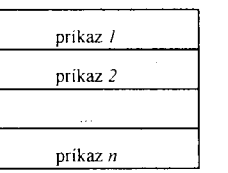
3. ZÁKLADNÉ ALGORITMICKÉ KONŠTRUKCIE A ICH ZÁPIS POMOCOU ŠTRUKTÚROGRAMOV

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.

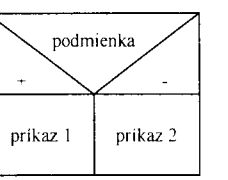
Na ukážku zápisu algoritmov v algoritmickom jazyku použijeme tzv. ***štruktúrogramy***. Nie sú (u nás) veľmi často používané, čo je škoda, pretože ich prepis do pascalu je, narozdiel od anachronicky používaných vývojových diagramov, veľmi jednoduchý.



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.



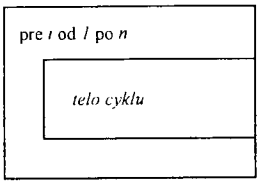
➢ Ak potrebujeme vetviť postup na viacero rôznych riešení v závislosti od podmienky, vkladáme viacero alternatív "do seba".

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.

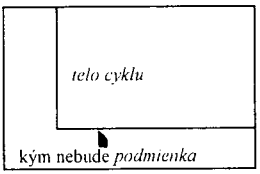


Poznámka: Veľmi často sa používa práve označenie riadiacej premennej *i* alebo *j*. Je to tiež "anachronizmus" (niečo, čo sa vžilo v minulosti a dnes sa používa zo zotrvačnosti) – prvý vyšší programovací jazyk fortran mal nedeklarované premenné (tie, ktoré neboli popísané na začiatku programu) automaticky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

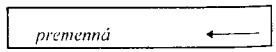
3. Cyklus s podmienkou na konci: Je prakticky opačný ako cyklus s podmienkou na začiatku - najskôr sa vykoná telo cyklu. Potom sa zisťuje splnenie podmienky cyklu. Ak je splnená, vykonávanie cyklu sa ukončí, v opačnom prípade sa riadenie procesu znovu vráti na vykonávanie tela cyklu. (Tento cyklus môžeme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



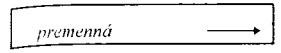
To sú dostačujúce prostriedky pre zápis ľubovoľného algoritmu – postupu. Ešte je potrebné okrem tejto riadiacej zložky algoritmického jazyka opísať **operačnú zložku**, t. j. aké elementárne činnosti (príkazy) vie procesor vykonávať.

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.

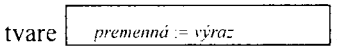


Príkaz výstupu: Umožňuje získať od procesora výsledky algoritmu alebo iné oznamy (napr. oznam o tom, že sme zadali nesprávne vstupné údaje). Preto sa pred "šípku von" zapisuje alebo názov premennej, ktorej hodnotu chceme získať, alebo text uzavretý medzi úvodzovky (napr. "nemá riešenie"). Súhrnne tieto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

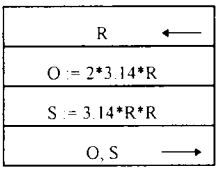
Podmienka je v programovacích jazykoch chápaná ako logický výraz, t. j. zistenie vzťahov (relácie) medzi výrazmi, prípadne zviazané logickými operáciami (*and* = "a súčasne", *or* = "alebo" a *not* = "neplatí, že").

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:

1. Vynes smeti
2. Urob si úlohy
3. Zjedz jogurt z chladničky
4. Uč sa!!!!
5. Buď dobrý (-á) (-é) !

Prídeme neskôr M+T

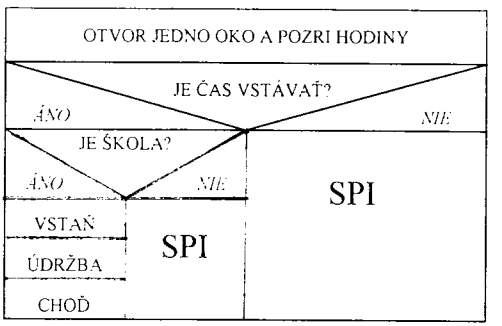
Jasné, však? A je to iba iným spôsobom zapísaná sekvencia. V štruktúrogramoch netreba dopĺňať čísla jednotlivých činností, ich poradie je dané tým, ako sú zapisované pod sebou. A do algoritmu sa nepridávajú nijaké iné činnosti ako príkazy.

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.

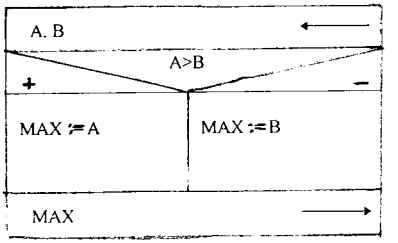
Upravte algoritmus tak, aby zohľadňoval vyššie uvedené pripomienky. Ako vyriešiť problém nedokonavého SPI?

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.

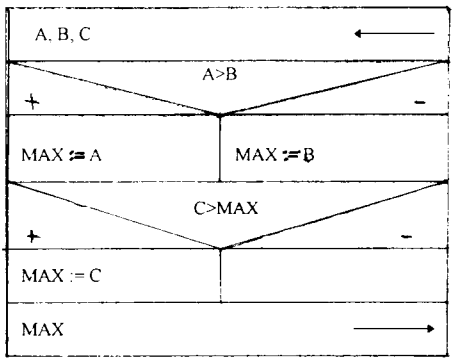
Predpokladajme, že máme tri hodnoty uložené v premenných A, B, C. Môžeme uvažovať všetky možné prípady: $A \geq B \geq C$ – potom *MAX* nadobudne hodnotu A; $B \geq A \geq C$ – potom *MAX* nadobudne hodnotu B; alebo $C \geq A \geq B$,...

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:

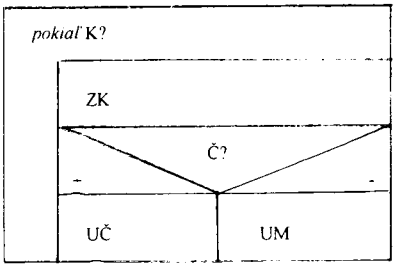
- *čítať štruktúrogramy a riadiť sa ich obsahom*
- *zobrať kocku z pásu* – *označenie ZK*
- *zistiť, či je červená* – *označenie Č?*
- *uložiť kocku do „červenej" škatule* – *označenie UČ*
- *uložiť kocku do „modrej" škatule* – *označenie UM*
- *zistiť, či je na páse kocka* – *označenie K?*

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.

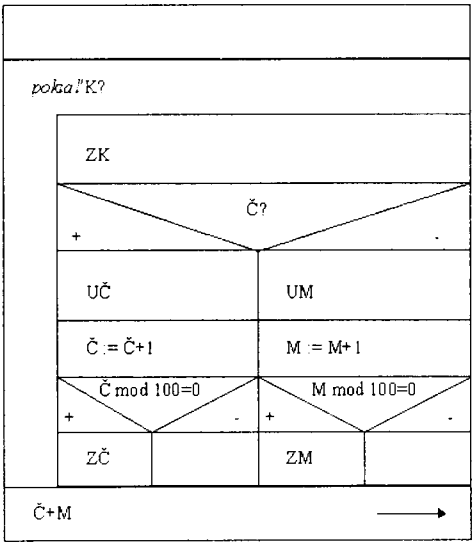
Je zrejmé, že budeme musieť rozšíriť Toborove schopnosti o ďalšie činnosti a zisťovania:

– 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.

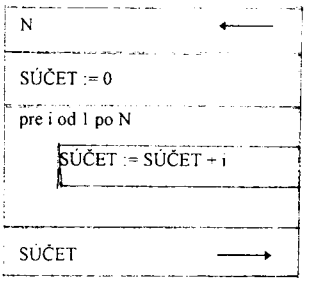
Máme pre konkrétnu hodnotu N zistiť súčet v tvare

```
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:



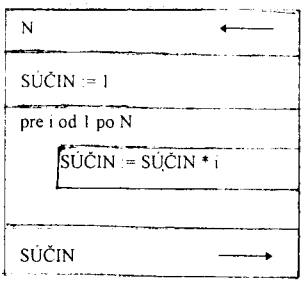
Príkaz v tele cyklu znamená toto: "K momentálnej hodnote premennej *SÚČET* pripočítaj momentálnu hodnotu *i* a výsledok ulož do premennej *SÚČET*." Takéto príkazy nám umožňujú cyklicky meniť obsahy premenných.

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...

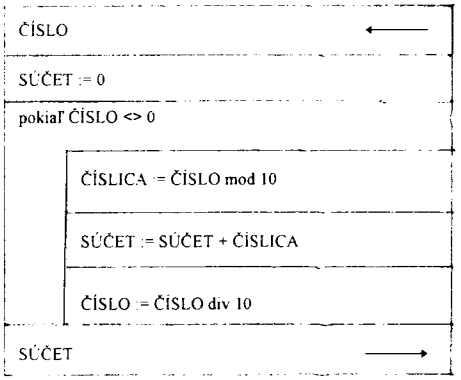
Ak je číslo veľmi dlhé, ani pre človeka nie je táto úloha úplne elementárna. Musí brať postupne číslicu po číslici a pripočítavať jej hodnotu k predchádzajúcemu súčtu. Podobným spôsobom musíme úlohy, v ktorých treba číslo spracovávať číslicu po číslici, riešiť aj na počítači. Navyše "odtrhnutie číslice" 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 nasledujúcej 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. Výber 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.