel? ... 108
7.24 Čas prideľuje každému rovnako ... 110
Počet dní medzi dvomi dátumami ... 110
Vytlačme si kalendár ... 111
Koľký deň v roku? ... 112
Ktorý deň v týždni? ... 113
Nestarnúci kalendár ... 115
7.25 Vieš, čo ťa čaká? ... 117
7.26 Zlatý rez ... 119
Delenie s veľkou presnosťou ... 119
Naspäť k zlatému rezu ... 120
ČO NA ZÁVER? ... 121