TURBO PASCAL II.

Ján Skalka

O čo ide (?)

Učebnica sa snaží o prierez celým Turbo pascalom - od práce s reťazcami, cez zavedenie typu záznam (record), používanie súborov, dynamické premenné, zásobník, front, cez binárne stromy, backtracking a rekurziu až po prácu v grafike, s tlačiarňou atď.

Je rozdelená na niekoľko častí:
V prvej sa snažíme o vytvorenie vzťahu k algoritmizácii a programovaniu vôbec. Ide v podstate o vysvetlenie niekoľkých partií z programovacieho jazyka. Väčšinou ide o úplne nové, v algoritmizácii nepoužívané, oblasti (práca s reťazcami, dynamickými premennými...).
Druhú časť tvoria programátorské techniky, ktoré prinášajú nové veci nie po syntaktickej, ale po algoritmickej stránke. Využívajú len známe veci, no v myšlienkových pochodoch sa dosť odlišujú od klasického sekvenčno - vetviaceho programovania (zásobník, front, backtracking...).
Tretia časť je venovaná grafike a grafike a grafike.
Štvrtou - poslednou - časťou sa snažíme ponúknuť nádejným budúcim programátorom svoje niekoľkoročné skúsenosti, ktoré sa väčšinou v príručkách pohromade nenachádzajú.

Celá učebnica poskytuje nadhľadový pohľad na Turbo pascal a snaží sa o jeho vysvetlenie (možno aj naučenie) netradične a v príkladoch.

1. Zoznamovanie s nástrojom

1.1 Kocúrkovský vrátnik (STRING)

V závode Kocúrkovské železiarne a spaľovne dodržiavajú zásadu, že počas jedného pracovného dňa smie to isté auto vojsť do závodu len raz. V rámci šetrenia lesov dodali na vrátnicu počítač. Povinnosťou vrátnika je zaznamenať ŠPZ (štátna poznávacia značka) každého vchádzajúceho vozidla, a ak v ten deň závod už navštívilo, nevpustiť ho.

Napíšte program, ktorý zistí, či auto možno vpustiť. (V Kocúrkove existuje len 20 vozidiel, každá značka pozostáva z 10 znakov:
4 písmená-dvojciferné číslo-dvojciferné číslo (napr. KOCO-12-50).

Jedným zo štandardných typov údajov v pascale je typ ***char***. V premennej typu ***char*** môže byť uložený ľubovoľný znak, ktorý sa nám podarí "vydolovať" z klávesnice alebo pamäti počítača. Veľkou nevýhodou tejto premennej je však skutočnosť, že môže obsahovať len jeden znak, čo je v niektorých prípadoch veľmi skľučujúce (nie sme schopní prečítať reťazec v inej podobe ako pole prvkov typu ***char***) .

Po prečítaní značky auta ako poľa znakov ju budeme porovnávať so značkami áut, ktoré už v závode boli, a v prípade, že toto auto ešte od svitania daného dňa závod nenavštívilo, ju zaradíme do zoznamu a auto vpustíme. Návrh programu v pascale môže vyzerať nasledovne:

```
Program Kocurkovo;
Type SPZ=Array[1..10] Of Char;
Var Pole:Array[1..20] Of SPZ;
     a,b:SPZ;
     Znak,Pocet_Aut,i:Integer;
     Vpustit:Boolean;
     Koniec:Char;

begin
 Pocet_Aut:=0;

 repeat
  Write('Zadaj cislo auta: ');
  Vpustit:=TRUE;    {načítanie značky}
  For i:=1 To 9 Do Read(a[i]);
  ReadLn(a[10]);

  i:=1;
  While i<=Pocet_Aut Do begin
    b:=Pole[i];     {kontrola značiek}
    Znak:=0;

    repeat
        Znak:=Znak+1;
    until (b[Znak]<>a[Znak]) or (Znak=10);
                                              {našlo sa}
    If (a[Znak]=b[Znak]) Then begin
                         WriteLn('Nevpustit !');
                         Vpustit:=FALSE;
```

```
                        i:=Pocet_Aut+1;
    end else i:=i+1
  end;

  If Vpustit Then begin
    WriteLn('Zdvihni zavoru!');          {ešte v závode nebolo}
      Pocet_Aut:=Pocet_Aut+1;
      For i:=1 To 10 Do Pole[Pocet_Aut]:=a;
  end;
  Write('Skoncit? ');
  ReadLn(Koniec);
 Until Koniec='a';
End.
```

Jednotlivé znaky značky sme zadávali v cykle, pretože počítač nie je schopný prečítať pole naraz.

Riaditeľ sa však na školení v neďalekom Káčerove dopočul, že takéto načítavanie je pomalé a neefektívne, preto okamžite starý program zavrhol.

Vysokoškolsky vzdelaný vrátnik sa po preštudovaní štvorkilogramovej kopy literatúry dozvedel, že v Turbo pascale existuje typ ***string***. Je to typ podobný ***char***, avšak s tým rozdielom, že tento typ je "inteligentný". Obsahuje tiež reťazec, pole znakov ako celok a pamätá si aj dĺžku reťazca, ktorý je v ňom uložený.

Napr. do ***array*** [***1..20***] ***of char*** sa vždy uloží aj vypíše práve 20 znakov, zatiaľčo premenná typu ***string*** nám vypíše len toľko znakov, koľko do nej uložíme.

Pomôžte vrátnikovi napísať nový program s využitím typu ***string****.*

Po napísaní programu však riaditeľstvo vznieslo novú požiadavku. Po "záverečnej" chcú mať aj výpis áut v abecednom poradí.

(Znaky a rovnako aj reťazce možno v Turbo pascale porovnávať. Platí, že 'a' <'b' <...'A'< .... <'Z'.)

Typ ***string*** nám umožňuje porovnávať reťazce ľahko a relatívne rýchlo.

```
for i:= 1 to Pocet_aut do
 for j:= 1 to Pocet_aut-i do
  if Pole[j+1] < Pole[j] then
                        Vymen (Pole[j ], Pole[j+1]);
for i:= 1 to Pocet_aut do WriteLn (Pole[i]);
```

Procedúra ***Vymen*** vymení medzi sebou hodnoty zadaných premenných.

V jeden slnečný deň, 15 minút pred koncom pracovnej doby, sa na riaditeľstvo dostala novina, že každý znak s ŠPZ vytlačený na tlačiarni spotrebúva atrament. Geniálny (generálny) riaditeľ preto po dlhej úvahe rozhodol, že pomlčky v ŠPZ zbytočne okrádajú kocúrkovský ľud o farbivo, a preto ich pri poslednom výpise (podľa abecedy), ktorý ide aj na tlačiareň, netreba.

Napíšte program, ktorý pred výpisom na obrazovku (tlačiareň) vymazáva z ŠPZ pomlčky.

Využijeme v Turbo pascale preddefinovanú procedúru:

```
Delete (reťazec, index1, počet vymazaných znakov);
```

index1 - index znaku, ktorý má byť vymazaný ako prvý

Napr.:

```
s:= 'jano'
Delete (s, 2, 2)                          {s = 'jo'}
```

a vymazanie pomlčiek:

```
for i:= 1 to Pocet_aut do begin
           Delete (Pole[i], 5, 1);
           Delete (Pole[i], 7, 1);
end;
```

Keď na druhý deň riaditeľ uvidel výpis, dostal srdcovú príhodu. Po návrate z pobytu v kúpeľoch prikázal vrátiť sa späť k pôvodnému výpisu. Lenže nastal problém: Vrátnika vymenili (resp. odišiel na lepšie platené miesto učiteľa) a nový sa nevyznal v myšlienkových pochodoch starého, preto nevymazal mazanie pomlčiek, ale pred výpisom opäť pomlčky povsúval. Ako to urobil?

Napíšte procedúru, ktorá povsúva pomlčky na pôvodné miesta.

V Turbo pascale existuje procedúra **Insert** v tvare:

```
Insert(vkladaný reť.,cieľový reť.,pozícia 1.znaku vkladaného reťazca)
```

Napr.:

```
s:= 'kola'
Insert ( 'by', s, 3)                  {s = 'kobyla'}
```

a pre vloženie pomlčiek:

```
for i:= 1 to Pocet_aut do begin
    Insert ('-', Pole[i], 5);
    Insert ('-', Pole[i], 8);
end;
```

Niekoľko týždňov všetko pekne klapalo, až kým si účtovník nezmyslel, že potrebuje súčet číselných častí značiek áut, napr. pre KOCO-10-34 je číselná časť 1034. A zodpovední boli zasa nútení doplniť program o ďalšiu časť (tak už to v živote programátora chodí).

Napíšte procedúru, ktorá zistí súčet číselných častí kocúrkovských značiek.

Z každého reťazca musíme povyberať čísla, teda znaky na 6., 7., 9. a 10. mieste a sčítať ich pre všetky autá. Lenže náš program by bol značne neprehľadný, preto ho prepíšeme pomocou ***case of*** (toto nový vrátnik urobiť vie). Dá sa teda zapísať plán riešenia úlohy, kde sú zahrnuté tieto možnosti:

1. nové auto,
2. výpis áut,
3. vypustiť pomlčky,
4. pridať pomlčky,
5. súčet,
6. koniec.

Keď návrh prepíšeme do tvaru programu a snažíme sa o zistenie súčtu, potrebujeme prostriedok, ktorý je schopný číslicové znaky pretransformovať na čísla (lebo výsledok sčitovania dvoch reťazcov je reťazec dlhý ako tieto dva reťazce dokopy). Skrátka, číselnú časť značky auta vložíme do reťazcovej premennej, zmeníme ju z reťazca na číslo a všetky číslice sčítame.

Použijeme funkcie ***Copy***, ***Concat*** a procedúru ***Val***.

Funkcia ***Copy*** zo zadaného reťazca skopíruje časť do iného reťazca:

```
Copy ([reťazec, z ktorého berieme], [index začiatku], [počet znakov]),

s:= Copy('značka' , 3, 4)            {s ='ačka'}
```

Funkcia ***Concat*** zlúči (sčíta) ľubovoľný počet reťazcov:

```
s:= Concat('ABC','CE','FGH')   {s ='BCCEFGH'}
```

Ten istý efekt možno docieliť aj sčitovaním reťazcov:

```
m:= 'abc'+'cd'+'de'                  {m = 'abccdde'}
```

Procedúra ***Val*** urobí z reťazca číslo:

```
Val ([reťazec],[premenná,do ktorej sa uloží číslo],[chybový kód]),

Val ('123', i, code)
```

, pričom ***i*** a ***code*** sú integer,

pritom, ak nie je ***code=0***, tak reťazec nebol správne zadaný, boli v ňom napr. písmená. Premenná ***code*** obsahuje index znaku, pri ktorom došlo k omylu.

Riešenie môže mať tvar:

```
 Sucet:= 0;
 for i:= 1 to Pocet_aut do begin
     Prva:= Copy (Pole[i ], 6, 2);
     Druha:= Copy (Pole[i], 9, 2);
     Cislo:= Concat (Prva, Druha);
     Val (Cislo, Number, Code);
     if Code <> 0 then WriteLn ('Chyba v znacke c. ', i)
 else Sucet:= Sucet + Number;
 end;
WriteLn ('Sucet znaciek je ', Sucet);
```

Asi po mesiaci sa na pretras dostala otázka bezpečnosti údajov uložených v počítači. Riaditeľ - teraz už prezident spoločnosti - hlavička to prefíkaná, vymyslel, že pri spustení programu sa bude žiadať heslo. A ak operátor zadá heslo s dĺžkou inou ako 9 znakov, program ho "k sebe" nepustí. Chudák vrátnik, ktorý to opäť dostal ako "stranícku úlohu", našiel vo svojej "minipríručke" vhodný prostriedok s názvom ***Length***.

```
Dlzka:= Length (reťazec)
```

Funkcia ***Length*** dáva ako výsledok počet znakov v reťazci.

Doplňte do programu vstup podľa želania pána riaditeľa.

Potom úvod do programu vyzerá napr. takto:

```
Write ('Zadaj heslo ');
ReadLn (heslo);
if Length (heslo) <> 9 then Halt;
```

Tento "vynikajúci, superveľký" program, v Kocúrkove dosiaľ nevídaný, sa podarilo úspešne predať. Softwarová firma (Kocovina, s.r.o.), ktorá program kúpila, ho neváhala prerobiť do dokonalejšieho - grafického - tvaru, ale o tom až nabudúce.

Cvičenia:

1. Napíšte "fool-proof" (chybné zadania čísel nedovoľujúci a primerane komentujúci) program na sčítanie dvoch ľubovoľne dlhých celých čísel.

2. Na oddelenie špionážnej činnosti dostali moták. Potrebovali zistiť, ktorý z desiatich špiónov im ho poslal. Jeho meno sa kdesi v motáku vyskytovalo. Pre zadané mená špiónov zistite, ktorý z nich bol odosielateľom správy: "janositeľvyznamenaniadruhejtriedyoznamujem" (riešením môže byť napr. "jano", "edy").

3. Ďalší moták možno dešifrovať tak, že treba vziať každé tretie písmeno. Takéto motáky aj v centre špionážneho výskumu vyrábali. Napíšte program, ktorý daný reťazec zakóduje aj dekóduje.

4. Centrum špionážneho výskumu inovovalo na základe svojej šesťmiliónovej dotácie svoje kódovanie správ takto: Špión aj centrum pozná kódové číslo. Je to číslo, ktoré vyjadruje, o koľko znakov "ďalej" v abecede od 'a' po 'z' (anglická abeceda) je znak, ktorým máme prepísať daný znak, správy (napr. ak je kódové číslo 3, potom znak 'a' sa prepíše znakom 'd', znak 'b' znakom 'e' atď. a znak 'x' znakom 'a', znak 'y' znakom 'b', znak 'z' znakom 'c'). Napíšte program, ktorý pre zadanú správu a kódové číslo dekóduje, resp. zakóduje túto správu (znak "medzera" sa zakóduje ako znak "medzera"). (Kódové číslo môže byť aj záporné.)

5. Aby využilo centrum špionážneho výskumu ďalších šesť miliónov Kk (kocúrkovských korún) dotácie, vymyslelo pre svoje kódovanie správ takýto postup: Kódovací kľúč predstavuje veta, ktorá obsahuje všetky písmena (anglickej) abecedy. Každému písmenu správy potom priradíme poradové číslo jeho výskytu vo vete (dvojmiestne číslo). Písmeno (znak) sa ale vo vete môže objavovať viackrát, v tomto prípade mu môžeme priradiť poradové číslo hociktorého výskytu. Zakódovanú správu ukončíme dvojicou núl. Medzera sa vynecháva. Predpokladajme, že poznáme kódovací kľúč a že sme dostali zakódovanú správu. Je potrebné vytvoriť program pre jej dekódovanie na základe zadania textu správy a príslušného kódovacieho kľúča. (Napr. kľúčom môže byť veta:
"najmilsiuzostavupismenhucifugatvorikazdydobryretazecqwx" - môžu sa vymyslieť aj lepšie kódovacie kľúče.) (Zabezpečte aj zakódovanie motáku.)

1.2 Ugandskí špióni (RECORDY)

Šéfa ugandskej spravodajskej služby prebudil zlý sen. Snívalo sa mu, že neznámi páchatelia vykradli jeho pracovňu a okrem iného vzali aj kartotéku s menami všetkých špiónov v štátnych službách. Rýchlo zdvihol telefón, no spomenul si, že vo sne mu z kancelárie ukradli aj ten, a tak bez rozmýšľania zavesil.

Skočiac do papúč, hnal sa do garáže, kde naštartoval svoj služobný bicykel a s revúcim motorom, obchádzajúc všetky dopravné predpisy, sa rútil do kancelárie.

Na mieste ho čakalo milé prekvapenie: Okolo objektu Ministerstva vnútra bolo zoradených 15 hasičsko-požiarnickych áut a okolo dvesto požiarnikov.

Oznámili mu, že jeho kancelária a priľahlé priestory práve odolali útoku teroristov a tí od zlosti podpálili neďalekú stodolu.

Pád ťaživého balvana zo srdca najvyššieho špióna spôsobil v priľahlých oblastiach zemetrasenie o sile 4 stupňov Richterovej stupnice.

Šéf USS si ešte toho dňa uvedomil, že na prvý raz sa to síce zločincom nepodarilo, ale po opakovaných pokusoch by sa mohlo stať, že budú sláviť úspech.

Rozhodol sa preto všetky údaje: správy, dokumenty, príkazy, kartotéku a ostatné byrokratické záležitosti uložiť do počítača a zašifrovať, aby prípadní zlodeji neodhalili štátne machinácie a štátnych špiónov.

Dokumenty, správy a všetko ostatné zveril svojim podriadeným, no kartotéku špiónov, ktorá bola najtajnejším materiálom v Ugande sa rozhodol spracovať sám.

So svojimi vedomosťami najprv uložil údaje o špiónoch do polí. Prvé pole obsahovalo meno, druhé priezvisko (občas až dvadsaťslovné - ako je v Ugande zvykom), ďalšie vek, hmotnosť, výšku, počet jazykov, ktoré špión ovláda, a špecializáciu.

Napíšte program, ktorý bude evidovať všetkých 18 ugandských špiónov (na väčší počet vláda nemá prostriedky) a na požiadanie vypíše všetky údaje o zadanom. Špiónov stačí zadávať jedným z priezvisk.

Tento prostriedok - evidovanie vo viacerých poliach, kde položky s rovnakým číslom napr. meno[7], vek[7], jazyky[7] zodpovedajú jednému špiónovi - sa však pri komplikovanejších operáciách (napr. zoradenie podľa abecedy, hmotnosti...) stal veľmi nevyhovujúcim a práca s ním zdĺhavou. Preto sa pán autor rozhodol využiť v pascale definovaný typ ***record***.

Je to typ, ktorý sa skladá z niekoľkých položiek (môžu byť rovnakých alebo rôznych typov). Pre špióna to môže byť napr.:

```
Type Spion = record
            Meno: string;
            Priezvisko: string;
            Vek: byte;
            Hmotnost: byte;
            Vyska: byte;
            Pocet_jazykov: byte;
            Specializacia: string;
end;
```

Možno písať aj úsporne:

```
Type Spion = record
     Meno, Priezvisko, Specializacia: string;
     Vek, Hmotnost, Vyska, Pocet_jazykov: byte;
end;
```

Jednotlivé položky môžu byť ľubovoľného definovaného typu, môžu byť zapísané v ľubovoľnom poradí a môže ich byť ľubovoľný počet.

Premenné typu ***record*** definujeme klasicky:

```
var Velky, Maly: Spion;
```

Ak chceme pracovať s konkrétnymi hodnotami, priraďovať, porovnávať... pracujeme s ***recordom*** pomocou selektora "**.**".

```
Maly.Vek:= 99;          {uloží do položky Vek špióna Maleho - 99 rokov}

Write (Maly.Meno);      {vypíše Meno špióna, ktorý je uložený v premennej Maly}

Velky:= Maly;           {všetkým položkám premennej Maly priradí všetky položky
                        premennej Velky}

ReadLn (Maly);          {kompilátor ohlási chybu, lebo priamo možno načítavať len
                        položky recordu}

ReadLn (Maly.Vyska);    {načíta výšku Maleho}

Pocet_jazykov:=7;       {chyba, položku možno volať len cez meno recordu}
```

Je možná aj deklarácia

```
var Pluk_Spionov: Array[1..18] of Spion;
```

*Napíšte program, ktorý pomocou premenných typu **record** načíta špiónov a bude schopný utriediť ich podľa zadanej položky a vypísať.*

Pozn.: Všimnite si, že pokiaľ ste v prvom príklade museli pri triedení vymieňať prvky v každom poli osobitne, tu stačí použiť jedinú výmenu.

Špionážna sieť pracovala ako hodinky, no po čase všetkých špiónov okrem Hugoštvána Harnoleja pochytali. A jeho finančné požiadavky na zamestnávateľov boli také vysoké (rástli spolu s infláciou), že USS si nemohlo dovoliť prijať nových špiónov.
Využívajúc doklady (pozostatky) pochytaných členov USS, mohol Hugo každé dva týždne meniť svoju totožnosť.
Pre špióna je ale, okrem iného, veľmi dôležité aj to, aby monogramy povyšívané na vreckovkách, šáloch, košeliach a ponožkách súhlasili so začiatočnými písmenami mena, ktoré má v dokladoch.

Napíšte procedúru, ktorá pre zadané iniciálky mena (dve) vypíše všetky použiteľné totožnosti, t.j.: zadané iniciálky musia zodpovedať v poradí buď menu a priezvisku alebo priezvisku a menu. (Napr. pre IJ: Ilona Jarabošová, Ivan Jašo, Jano In atď.)

Razí sa však stalo, že Hugo, cestujúc po zahraničí, vytiahol doklady podľa ktorých mal 8 rokov a len o chĺpok unikol guľkám, ktorými ho zasypali škorpióny nedôverčivých strážcov letiska.

Napíšte procedúru, ktorá pre zadaný vek vypíše totožnosť osoby, ktorej vek je rovnaký alebo sa líši o minimum rokov.

Ďalšie, veľmi podobné problémy nastali, keď sa chcel Hugo vyviesť na 218. poschodie výťahom. Obsluha bola ochotná prevádzkovať výťah len od určitej hmotnosti. Okrem toho nesmela hmotnosť posádky vo výťahu prekročiť maximálnu povolenú.

Napíšte procedúru, ktorá pre zadanú maximálnu a minimálnu hmotnosť zákazníka vyhľadá Hugovi vhodnú totožnosť.

Pozn.: Existuje pomôcka, vďaka ktorej netreba počas práce s ***recordom*** vypisovať celý prístup k jeho položkám. Stačí použiť ***with***:

```
with Maly do begin
            ReadLn (Meno);
            ReadLn (Priezvisko);
            ReadLn (Vek);       ...
end;
```

Vykoná to isté ako:

```
ReadLn (Maly.Meno);
ReadLn (Maly.Priezvisko);
```

```
ReadLn (Maly.Vek);
     ...
```

Pozn.: Súčasťou ***recordu*** môže byť aj iný ***record***:

Cvičenie:

1. Navrhnite record, v ktorom budete evidovať knihy, a napíšte procedúry, ktoré budú schopné nájsť, pridávať, mazať a vyhľadávať a zisťovať cenu pre rovnaké knihy.

2. Napíšte program, ktorý dokáže evidovať žoldnierov na pirátskej lodi, dokáže vypočítať ich plat podľa počtu akcií, ktorých sa zúčastnili, a v prípade ich smrti rozdelí majetok rovným dielom medzi ostatných. (Record bude obsahovať nielen základné údaje, ale aj počet akcií - za každú môžete dať rovnakú odmenu - a celkový majetok).

3. Vytvorte zoznam zvierat v ZOO. Evidujte ich hmotnosť, vek, počet mláďat, ... Vytvorte procedúru, ktorá vyberie zvieratá so zadaným počtom mláďat, prípadne zadaným vekom.

4. Vymyslite štruktúru, ktorá dokáže reprezentovať klasické bludisko: možno sa pohybovať štyrmi smermi, medzi susednými miestnosťami môže alebo nemusí byť priechod.

5. Navrhnite štruktúru, pomocou ktorej by ste boli schopní (čo najprehľadnejšie) evidovať údaje o prospechu žiakov počas štyroch rokov štúdia na škole.

(Nesnažte sa definovať celú štruktúru naraz. Umenie programovať je skryté práve v rozdrobení problému na mnoho elementárnych častí, ktoré spolu dávajú výsledok.
Najprv treba definovať zoznam (record) predmetov pre jeden polrok, z neho pre celý rok a výsledná štruktúra môže vyzerať takto:

```
Type Studium=record
        meno:string;
        adresa:string;
        rc:rodne_cislo;
        rok:array [1.4] of vysvedcenie;
        ...
end;).
```

6. Navrhnite štruktúru záznamu pre predajňu počítačov. (Typ počítača, veľkosť disku, ...).

1.3 Mäsožravá mucholapka (FILE)

V jednom malom meste na okraji malého sídliska sa v malom lesíku tiesni niekoľko veľkých budov s nápisom Biologický ústav pre výskum netradičných biologických kultúr.

Jedného krásneho, mesačného večera sa ozvalo pri kancelárii pána Kalerába, riaditeľa ústavu, zvonenie. Pán riaditeľ, ako každý iný človek, trochu zaodŕhal, prešiel si rukou po vlasoch a zakričal:

- Ďalej!

Nič.

- Ďalej! - zahulákal na dvere znova.

To isté - nič.

"Niekto si tu zo mňa vystreľuje, " - napadlo ho a otočil sa ku dverám chrbtom.

Práca mu šla od ruky (bodaj by nie, popíjať z hrnčeka čaj dokáže bez problémov už aj trojročné decko) a tak skončil ešte pred polnocou. Pozhasínal svetlá, zamkol trezor a vykročil von.

- Dofrasa! Kto sem dal ten kvetináč! - zanadával zakopnúc do veľkého črepníka s nádherne vyvinutou rastlinou.

- A čo je to? - sklonil sa k nemu. - Dionaea muscipula. Mucholapka! - stihlo mu dôjsť krátko na to, ako sa mu okolo krku ovinulo jedno z ramien milej kvetinky.

Doktor Kaleráb nebol z tých, ktorí hneď strácajú hlavu, a namiesto svojej plešiny ponúkol rastlinke kúsok hovädziny, ktorý mu zostal od obeda v taške.

Mäsožravá rastlina? To bude super!

A na druhý deň už mucholapka chytala muchy v areáli Biologického ústavu pre výskum netradičných biologických kultúr.

Ako už sám názov hovorí, skúma sa tu správanie a život exotických rastlín a živočíchov. Pre každého jedinca majú založenú knihu správania, do ktorej zaznamenávajú každodenný stav a na základe správania sa v jednotlivých dňoch roka potom výskumníci zisťujú pravdepodobné reakcie v nezvyčajných podmienkach (napr. šancu organizmu prežiť víkend na Mesiaci; možnosti vývinu v sopečnom kráteri atď.).

Keď chcú zistiť napríklad reakciu anakondy paraguajskej na opaľovací krém Sunia, vložia do počítača údaje o teplote ovzdušia a agresivite anakondy za každý deň v roku a program im za pomoci zložitých výpočtov určí, ako sa bude anakonda správať po jeho požití.

Spočiatku s vkladaním údajov neboli žiadne problémy, no keď tretia sekretárka dala výpoveď kvôli chronickému presileniu zápästí, začal pán riaditeľ uvažovať o radikálnej zmene prístupu k práci.

Prečo by sa mali údaje do počítača vkladať vždy znovu, pri každom novom výpočte? Čo keby sa vložili len raz a pre každé ďalšie použitie by boli stále poruke?

Hm, ale to by sa počítač nesmel vypnúť! Naozaj? Čo keby sme ich uložili na disketu (alebo na pevný disk)? Počítač môžeme vypnúť koľkokrát chceme a keď budeme údaje potrebovať, vložíme disketu a pracujeme tam, kde sme prestali.

Údaje sú na diskoch a disketách uložené v súboroch. Ak chceme uložiť nejaký údaj, musíme zadať súbor, do ktorého ho zapísať. Ak chceme údaj vybrať (prečítať), opäť musíme určiť, v ktorom súbore ho hľadať.

Takto je jednoznačne určené, kde čo je, a nemôže sa stať, že by sme načítali iný údaj, ako chceme, alebo že by sme nenašli, čo sme raz uložili.

Každý súbor v pascale je nejakého typu (***integer***, ***char***...). Môžeme teda doň zapisovať len údaje, ktorých typ je totožný s typom súboru. (Do súboru typu ***integer*** nám kompilátor dovolí zapisovať len celé čísla, do ***char*** znaky ap.) Rovnako je to aj s čítaním.

Súbor, do ktorého chceme údaje ukladať, však nemusí vôbec existovať. (???) Vieme si ho vytvoriť.

*Napíšte procedúru, ktorá uloží údaje o teplote počas januára do súboru **januar.tpl**.*

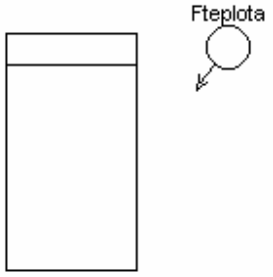
Ak chceme so súborom pracovať, musíme ho deklarovať v deklaračnej časti.

```
Program Praca_so_suborom;

var  Fteplota: File of integer;
           {teplotu budeme ukladať v celých číslach}
     Denna_teplota: integer;
     i: integer

begin
 Assign (Fteplota, 'c:\januar.tpl);
           {pripravíme, inicializujeme súbor
           c:\januar.tpl; ak nezadáme celú cestu, len
           meno súboru, tento sa vytvorí v aktuálnom
           adresári;
           Fteplota je premenná, v ktorej je uložená adresa
           miesta na disku, kam sa ma údaj zapísať alebo
           odkiaľ sa ma prečítať}

 Rewrite (Fteplota);     {pripraví súbor na zapisovanie;
                          ak ešte neexistuje, vytvorí ho;
                          POZOR! Ak súbor už existuje,
                          zmiznú z neho všetky údaje a
                          pripraví sa len na zapisovanie
                          nových}

 for i:=1 to 31 do begin
           {cyklus, ktorý zabezpečí zopakovanie načítania
           pre všetkých 31 januárových dní}

Write('Zadaj teplotu 'i, '. januara');
ReadLn(Denna_teplota);
Write(Fteplota, Denna_teplota);
           {zapíše do otvoreného súboru definovaného
```

```
            ako Fteplota údaj o teplote}
 end;
 Close (Fteplota);       {keď prácu so súborom ukončíme,
                         treba ho zavrieť}
end.
```

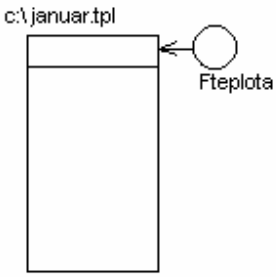
Ako to celé prebiehalo v počítači?
(zjednodušene)

Assign(Fteplota, 'c: \ januar.tpl');

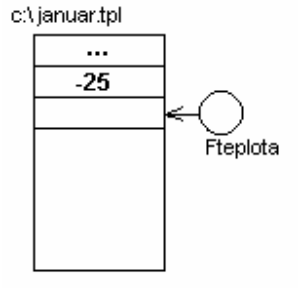


Vznikla bunka, ktorá sa spojila so začiatkom súboru (ak existoval). Pokiaľ súbor ešte nebol vytvorený, bunka čakala, čo bude ďalej.

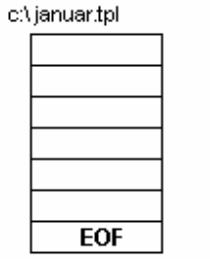
Nasledovalo:
Rewrite(Fteplota);
Bunka vytvorí súbor bez ohľadu na to, či existoval alebo nie (t.j. ak existoval, bude prepísaný) a nastaví sa na jeho začiatok.



Write(Fteplota, Denna_teplota);
Zapíše sa hodnota uložená v premennej ***Denna_teplota*** na miesto, kam bunka ukazuje a posunie ju o políčko ďalej.
....
To isté sa zopakuje 31 ráz. Bunka vždy zapíše hodnotu a posunie sa ďalej.



Close(Fteplota);
Súbor sa zatvorí, na poslednú položku dá znak **EOF** a bunka ***Fteplota*** zanikne.



Fteplota je v podstate smerník, ktorý ukazuje na adresu na disku, kam sa bude zapisovať alebo odkiaľ sa bude čítať.

Uvedený program nám údaje uložil, ale čo keď ich chceme čítať a spracúvať?

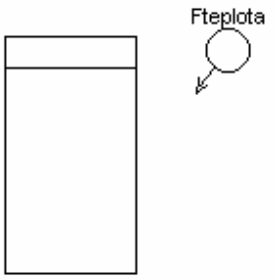
Napíšte procedúru, ktorá prečíta, vypíše uložené údaje o teplote a zistí priemernú teplotu.

```
Procedure Precitaj_teplotu;
var   Fteplota: File of integer;
      Denna_teplota, Sucet_teplot: integer;
      i: integer
      Priemer_teplot: real;

begin
 Assign (Fteplota, 'c: \januar.tpl);
                          {pripraví opäť súbor na prácu}

 Reset (Fteplota);
                          {vo všeobecnosti pripraví súbor
                          na čítanie;
                          ak súbor neexistuje, počítač
                          vyhodí chybu a program ukončí}

 for i:=1 to 31 do begin

      Read (Fteplota, Denna_teplota);
                          {číta teplotu zo súboru}

      Sucet_teplot:= Sucet_teplot + Denna_teplota;

      WriteLn (i, '. januar - ', Denna_teplota);
 end;

  Close (Fteplota);       {zatvorí súbor}
```

```
   Priemer_teplot:= Sucet_teplot / 31;
   WriteLn (' Priemerna teplota za mesiac januar je ',
Priemer_teplot);
 end;
```

Opäť:

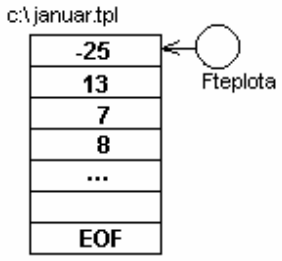
Assign (Fteplota, 'c: \ januar.tpl');



Vzniká bunka

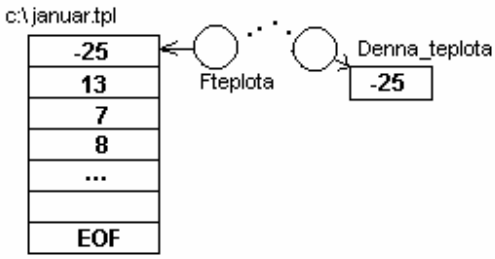
Reset (Fteplota);

Ak súbor neexistuje, bunka sa nemá kam nastaviť, nevie, čo so sebou, a preto vygeneruje chybu. Inak sa nastaví na začiatok súboru. (Súbor už existuje, teda sú v ňom uložené aj údaje.)



Write (Fteplota , Denna_teplota);

Bunka vezme zo súboru hodnotu, na ktorú práve ukazuje, a vloží ju do premennej ***Denna_teplota***. Bunka sa potom posunie na ďalší prvok súboru.



V tejto situácii možno zo súboru nielen čítať, ale doň aj zapisovať:

Write(Fteplota , Denna_teplota);

Do súboru by sa zapísala hodnotu uložená v premennej ***Denna_teplota***, teraz ***-25***. Bola by zapísaná na miesto, kam ukazuje bunka. Teda v súbore by po vykonaní tohto príkazu bolo -25, -25, 7, 8 ... (hodnota 13 bola prepísaná). Po prebehnutí operácie bunka zasa ukazuje o políčko ďalej.

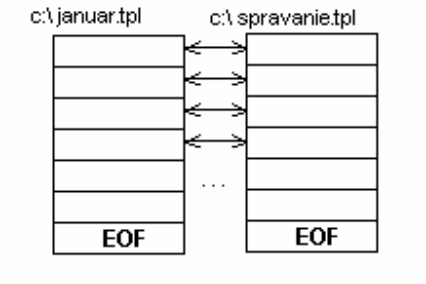
V prípade, že chceme čítať a bunka ukazuje na koniec súboru (**EOF**), vieme túto hodnotu prečítať a vieme ju aj otestovať:

```
Read(Fteplota, Hodnota);
if Hodnota=EOF(Fteplota) then WriteLn('Koniec suboru ');
```

Pán riaditeľ Kaleráb vyriešil problém s pisárkami, no potreboval uložiť nielen teplotu, ale aj správanie sa organizmu pri nej.

Na vyriešenie tejto požiadavky máme niekoľko možností:

1. Uložíme do jedného súboru teplotu, do druhého správanie sa rastliny. Druhý súbor môže byť typu ***char*** a správanie bude kódované:



A - výborné
B - veľmi dobré
C - dobré
D - dostatočné
E - agresívne

Teplota a správanie sa v ten istý deň budú uložené v zodpovedajúcich si políčkach súboru.

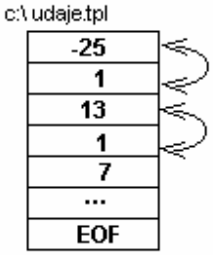
Pozn.: Budú otvorené dva súbory naraz

2. Budeme vkladať do jediného súboru, a to tak, že najprv uložíme teplotu, potom správanie. Napr.:

```
Write(Fteplota, Denna_teplota);
Write(Fteplota, Spravanie);
```

Nastane tu však malý problém. Súbor, do ktorého ukladáme teplotu, je typu ***integer***, druhý je typu ***char***. (???)

Vyriešime to prostým prekódovaním správania:



1 - výborné
2 - veľmi dobré
3 - dobré
4 - dostatočné
5 - agresívne

... a po uložení si budú zodpovedať dvojice údajov za sebou.

Napíšte procedúry, ktoré uložia aj prečítajú teplotu a správanie sa mucholapky počas januára. Správanie sa bude vypisovať dešifrovane, t.j.:

nie 30. januára -15° C A,
ale 30. januára -15° C výborné.

Na riešenie problému si vyberte jeden z predošlých spôsobov (alebo aj lepší).

Existuje však ešte ďalší, a podľa autora najpohodlnejší, spôsob, ako vyriešiť danú úlohu.

Súbor môže byť **ľubovoľného** typu, teda aj typu definovaného užívateľom či typu ***record***, ***array*** a ďalších viac či menej zložitých, prípadne zo spomínaných typov poskladaných štruktúr.

teda:

```
Type Udaje_o_Mucholapke = record
     Teplota: integer;
     Spravanie: string;  {môžeme smelo opísať správanie
                          až na 255 znakov}
end;

var  Fmucholapka: File of Udaje_o_Mucholapke;
     Udaj: Udaje_o_Mucholapke;
```

Premenné typu ***record*** sa ukladajú rovnako ako premenné jednoduchých typov. Tak isto sa aj čítajú.

- Keď sme spoznali nástroj na jednoduché spracúvanie súborov, tak prečo ho plne nevyužiť? - povedal si pán riaditeľ. - Ak spracujeme okrem teploty aj ďalšie údaje, budú výpočty týkajúce sa správania ešte presnejšie.

Zapojili teda do výpočtov:

1. teplotu,
2. vlhkosť ovzdušia,
3. hluk,
4. známku za správanie,
5. poznámky ku správaniu.

Napíšte procedúru, ktorá všetky tieto údaje vloží do súboru.

Napíšte ďalšiu procedúru, ktorá zistí deň, v ktorý bol najväčší hluk, vypíše známku a poznámku ku správaniu sa mucholapky.

Po vyriešení týchto najpálčivejších a najdôležitejších záležitostí prišli všedné dni.

Taký všedný deň prebieha úplne normálne: Ráno sa príde do práce, nakŕmia sa zvieratá a mäsožravé rastliny, polejú sa rastliny - vegetariánky, do priestorov sa vpustí niekoľko slnečných lúčov atď., atď.

Na záver pracovného dňa sa odčítajú dôležité priemerné hodnoty a vložia sa do počítača.

Musia sa, samozrejme, uložiť na koniec súboru.

Neexistuje procedúra, ktorá by bola schopná priamo zapisovať na koniec súboru. Vždy sa tam treba nejako prepracovať.

Napíšte procedúru, ktorá zistí počet prvkov v súbore a procedúru, ktorá zapíše zadaný prvok na jeho koniec.

Telo procedúry na zistenie veľkosti súboru môže vyzerať takto:

```
Reset(Subor);
Pps:= 0;                  {Počet prvkov v súbore}
while not (Eof (Subor)) do begin
                    Pps:=Pps+1;
                    Read (Subor, Udaj);
                          {tým, že sme prečítali všetky
                          prvky, zistili sme koľko ich je}

end;
```

Zostalo vytvoriť procedúru na pridanie prvku. Na koniec súboru môžeme prejsť testovaním EOFu (pozri hore) alebo prečítaním všetkých prvkov (ak poznáme ich počet).

```
Reset (Subor);            {teraz sa vrátime na začiatok
                          súboru}

for i:=1 to Pps do
     Read (Subor, Udaj);        {sme nastavení na koniec
                                súboru}

Write (Subor, Posledny_Udaj);   {a môžeme zapísať}
```

Takéto neustále "prechádzanie sa" po súbore je príliš zdĺhavé a aj napriek tomu, že priamo na koniec súboru zapisovať nemožno, dá sa tam nastaviť. Zabezpečí to procedúra ***Seek***:

```
Seek(Subor,Miesto_kam_sa_nastavit);
```

Miesto_kam_sa_nastavit je číslo položky v súbore, t.j. poradové číslo položky. Začína sa počítať od 0, t.j. prvý údaj je na mieste 0, druhý na 1 atď.

Poslednú položku zistíme z veľkosti súboru:

```
Velkost:= FileSize (Subor);
```

Upravte procedúru na zapisovanie poslednej položky do rozumnejšieho tvaru.

Pán Kaleráb, hrdý na perfektné spracúvanie údajov, sa potreboval pochváliť, a tak pozval na exkurziu delegáciu z Japonska.

Šikmookí návštevníci si, samozrejme (a možno len zo slušnosti), pochvaľovali prístup a postup ústavu. Okrem iného boli zvedaví aj na správanie sa mucholapky počas veľkého zemetrasenia v Kalifornii (27.2.) - v 58. deň roka.

Napíšte procedúru, ktorá zistí správanie sa mucholapky v zadaný deň roka. Pre jednoduchosť budeme zadávať len deň roka, nie mesiac a deň mesiaca.

Z predošlého sa vieme nastaviť na ľubovoľné miesto v súbore, teda by nemal byť problém úlohu vyriešiť.

Občas sa stávalo, že niektorý z pestovaných či chovaných organizmov sa potreboval presťahovať (zmeniť miestnosť alebo dokonca opustiť Biologický ústav pre výskum netradičných kultúr). Vtedy sa predošlé údaje (aspoň podľa vedenia) stávali bezpredmetnými, lebo zmena prostredia zmenila aj reakcie jednotlivých organizmov.

Súbory so zhromaždenými údajmi teda strácali význam a zbytočne zaberali drahocenné miesto na disku. Treba ich vymazať.

A na to slúži procedúra ***Erase***:

```
Erase(Fsubor);
```

Napíšte procedúru, ktorá zmaže zadaný súbor.
1. Inicializuje sa súbor (aby sa vedelo, ktorý treba zrušiť),
2. Vykoná sa samotný akt.

Po čase odišiel starý programátor do dôchodku a na jeho miesto nastúpil nový - mladší. Nepoznal systém práce a popri programovaní sa mu podarilo používaním ***Rewrite*** znehodnotiť (vymazať) niekoľko dôležitých súborov.
Po ostrej diskusii s riaditeľom (väčšinou hovoril len pán Kaleráb), sa nový programátor rozhodol pri každom otvorení súboru najskôr zistiť, či ten už náhodou neexistuje.

*Napíšte funkciu, ktorá zistí, či zadaný súbor už existuje (pre jednoduchosť uvažujte konkrétne súbor typu **integer**).*

```
Function Existencia (Subor: string): Boolean;
var  Fsubor: File of integer;

begin
 Existencia:= FALSE;
 Assign(Fsubor, Subor);

 {$I-}
 Reset(Fsubor);
 {$I+}

 if IOResult = 0 then begin
                  Existencia:=TRUE;
                  Close(Fsubor);
 end;
end;
```

{$I-}*** vypne kontrolu chybového hlásenia. V normálnom prípade by pri neexistencii súboru nastalo násilné ukončenie programu kvôli chybe. Direktíva ***{$I-} však túto kontrolu vypne, a preto program pokračuje ďalej napriek chybe (ak nastala).
Výskyt chyby je však zapamätaný. Jej existencia je uložená v systémovej premennej ***IOResult***, ktorá má pri chybe hodnotu rôznu od nuly.
{$I+} registrovanie chyby opäť zapne a umožní nám tak kontrolu.

Po tejto epizóde sa život v BÚVNK vrátil do starých koľají. Občas síce mucholapka zožerie desiatu niektorému zamestnancovi, ale inak beží všetko postarom.

Cvičenie:

1. Vytvorte počítačového správcu, ktorý sa bude starať o majetky strýca Držgroša. Bude evidovať názov (domy, obrazy, kone,...), cenu, polohu (mesto, štát), počet kusov a poznámky.
*Na požiadanie vypíše zoznam, celkovú sumu, **rôzne** lokality, kde sa majetok nachádza a dokáže nájsť najdrahší a najlacnejší kus.*
Údaje sa budú ukladať do súboru a po novom spustení programu budú k dispozícii.

2. Napíšte program, ktorý bude mať na starosti priebeh Majstrovstiev sveta v hokeji. Bude evidovať zápasy (v súbore) a na požiadanie zobrazí aktuálnu tabuľku (poradie, skóre, počet bodov). Evidujte len zápasy (skóre), vytvorte procedúry, ktoré z nich dokážu ostatné údaje vypočítať.

1.4. Horlivý štatista (RANDOM)

Pišta Vrták bol dlhoročným zamestnancom Štatistického úradu pre neverejný výskum verejnej mienky. Jeho úlohou bolo vyberať zo štatistického súboru osoby a posielať im dotazníky.

Táto práca bola vysoko namáhavá, pretože vybrať z 50 000 obyvateľov mestečka náhodne asi tisícku občanov, napísať na obálky ich adresy, nalepiť známky a odniesť celý ten balík na poštu si vyžiadalo strašné množstvo energie.

Pišta bol však vytrvalý a svedomitý, preto mu táto práca začala zvyšovať hladinu žalúdočnej kyseliny až po niekoľkých rokoch. Začali ho umárať sny, v ktorých blúdil policami plnými adries a dotazníkov, dusili ho nezalepené obálky, na jazyku cítil chuť lepidla, prsty mu dreveneli od písania adries a ráno sa prebúdzal unavenejší ako večer, keď si líhal.

Po niekoľkých týždňoch, keď už svoju moru poznal naspamäť, rozhodol sa navštíviť svojho bývalého spolužiaka, ktorý bol v tom čase psychiatrom.

Po vrúcnom zvítaní a dlhodobom spomínaní sa konečne dostali k jadru veci. Pištove problémy hlboko doľahli na jeho vzdelanejšieho a inteligentnejšieho priateľa. Po niekoľkominútovom premýšľaní mu jednoznačne odporučil počítač, ktorý okrem toho, že všetku prácu urobí sám, dovolí mu odreagovať a zabaviť sa neodolateľnými softwarovými produktami - hrami.

Pišta teda na odporúčanie svojho "osobného lekára" podal na riaditeľstvo zlepšovací návrh - kúpiť PC.

Netrvalo dlho a v kútiku, na písacom stole, sa krčil a do miestnosti vyžaroval počítač s monitorom aj klávesnicou.

Radosť úradníka Vrtáka nemala hraníc. Svoju novú pomôcku každé ráno, skôr ako začal pracovať, zbavil prachu a vyleštil. Po dlhých dňoch neúnavnej práce sa mu napokon podarilo uložiť do stroja všetkých 50 000 meštiakov sídelného mesta a mohol sa začať projekt "automatizácie štatistických procesov".

Prvoradou úlohou bolo vybrať asi 1 000 - člennú vzorku obyvateľov, ktorej treba zaslať dotazník na tému "Čo si myslíte o používaní sprayov na ničenie hmyzu v detskej izbe?"

Na náhodné vygenerovanie náhodného čísla sa v pascale používa funkcia ***random***. Má tvar:

```
a:= Random (X),
```

kde ***a*** je premenná číselného typu ***integer***, ***real*** alebo typov odvodených, X je číslo alebo číselná premenná či konštanta.

Funkcia ***Random*** uloží do premennej ***a*** pseudonáhodné číslo z intervalu <0, X). Ak je premenná ***a*** celočíselná, výsledkom funkcie bude celé číslo, ak reálna, výsledkom bude číslo reálne.

Ak chceme vygenerovať čísla záporné alebo čísla z iných intervalov, použijeme jednoduchú matematiku.

```
Napr. pre celé čísla <1, 10>          a:= Random (10)+1
```

Výsledkom ***Random(10)*** bude celé číslo 0, 1, 2, .., 9, čiže maximálne 9. Ak chceme dosiahnuť na 10, pripočítame 1, ktorá zároveň zabezpečí, že v ***a*** bude najmenej 1. (Minimum ***Random(10)*** je 0.)

```
Pre (-10, 0>            a:= -Random (10)
pre <-5, 5)             a:= Random (10)-5   alebo
pre (-5, 5>             a:= 5-Random (5)
pre <-1, 1.5>           a:= Random (2.5)-1
                        kde a je typu real atď.
```

Napíšte program, ktorý vyberie zadaný počet náhodných evidenčných čísel zo súboru 50 000 osôb. (Výsledkom je výpis celých celých čísel z <1, 50 000>.)

Program, ktorý Pišta napísal, perfektne šľapal a po nejakom čase dokonca sám tlačil obálky a lepil známky. Všetko šlo ako po masle, až naraz...

Po niekoľkých desiatkach ankiet začali chodiť na Oddelenie pre styk s verejnosťou rozhorčené listy od adresátov, v ktorých sa sťažovali, že všetky dotazníky chodia len k nim, že ich sused nedostal ešte nič a že...

Listov chodilo stále viac, a tak sa správy dostali až na stôl sekčného šéfa. Ten si hneď predvolal na koberec podriadeného Vrtáka a žiadal vysvetlenie. Pišta celý ohúrený (veď to počítač) začal skúmať príčiny krízovej situácie a ...

Ak porovnáte dve sady náhodných čísel, t.j. ak spustíte predošlý program dva razy, zistíte, že v oboch prípadoch sú náhodné čísla úplne rovnaké. Čo robiť?

Pascal má okrem funkcie na generovanie "náhodných" čísel aj procedúru, ktorá tieto čísla ešte viac znáhodní:

```
Randomize
```

Dalo by sa povedať, že poradie, v akom budú nasledovať náhodné čísla, teda aj ich hodnota sú už vopred dané.

Pri spustení programu sa tieto čísla začali vypisovať od začiatku, teda od prvého. Procedúra ***Randomize*** však zabezpečí, že prvé číslo už nebude prvé.

Tento model je veľmi, veľmi zjednodušený, no na pochopenie problému snáď stačí. (V skutočnosti sa nové náhodné číslo vypočítava v závislosti na predošlom a procedúra ***Random*** toto predchádzajúce číslo zmení.)

Upravte predchádzajúci príklad tak, aby generovanie bolo skutočne náhodné.

Pozn.: Procedúru ***Randomize*** úplne stačí použiť v programe raz.

Napriek tejto úprave a oprave však verejnosť nekompromisne žiadala odvolanie pôvodcu svojich nepríjemností, a tak sa Pišta Vrták stal bývalým zamestnancom ŠÚNVVM.

Cestou z úradu práce rozmýšľal nad tým, kde by mohol najlepšie zveľadiť svoje trojmesačné odstupné. Jednoznačne mu vyšli hazardné hry a kasíno. No nechcel hrať tak, aby, ako mnohí pred ním, prehral gate aj nos medzi očami. Preto sa rozhodol pristupovať k problému vedecky.

Polovicu svojich peňazí investoval do kúpy osobného počítača a začal hľadať hru, v ktorej by mohla zvíťaziť vedeckosť nad náhodou.

Najsľubnejšou sa mu zazdali kocky.

Hádzalo sa dvoma kockami a tipovali sa súčty. Kto uhádol súčet, bral všetko. Tento problém možno pomerne jednoducho riešiť matematicky, no Pištovi inkriminovaný predmet k srdcu neprirástol, preto sa rozhodol pracovať pomocou počítača. Povedal si, že staviť na súčet, ktorý padne najviac ráz, je najistejšie. A aby bol výsledok skutočne reálny, treba simulovať hodenie kociek minimálne 1 000 ráz.

Napíšte program, ktorý bude náhodne hádzať dvoma kockami, teda generovať čísla, ktoré padnú (1..6), zistí súčet a po tisícom "hode", vypíše, ktorý súčet koľkokrát padol a na ktorý treba staviť.

Po uplatnení tejto stratégie a s hromadou šťastia sa podarilo Pištovi rozbiť bank. Hazard sa mu natoľko zapáčil, že sa rozhodol zisk investovať do postavenia novej herne a tam zaviesť súboje svojich zákazníkov proti počítaču. Neorientoval sa však na hry, pri ktorých existovala víťazná stratégia. Zameral sa na také, v ktorých hrajú hlavnú úlohu karty.

Napíšte program, ktorý pomieša 32 sedmových kariet. (Jednotlivé karty môžu byť reprezentované typom ***string****, v ktorom bude uložená farba + hodnota karty. Na začiatku sú karty nejakým spôsobom usporiadané).*

Vysoké zisky z prevádzky kasína umožňovali donovi Pištovi luxusný život...

... až do najbližšieho odvedenia daní. Daňoví úradníci ho ošklbali natoľko, že musel predať všetok svoj novonadobudnutý majetok, aby bol schopný vyrovnať dlhy so štátom.

Po vyrovnaní sa mu zostalo akurát na dve pivá a zmrzlinu.

Jediná hazardná hra, ktorú si teraz môže dovoliť, je pexeso.

Napíšte program, ktorý zamieša 50 kariet pexesa a uloží ich do obdĺžnika 5x10. Karty môžu byť reprezentované 25 písmenami abecedy.

Možno sa Pišta časom profesionalizuje a opäť zbohatne. Ktovie?

Cvičenie:

1. Napíšte program, ktorý bude vymýšľať matematické príklady (+,-,,/) a testovať vaše schopnosti. Počet príkladov zadáte pri vstupe a po ich vyriešení budete počítačom ohodnotení.*

1.5 Bangladéšski karatisti (NEW)

Kalendár ukazuje začiatok júla a na letné sústredenie má prísť karatistický výber z Bangladéša. Šikmookí priatelia, ako je typické pre obyvateľov trópov, nemienia bývať v luxusnom hoteli, ale rozhodli sa rozložiť si stany. Výpravu očakávajú každým dňom. Uvítacia komisia je pripravená, kuchár má zásoby v chladničke, len rozmraziť a vraziť do hrnca...

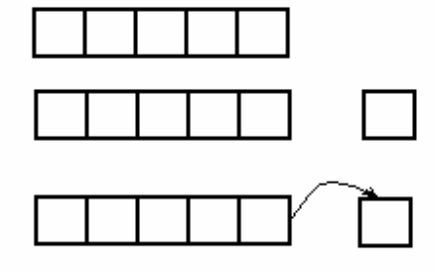
Zrazu však vyskočil na svetlo denné problém. Evidenčnú disketu, ktorá obsahovala počet stanov, mená a údaje športovcov, napadol vírus a patrične preformátoval údaje i seba samého.

Všetky informácie sú stratené, správca stanového tábora Kvapkajúca Voda na pokraji zrútenia, personál (kosič, hrabač, hasič a dvadsať manažérov) pobehuje ako splašené stádo mustangov po celom okolí a hľadá pomoc.

Po presnorení všetkých dier, putík a butikov sa im napokon podarilo zohnať inteligentného a mysliaceho človeka s programátorským vzdelaním - Miša.

Mišo, hlavička to cukrovo rafinovaná, našiel (po dvoch fľašiach (minerálky)) spôsob ako utajiť neschopnosť a babráctvo celého vedenia stanového tábora: pod zámienkou kontroly budú všetci pri príchode diktovať svoje údaje, ktoré sa kvôli dôveryhodnosti budú ťukať do počítača. Managerovi pre ubytovanie sa medzitým podarilo vydolovať z pamäti, že v každom stane je miesto pre 4 osoby, no na počet stanov si už nespomenul. Mišovi to však nevadilo a jedna diera v cudzej pamäti ho z miery nevyviedla. Vo svojej programátorskej práci sa už s takýmto problémom stretol a vie ho dokonca aj vyriešiť.

V normálnom prípade, t.j. vtedy, ak by sme poznali počet stanov, by sa údaje uložili do poľa záznamov - recordov. Keďže však počet stanov je veľkou neznámou, musíme pole simulovať (nahradiť) iným spôsobom. Na pomoc nám môže prísť procedúra ***new***, ktorá akoby (ak máme prirovnávať k poľu) vytvorila nový prvok poľa.



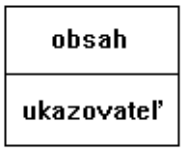
1. Máme pole.

2. Procedúra ***new*** nové políčko iba vytvorí,

3. preto ho k poľu musíme ešte "prilepiť".

Toto možno opäť simulovať tak, že každý prvok poľa rozdelíme na dve časti (***record***).

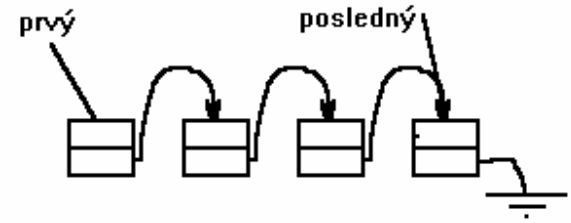
V prvom chlieviku bude obsah, ktorý môže predstavovať jeden prvok, celé pole, record ap.
V druhom bude ukazovateľ (smerník) na ďalší prvok.



Deklarácia môže vyzerať napr. takto:

```
Type Ukazovatel =^ prvok;
            Prvok = record
             Obsah: lubovolny_typ;
             Dalsi: Ukazovatel;
            end;
```

Prvý riadok je ukazovateľom na chlievik s recordom ***Prvok***



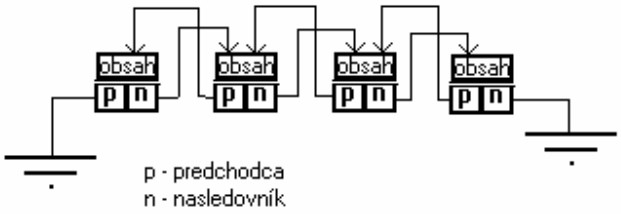
Ďalej je definovaný chlievik, ktorý obsahuje aj ukazovateľa na ďalší prvok.
Posledný prvok (ktorého ukazovateľ už neukazuje na ďalší chlievik (lebo žiadny nie je)), treba uzemniť (ukončiť) tak, že mu priradíme hodnotu ***NIL*** (nula):

```
posledny^.dalsi:=NIL;
```

Tým zabezpečíme, aby neukazoval (ak v ňom zostali hodnoty z kompilácie) na neexistujúci prvok, s ktorým by sme mohli mať neskôr problémy.
Na začiatok "poľa" - zoznamu - ukazuje vždy smerník, v ktorom je uložená adresa prvého prvku. Ak zmeníme jeho hodnotu svojvoľne, priamym priradením ap., môže sa stať, že stratíme spojenie s celým zoznamom (zmizne v spleti pamäťových miest - bude tam, ale nebudeme sa môcť k nemu dostať).
Z jedného recordu - chlievika, môže ísť aj viac ukazovateľov:
Pre deklaráciu:

```
Type Ukazovatel = ^prvok;
     Prvok = record
       Obsah: lubovolny_typ;
       Predchodca: Ukazovatel;
       Nasledovnik: Ukazovatel;
     end;
```

môže vyzerať systém takto:

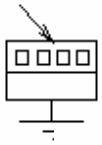


Mišo sa rozhodol pobehovať s počítačom po tábore a ukladať doň údaje tak, ako sa budú stavať stany. Keď sa postaví stan, Mišo napíše mená jeho obyvateľov a počká na ďalší. O karatistoch by mal vedieť meno, stupeň technickej vyspelosti (dan) a hmotnosť.

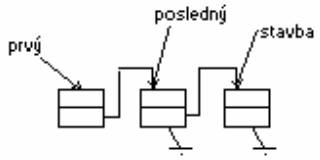
Po 22 hodinách sa Mišovi podarilo napísať program:

```
Type Osoba = record
             Meno: string;
             Dan: 1..10;
             Hmotnost: byte;
      end;

      Ukazovatel = ^Stan;
      Stan = record
             Obyvatel: Array[1..4] of Osoba;
             Dalsi: Ukazovatel;
      end;

var   Clovek: Osoba
      Prvy, Posledny, Stavba: Ukazovatel;
      Volba: char;
```

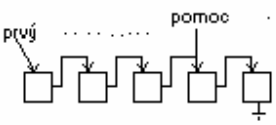
```
Procedure Novy;                          {postavíme stan}
```



```
begin
 if Prvy=NIL then begin
      New (Prvy);
      Napln (Prvy);
      Posledny:=Prvy;
  end else begin                         {pridáme
stan do zoznamu a budeme naň ukazovať ako na posledný}
      New (Stavba);
      Stavba^.Dalsi:= NIL;
      Posledny^.Dalsi:=Stavba;
      Posledny:=Stavba;
 end;
end;
```




```
Procedure Vypis;
var Pomoc: Ukazovatel;
 i: integer;

begin
 Pomoc:=Prvy;
while Pomoc<>NIL do begin
      for i:=1 to 4 do
      Writeln(Pomoc^.Meno[i],' ',
         Pomoc^.Dan[i],' ',
```



```
        Pomoc^.Hmotnost[i]);
     Pomoc:=Pomoc^.dalsi;
 end;
end;

begin
 Prvy:= NIL;
 Repeat
 WriteLn (' S - novy stan');
 WriteLn (' V - vypis');
 WriteLn (' K - koniec');
 ReadLn (volba);
 case volba of
     'S': Novy;
     'V': Vypis;
 end;
 until Volba='K';
end.
```

O pár dní, v pondelok, sa výprava dostavila na miesto určenia. Šihan (majster, tréner) a jeho žiaci boli po dlhom pochode (išli pešo až z Bangladéša) strašne unavení, a preto Mišovi jeho bluf vyšiel.

Na druhý deň, po dôkladnej prehliadke tábora, veľký učiteľ Han-mu-silk-wong našiel prehrešok voči etikete. Stany neboli usporiadané.

Stany sú usporiadané vtedy, ak naľavo je stan, ktorý má obyvateľa s najvyšším opaskom - hodnosťou. Ďalší stan - napravo - je taký, ktorého hodnostne najvyšší obyvateľ má stupeň buď rovnaký alebo nižší. Ostatných troch obyvateľov uvažovať nemusíme.

Usporiadanie môže byť napr.: 8311, 7444, 7666,

a nesmie byť: 8311, 9222.

Šihan po chvíľke rozmýšľania požiadal Miša, aby mu z vložených údajov pomohol zostaviť nové rozmiestenie stanov. Šihana s 10. danom do preskupovania zaraďovať netreba - spáva vonku.

Procedúra, ktorá vyrieši porušenie etikety bez kárnych opatrení, môže vyzerať takto:

```
Procedure Zoradenie;

begin
 if Prvy = NIL then Exit;       {ak nie sú stany, niet čo
                                 usporiadavať}

 Uprav_Stan (Prvy);       {Procedúra, ktorá prehľadá stan
                           a preusporiada ho tak, aby
                           "najvyšší" bol v premennej
                           Stan^.Osoba[1]}
 Sprac:= Prvy^.dalsi;
 while Sprac<>NIL do begin
     Pomoc:=Sprac;
     Uprav_Stan (Pomoc);
     Miesto:= Prvy;
     if Miesto^.Osoba[1].Dan<Pomoc^.Osoba[1].Dan then begin                    {z

           Pomoc^.Dalsi:=Prvy;
           Prvy:=Pomoc;
           end else begin
```

```
while ((Miesto^.Dalsi.Osoba[1].Dan>Pomoc^.Osoba[1].Dan)
            and (Miesto^.Dalsi<> NIL)) do
     Miesto:=Miesto^.Dalsi;
end;

if Miesto^.Dalsi=NIL then then begin  {zaradí stan na
                                       koniec}
     Pomoc^.Dalsi:=NIL;
     Miesto^.Dalsi:=Pomoc;
end else begin     {zaradí stan medzi stany; keby sme túto
                   činnosť robili v poli, museli by sme
                   najprv       nasledujúce prvky posunúť o jeden
                   a až potom vkladať}
Pomoc^.Dalsi:=Miesto^.Dalsi;
Miesto^.Dalsi:=Pomoc;
end;
Sprac:=Sprac^.Dalsi;
     end;
end;
```

STREDA:

Spokojnosť s Mišom karatisti prejavili tak, že mu založili devízové konto v miestnej pobočke miestnej banky. Okrem toho ho pozvali na klubový turnaj ako čestného diváka.

Turnaj sa uskutočnil v strede jazera, samozrejme, na ostrovčeku, kam všetci aktívny pretekári doplávali na vlastných (nie motorových člnoch, ale pupkoch).

Zápasilo sa vždy po dvoch, tak aby bol rozdiel výkonnostných stupňov súperov čo najmenší. Víťaz postupoval do ďalšieho kola.

*Napíšte procedúru, ktorá bude simulovať súboje jednotlivých reprezentantov, víťazov posunie do ďalšieho kola. Víťaza určite náhodne (pomocou funkcie **random**). Pretekár, ktorý nebude mať partnera postupuje do ďalšieho kola automaticky. Výsledkom nech je meno absolútneho víťaza.*

Návod: 1. Obyvateľov stanov zoradiť do nového reťazca, kde jeden prvok bude jeden športovec, a usporiadať podľa stupňov. 2. Do súboja brať vždy dvoch susedných (sú usporiadaní a teda majú aj najbližšie výkonnostné stupne).
3. Určiť víťaza. Porazeného z reťazca vyhodiť.
4. Vziať ďalšiu dvojicu. Pozor! - víťaz predošlého zápasu má právo na oddych.
5. Turnaj sa skončí, keď v zozname zostane len jeden bojovník.

Večer nasledovala veľká oslava, na ktorej sa jedlo a pilo - samozrejme striedmo.

ŠTVRTOK:

Prišlo ráno a niektorí z hodovníkov zistili, že sa nemôžu vďaka bolestiam v oblasti brušnej dutiny ani pohnúť.

Privolaný doktor (filozofických vied) konštatoval pravdepodobnú nákazu salmonelou a odporučil okamžitú evakuáciu a izoláciu tých stanov, v ktorých sa nachádzali chorí.

Napíšte procedúru, ktorá pre zadané mená (postihnutých) vyhodí všetky stany, v ktorých sa títo nachádzajú.

PIATOK:
Po neutešených 24 hodinách čakania sa šihan rozhodol ísť navštíviť svojich zdravotne postihnutých zverencov a tábor i tréningy zveril niekoľkým svojim najlepším žiakom. Prikázal im, aby si zvyšok tábora podelili na rovnocenné skupinky (t.j. aby v každej bolo zhruba rovnaké zastúpenie všetkých opaskov - výkonnostných stupňov).

Napíšte procedúru, ktorá pre počet vedúcich (najvyššie opasky) rozdelí tábor na skupinky približne rovnakej výkonnosti aj počtu.

SOBOTA:
Po šihanovom návrate z nemocnice sa všetci dozvedeli, že ich kolegovia neochoreli na baktérie, ale na náhlu zmenu podnebia. A preto budú všetci postihnutí v najbližších dňoch odexpedovaní do vlasti lietadlom Červeného kríža. Potešené dobrou novinou sa vedenie stanového tábora rozhodlo usporiadať banket, na ktorom sa malo podávať šampanské a kaviár. Hostia pozvaní neboli, banketu sa mohli zúčastniť len karatisti.
A tu sa objavil ďalší malý problém. Karatisti, rovnako ako aj ďalší vrcholoví športovci, majú presne predpísaný počet kalórií, ktoré môžu za deň prijať. Ich hodnota sa zisťuje podľa hmotnosti jedincov. Karatista smie zjesť len 1/1 000 svojej hmotnosti z kaviáru a vypiť 4% svojej hmotnosti šampanského.

Napíšte procedúru, ktorá zistí, koľko kg kaviáru a koľko l šampanského treba na banket priniesť (1l šampanského je asi
1 kg).

NEDEĽA:
A prišiel čas lúčenia...
Karatisti si zbalili stany a odišli.

Napíšte procedúru, ktorá vyčistí pamäť od údajov, ktoré boli do nej povkladané pri príchode karatistov a počas celej ich činnosti.

Každý objekt, ktorý vznikne pomocou procedúry ***new***, si v pamäti pri svojom narodení vyhradí miesto a drží si ho až do vypnutia počítača alebo do svojho zrušenia. Zrušenie jedného políčka (recordu) vykonáva procedúra ***dispose***.

```
Dispose (Objekt)
```

zruší políčko, na ktoré ukazuje smerníková premenná ***Objekt***. Zruší však len jeden ***record*** a ostatné recordy, ktoré sú jeho nasledovníkmi, zaberajú miesto ďalej. A pokiaľ neopatrne zrušíme ***record***, ktorý jediný na ne ukazoval, pamäť už od nich neoslobodíme, zostanú v nej stratené a miesto zaberajúce.

```
Procedure Odstranenie;

begin
 while Prvy<>NIL do begin
                   Pomoc:=Prvy;
                   Prvy:=Prvy^.Dalsi;
                   Dispose (Pomoc);
```

```
 end;
end;
```

Procedúra “citlivo” - po jednom - zlikviduje všetky ***recordy***.

*Doplňte **dispose** do všetkých vytvorených procedúr, v ktorých rušíte objekty.*

Cvičenia:

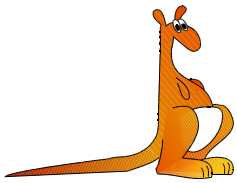
1. Napíšte program pre dopravnú políciu, ktorý bude evidovať autá (značky), ktoré sa dopustili priestupku, a sumu, ktorú ich bude priestupok stáť. V prípade viacnásobného porušenia predpisov sa nová pokuta pripočíta k už evidovaným. V prípade vyrovnania “účtu” sa auto z kartotéky vyhodí.

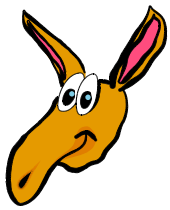
2. Študenti, ktorí predošlý deň zdrhli z vyučovania sa majú ísť ospravedlniť riaditeľovi. Nikto si však netrúfa, všetci sa boja. Preto sa dohodli na riekanke. Postavia sa do kruhu, budú vypočítavať, na koho padne posledná slabika - ten je mimo nebezpečenstva - vypadáva z kruhu potencionálnych ospravedlňovačov. Posledný, ktorý v kruhu zostane, ide za riaditeľom.
Napíšte program, ktorému zadáte mená všetkých študentov, počet slabík riekanky, náhodne ich zoradíte do kruhu a budete vypočítavať. Výsledkom bude meno študenta, ktorý zostane ako posledný.

3. Simulujte činnosť skupiny horských nosičov. Každý z nich bude reprezentovaný jedným recordom a každý z nich bude niesť náklad so zadanou hmotnosťou. Sú zoradení v lineárnom zozname (za sebou). V prípade, že niektorý z nich vypadne (alebo spadne), rozdelí sa jeho náklad rovnomerne medzi ostatných.
Napíšte program, ktorý dokáže vykonávať tieto operácie a na požiadanie vypíše mená nosičov aj s hmotnosťou, ktorú nesú.

2. Technika je takmer všetko

2.1 Austrálski školáci (Zásobník)





V austrálskom buši, kde jediným zákonom je zákon silnejšieho a zákernejšieho, kde v tieni storočných kmeňov a pod zeleňou obrovských listov syčia hady, tam, kde kengury klamú svojimi rýchlymi skokmi po krvi túžiacich nepriateľov, na miestach, kde panenská príroda stále odoláva tlaku civilizácie - tu žijú napriek všetkému aj ľudia.

Žijú tu, bývajú, majú deti a starajú sa o ne. Učia ich rozprávať (to kým je dieťa ešte úplne malé), jazdiť na koni (asi v tom istom čase - niektoré deti skôr vedia jazdiť ako rozprávať), zásady prvej pomoci ("Keď ťa uštipne včela, vytiahni si najprv žihadlo a až potom začni vrieskať!"), starať sa o hospodárstvo - dom alebo farmu - na ktorom žijú, a to je asi tak všetko. Veľmi málo detí sa doma naučí skutočne dobre čítať a písať (len počítanie im ako-tak ide vďaka tomu, že každý večer zisťujú, koľko oviec majú v ohrade).

Vzhľadom na neustály pokrok vo vývoji a vede sa skupina farmárov rozhodla posielať svoje deti do školy a vychovať z nich "mladé, perspektívne kádre, ktoré budú schopné zveľaďovať a rozvíjať rodinné dedičstvo".

Posielať deti do školy - to bola prvotná myšlienka. Lenže posielať a posielať je rozdiel. V Austrálii, kde dve obydlia sú od seba vzdialené často až niekoľko desiatok kilometrov, deti do školy pešo nemôžu. Viesť dopravný prostriedok tiež nemajú povolené, na koni by im cesta trvala príliš dlho a na internát ich rodičia nepustia. Čo s tým?

Po niekoľkých zasadnutiach farmárskej Rady starších padol návrh dopravovať deti do školy lietadlom. Počiatočné nadšenie však po niekoľkých minútach stlmil triezvy pohľad na veľkosť konta Rady starších. Na malé luxusné osobné lietadlo na ňom nebolo. Aj na veľkú helikoptéru chýbali asi tri štvrtiny. Napokon sa otcom rodín podarilo objaviť lietadlo (alebo to bol skôr vrak z prvej svetovej vojny), ktoré bolo pojazdné, zmestilo sa doň množstvo ľudí, ale malo tvar úzkej rúry - keď chcel vystúpiť človek, ktorý bol kdesi v strede, museli najprv vyliezť von všetci, čo boli medzi ním a dverami.

Nič lepšie sa nenašlo, a tak sa akcia rozbehla za pomoci lietajúceho veterána.

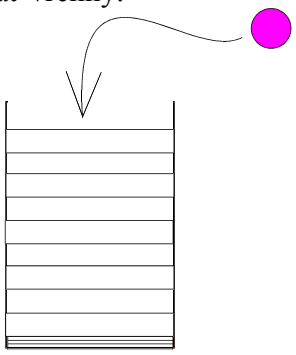
S odvozom do školy problémy neboli. Deti nastupovali, ukladali sa od posledného najhlbšieho miesta a na školskom letisku napokon bez problémov vystúpili.

Horšie to bolo pri ceste domov. Pilot najprv lietal tak, že odvážal domov toho frkana, ktorý sedel najbližšie ku dverám, no po troch mesiacoch Rada starších zistila, že spotreba benzínu je šialene vysoká (počas rozvozu sa tá istá časť trasy letela niekoľkokrát). Pilot si teda vypracoval efektívny plán letu a lietal podľa neho.

Pri tomto spôsobe rozvozu absolventov domácej materskej škôlky však musel na každom mieste postáť a čakať, kým vystúpia všetci, čo stoja vystupujúcemu v ceste, vystupujúci vystúpi a všetci zvonka sa zasa vrátia späť do lietadla.

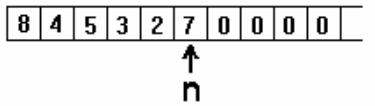
Pilot bol strašne nervózny človek a zúril, keď musel zbytočne čakať. Pri jednej takej dlhej zastávke sa rozhodol vypracovať plán alebo lepšie - zasadací poriadok, podľa ktorého si deti budú sadať do lietadla. Ten, ktorý ide posledný von, t.j. ten, čo posledný vystupuje, si sadá až na koniec lietadla a ten, ktorý to má domov najbližšie, sa ukladá pri dverách.

Takáto štruktúra sa nazýva zásobník (alebo LIFO - last in, first out - posledný dnu, prvý von). Prvky - tu žiaci - sa ukladajú navrch zásobníka a priamo sa možno dostať len k tomu, ktorý bol uložený naposledy, t.j. je na samom vrchu zásobníka. Ak chceme pracovať s iným prvkom, musíme zo zásobníka vybrať najprv tie, ktoré sú nad ním. Pri práci vieme urobiť len dve činnosti (operácie) - pridať nový prvok a vybrať vrchný.



Napíšte procedúru, ktorá pre zadané mená žiakov a zadanú vzdialenosť bydliska vypíše ich zasadací poriadok (taký, aby bol pilot čo najmenej nervózny) a uloží ich podľa neho.
(Pracujte s dynamickými premennými a kľudne predpokladajte, že v lietadle je nekonečne veľa miesta - neobmedzený priestor.)

Zásobník možno okrem dynamických štruktúr (pamätáme si skutočne len vrch a vieme, že niekde je dno - keď nasledovník je NIL) reprezentovať aj pomocou poľa.



Prvý prvok býva obyčajne dnom zásobníka a posledný jeho vrchom. Posledný prvok (alebo lepšie ukazovateľ na posledný prvok) býva premenná, v ktorej je uložený počet prvkov poľa. Zvyšné prvky sú obyčajne vynulované.

Schematické riešenie úlohy:

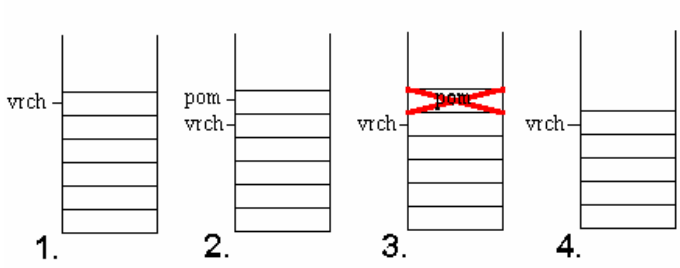
```
repeat
  Nacitaj_meno_a_vzdialenost
  Vloz_do_struktury_tak_aby_bol_system_usporiadany;
until Koniec;
Vypis_cely_system;
```

Pán pilot okrem toho, že bol nervózny, bol aj puntičkár (videli ste už hroznejšiu kombináciu?). Okrem tohto zoznamu si zmyslel, že chce aj meno žiačika, ktorý bude práve vystupovať.

Napíšte procedúry vysaď a vypíš vrchného. (Procedúra vysaď žiaka odoberie z vrchu zásobníka a procedúra vypíš vypíše vrchného, ktorý sa chystá vystúpiť).

Nech ***vrch*** je smerník na vrchol zásobníka typu napr.:

```
Type pZiak = ^Ziak;
     Ziak = record
            Meno: string;
            Vzdialenost: word;
            Dalsi: pZiak;
     end;
var Vrch: pZiak;


Procedure Vysad;
var Pom: pZiak;

begin
 if Vrch<>NIL then begin
                   Pom:= Vrch;
                   Vrch:= Vrch^.Dalsi;
                   Dispose (Pom);
 end else
                   Vypis (' Nemam koho vyhodit.');
end;
```



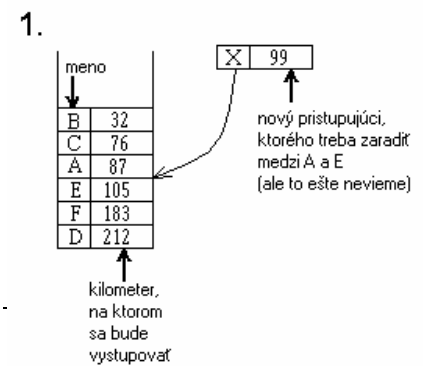
Ak je niekto v zásobníku, tak si zapamätáme smerník na najvrchnejší prvok (aby sme neskôr mohli od neho vyčistiť pamäť), potom ho vyberieme tak, že *vrch* posunieme o prvok nižšie a pôvodný vrch vymažeme.

Procedúra ***Vypis_vrchneho*** je triviálna.

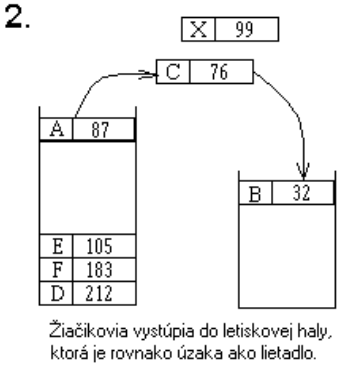
Postupne, ako čas bežal, sa letecká linka stala výhodným dopravným prostriedkom nielen pre malých žiačikov, ale aj pre ďalších obyvateľov nehostinnej buše. Nastupovali i vystupovali na zastávkach, kde lietadlo vykladalo žiakov, a vozili sa ním po celej trase.

Zozačiatku boli opäť problémy s ich usadením sa v tesnom lietadle, no po čase pilot našiel riešenie.

Napíšte procedúru, ktorá bude pracovať s dvoma zásobníkmi len pomocou procedúr ***vyber*** *a* ***vlož*** *tak, aby správne zaradila nového cestujúceho (t.j. aby sa dostal na miesto, ktoré mu prináleží podľa kilometra, na ktorom bude vystupovať).*



Ako to bude programe? Asi



vyzerať v takto:

```
Procedure Zastavka_s_nastupovanim;

begin
  Citaj_noveho_cestujuceho;
  while     Na_vrchu_zasobnika_je_ziak_vystupujuci
            _na_blizsom_kilometri_ako_pristupujuci   do begin
    Vyber_vrchneho_z_lietadla;
    Vloz_ho_do_haly;
  end;
  Vloz_noveho_do_lietadla;
  while Hala_nie_je_prazdna do begin
    Vezmi_vrchneho_z_haly;
    Vloz_ho_do_lietadla;
  end;
end;
```

*Upravte celý program tak, aby bol schopný pridávať, vyberať a zaraďovať nových cestujúcich do zásobníka len pomocou procedúr **vloz** a **vyber**. K dispozícii máte len dva zásobníky.*
Zabezpečte možnosti:

1. ukáž vrchného,
2. vyhoď vrchného,
3. nový cestujúci,
4. koniec.

Cvičenie:

1. Majme tri šnúrky a n očíslovaných guličiek. Na začiatku sú náhodne navlečené na šnúrke č.1. Cieľom je dostať ich na šnúrku č. 2 tak, aby boli uložené od 1 po n. Pri prenášaní možno požívať tretiu šnúrku.

a. Napíšte program, ktorý bude prekladať podľa vašich príkazov.

b. Napíšte program, ktorý sám nájde riešenie a vypíše postupnosť vedúcu k správnemu riešeniu.

Ako šnúrky použite zásobníky.

2. Na tanečný banket bol pozvaný neznámy počet ľudí. Podmienkou bolo, že žiaden z nich nesmie prísť s partnerom. Úlohou uvádzača bolo popáriť navzájom mužov a ženy. Aby mal v práci poriadok a nemusel sa trápiť tým, že pán pri klavíri nemá partnerku a panej pod stolom sa tiež žiaden neušiel, rozhodol sa prichádzajúcich "zatvárať" do priľahlého salónu.

Ak príde muž a v salóne je žena, vezme ju zo salónu a pridá k mužovi.

Ak príde muž a v salóne je muž, usadí ho do salónu.

Keď potom príde žena, jedného muža zo salónu vyberie a pridelí jej ho atď.

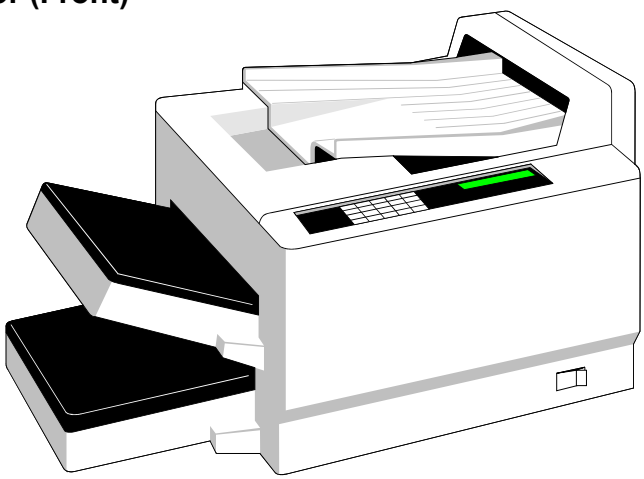
V prípade, že nie sú ženy, mužov v salóne pribúda, v prípade, že nie sú muži, v salóne je klebetnícky krúžok žien.

a. Napíšte program, ktorý podľa zadávaných príchodzích (ako budú vstupovať) rozhodne, čo s nimi (zavrieť do salónu, vybrať zo salónu). V prípade požiadavky vypíše obsah salóna.

*b. Napíšte program, ktorý pre zadaný reťazec (zložený z **M** a **Z**) vypíše, či muži a ženy budú v pároch.*

V oboch prípadoch používajte zásobník!

2.2 Mysliaci operátor (Front)



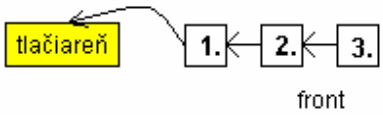
V rámci modernizovania zastaralých technických a technologických prostriedkov priviezli do Výpočtového strediska ústavu pre správu a spracúvanie nových technológií tristopätnásť nových strojov a prístrojov.

Ako to už býva vo veľkých spoločnostiach, tretina sa rozkradla, druhá sa odložila do skladu a zvyšok sa porozdeľoval všade, len nie do VS. Vedúci, pán Maximilián Chrobák, si vo vedení nakoniec vydupal aspoň jednu farebnú laserovú tlačiareň.

Tá sa pripojila do siete, takže ju teoreticky mohli využívať všetci zamestnanci. Teoreticky... Prax bola taká, že ak niekto z pracovníkov chcel vytlačiť správu pre najvyššieho administrátora, vyúčtovanie leteniek a poslednej služobnej cesty alebo len farebné fotografie z výročnej rodinnej slávnosti a tlačiareň bola zaneprázdnená (t.j. tlačila už niečo iné), celý kolos zamrzol. Spadol systém, sieť sa preťažila a často sa stratila aj celodenná práca.

Pán Chrobák sa preto rozhodol prerobiť celý operačný systém a upraviť ho tak, aby vždy ďalšia zásielka na tlačiareň čakala, kým sa vybaví posledná. Pokiaľ by úloh bolo viac, zoradia sa za sebou - do radu.

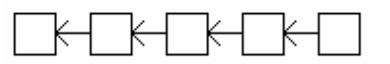
Kým bude úloha, ktorá je na prvom mieste v rade, spracúvaná, ostatné nebudú zmrazovať sieť, produkovať pracovníkom VS šediny, ale len pokojne, ticho čakať.



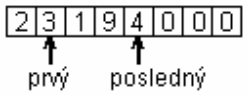
Rad, do ktorého sa budú úlohy stavať, sa v informatickej teórii nazýva FRONT alebo FIFO (first in, first out - prvý dnu, prvý von). Znamená to, že úloha, ktorá sa do radu dostane ako prvá, bude aj ako prvá vyriešená.

Napíšte program, ktorý bude simulovať činnosť tlačiarne a jej frontu (t.j. príchod úlohy, vyriešenie úlohy z radu, odoslanie správy o jej ukončení.

1. pridaj úlohu na koniec radu,
2. rieš úlohu - tlač,
3. koniec pracovnej doby).



Pri fronte, podobne ako pri zásobníku, máme prístup len k prvku, ktorý je na začiatku, a vieme pridávať na koniec.



Front opäť môžeme implementovať pomocou poľa a pamätať si (pokiaľ nechceme v každom kroku posúvať celé pole) index políčka (úlohy), ktorú spracúvame, a koniec frontu.

O práci s frontom pomocou poľa by sa dali vymyslieť a aj boli vymyslené mnohé teórie, no faktom napriek tomu zostáva, že pri poli sme opäť pamäťovo obmedzení jeho rozsahom.

Najideálnejšou možnosťou je zasa použitie dynamických štruktúr. Jedna položka frontu môže vyzerať asi takto:

```
Type Ppolozka = ^Polozka;
     Polozka = record
           Udaj: string;      {údaj zaradený do frontu}
           Dalsi: Ppolozka;
     end;
```

Pridávanie bude vyzerať asi takto:

```
Procedure Pridaj;
var Pomoc: Ppolozka;

begin
  if Zaciatok_frontu = NIL then begin
    New(Zaciatok_frontu);
    Nacitaj(Zaciatok_frontu^.Udaj);
    Zaciatok_frontu^.Dalsi:= NIL;
    Koniec_frontu:= Zaciatok_frontu;
  end else begin
    New (Pomoc);
    Nacitaj (Pomoc^. Udaj);
    Pomoc^.Dalsi:= NIL;
    Koniec_frontu^.Dalsi:= Pomoc;
    Koniec_frontu:= Pomoc;
  end;
end;
```

A odstránenie položky po vykonaní úlohy:

```
Procedure Zrus;
var Pomoc: Ppolozka;

begin
 if Zaciatok_frontu > NIL then begin
    Pomoc:= Zaciatok_frontu^;
    Zaciatok_frontu:= Zaciatok_frontu^.dalsi;
    Dispose (Pomoc);
 end else Vypis ('Niet co vyhodit');
end;
```

Zdanlivo nezmyselnú prácu s premennou ***Pomoc*** využívame na uvoľnenie pamäti.

Vráťme sa ale späť do Výpočtového strediska. Operačný systém, ktorý spracúval úlohy usporiadané vo fronte chvíľu vyhovoval.

Ale po čase, keď pán vedúci potreboval v špičke vytlačiť prospekty pre zahraničných spolupracovníkov čakajúcich v kancelárii, a musel čakať vyše troch hodín, zmenil mienku.

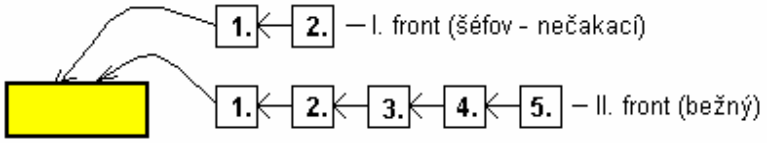
- Toto je neúnosné! Tu sa nedá pracovať, - búchal na druhý deň päsťou do stola a nadával svojmu operátorovi. - To sa musí zmeniť. Ja som tu šéf a keď niečo chcem, ostatní musia čakať!

- Hahaha! Vysvetlite to tej tlačiarni, - podpichol ho operátor.

- Na to ste tu vy. Máte na to dvadsaťštyri hodín, - odvrkol vedúci a ešte raz poslal uvoľnené vášne do stola.

Vyriešte tento problém.

Operátor, celý zúfalý zo seba (že nevedel držať jazyk za zubami) aj zo svojho vedúceho (že je taký nervózny) dumal, dumal, až napokon došiel k záveru, že treba vytvoriť fronty dva. Jeden bude prioritný - kým v ňom budú nejaké úlohy, budú sa tlačiť tie a všetko ostatné bude čakať. A bude ho obhospodarovať len a len veľký vedúci šéf.



Druhý, bežný front sa bude spracúvať len v prípade, ak prvý bude prázdny. Ak sa medzitým dostane zasa nejaká úloha do prvého, príde na rad hneď, nebude čakať na vyprázdnenie druhého frontu.

Napíšte procedúru, ktorá bude simulovať činnosť vylepšeného operačného systému tlačiarne (fronty nebudú dva, ale tri - aj operátor chce vlastné priority).

1. *úloha od šéfa,*
2. *úloha od operátora,*
3. *úloha od neprivilegovaných,*
4. *spracovanie úlohy,*
5. *koniec.*

Pozn.: Občas, pri niektorých zložitejších úlohách, sa používa aj front s položkami ukazujúcimi oboma smermi.

2.3 Alpskí dediči (Rekurzia)

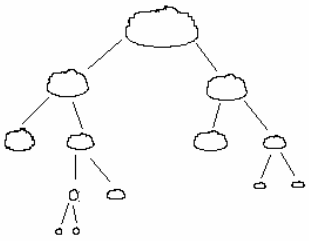
Vysoko v Alpách, kde lesy prechádzajú do kosodrevinatého porastu a kde teplota nestúpa nad bod mrazu, žije geografmi neobjavený, vedcami neprebádaný a človekológmi nepreskúmaný kmeň divých Kablostov. Sú to ľudia podobní bežným, Zem obývajúcim tvorom, vedia rozmýšľať (ich inteligencia dokonca predčí úroveň niektorých politikov), sú priateľskí, ale nejavia záujem o svet mimo svojho najbližšieho okolia. Žijú usporiadaným životom - ráno vstávajú spolu so svišťami a spávať chodia zároveň so zamŕzaním vodopádu vedľa osady. Hlboko v údolí, kam sa turista dostane len ako parašutista, sa živia mäsom divých zvierat, podsnehovými bylinami horských strání a čokoládou.

Všetci obyvatelia sa držia zásady, že prirodzený prírastok obyvateľstva sa musí udržiavať okolo 0.00 - keby sa počet ľudí v osade zvyšoval, postupne by ubúdalo potravy a hlad nie je to, čo by sa náčelníkovi Mururovi páčilo. Preto smie mať každá rodina najviac dve deti (jedno za otecka, druhé za mamičku). Pravidlá sú tvrdé, no život v horách nie je o nič mäkší. Kablostovia si to uvedomujú, a preto prikázanie veľkého Mururu dobrovoľne dodržiavajú.

Už niekoľko storočí sa v osade nič ohromujúce nestalo, no naraz šaman dospel k záveru, že najbohatší obyvateľ - starý Tasasamataro - je nevyliečiteľne chorý a bolo by načim rozdeliť jeho majetok medzi potomkov.

Je známe, že pri ťažkom živote Kablostov sa žiaden z nich nedožije veku nad 120 rokov, preto súčasne žije v osade najviac päť generácií potomkov jedného starca.

Tasasamataro sa teda rozhodol rozdeliť svoj obrovský majetok - 2 483 koží, 16 921 horských jarabíc, 518 litrov domácej lavórovice a 2 003 platničiek čokolády - spravodlivo medzi potomkov.



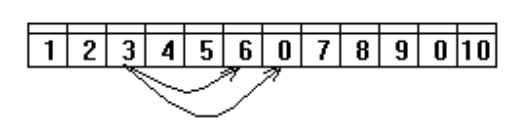
Uvažoval dlho, veľmi dlho, ako by mohol byť čo najspravodlivejší, ako sa odvďačiť svojim deťom a deťom svojich detí za to, že mu neustálym otravovaním znepríjemňovali posledných 72 rokov života, až došiel na zaujímavú myšlienku.

Rozdelí svoj majetok medzi svoje dve deti. A oni polovicu z toho, čo dostanú rozdelia ďalej - medzi svoje deti. A tie si zasa polovicu nechajú a zvyšok rozdelia. A tak až do piateho kolena.

A potomkovia sa hneď začali deliť a dohadovať, čo s dedičstvom. Starý Tasasamataro sa na nich len díval a šibalsky sa uškŕňal popod svoje dlhánske fúziská.

Napíšte procedúru, ktorá rozdelí majetok medzi Tasasamatarových potomkov. (Počítajte aj s tým, že nie každý má dve deti.)

Najlepšie by bolo uložiť všetkých potomkov do poľa tak, že v prvom políčku bude Tasasamataro, v druhom a treťom jeho deti, vo štvrtom a piatom deti druhého atď. Čiže potomok bude uložený v políčku ***(2 * predok)*** a ***(2 * predok + 1)***. Ak potomok existovať nebude, zostane políčko prázdne.



Povkladajme teda potomkov veľkého boháča do poľa záznamov...

... a poďme rozdeľovať majetok.

Povedzme, že celý majetok má cenu 200 000 tabličiek čokolády. **2** a **3** dostanú po polovici, teda po 100 000. Ale obaja majú potomkov, tak im musia dať polovicu z toho, čo majú. **4** a **5** - potomkovia **2** - dostanú spolu 50 000, teda 25 000 pre každého. **6** - ako jediný potomok **3** dostane 50 000. Ale zasa sa to musí deliť ďalej.

A našou úlohou je nekomplikovať si život neustálym pamätaním si kto, komu, kedy a koľko dá, ale napísať program, ktorý majetok rozdelí za nás.

Ako bude vyzerať suma, keď sa dostane k niektorému potomkovi?

```
Procedure Delenie (Suma: word);

begin
  if Existuju_dvaja_potomkovia then begin
    Vezmi_si_polku;
    Daj_prvemu;
    Daj_druhemu;
  end else
    if Existuje_jeden_potomok then begin
      Vezmi_si_polku;
      Daj_prvemu;
    end else Vsetko_je_tvoje;
end;
```

Lenže ***Daj_prvemu***, ***Daj_druhemu*** urobí zasa to isté - rozdelí čokoládu ďalším potomkom a potomkom potomkov, a tak môžeme písať:

```
Procedure Delenie (Clovek: integer, Suma: word);

begin
  if Existuju_dvaja_potomkovia then begin
    Vezmi_si_polku;
    Delenie (Clovek*2, Suma div 4);
    Delenie (Clovek*2+1, Suma div 4);
  end else
    if Existuje_jeden_potomok then begin
```

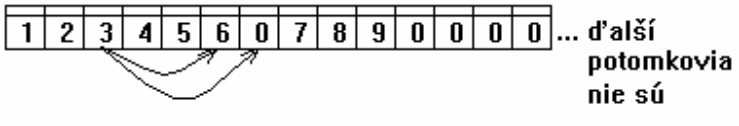
```
      Vezmi_si_polku;
      Delenie (Clovek*2, Suma div 2);
    end else Vsetko_je_tvoje;
end;
```

pretože pri každom potomkovi sa proces opakuje.

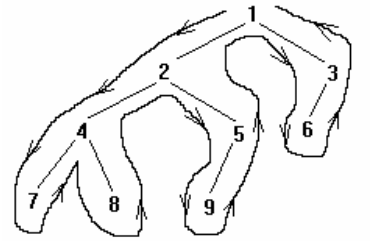
V procedúre voláme tú istú procedúru. Nie je to chyba - je to **rekurzia**. (prostriedok veľmi zjednodušujúci prácu so zložitými problémami). Predstavte si, že by ste celé delenie museli naprogramovať iným spôsobom. Ako? Dá sa, ale je to neporovnateľne zložitejšie a náročnejšie ako použiť rekurzívne volanie procedúry.

Ak v procedúre zavoláme ju samu, vykoná sa druhé volanie procedúry a vrátime sa na miesto, odkiaľ sme procedúru volali.

Ako bude vyzerať delenie povedzme pre tri generácie?



Graficky vyzerá práca procedúry ***Delenie*** asi takto.



Ak budeme sledovať algoritmus, tak:
- v prvom kroku má **1.** - 200 000,
- polovicu si nechá a štvrtinu pošle **2.**.

  Otvára sa procedúra ***Delenie*** znova,
  - v prvom kroku má **2.** - 50 000,
  - polovicu si nechá a štvrtinu pošle **4.**.

    Ďalšie otvorenie ***Delenia***,
    - v prvom kroku má **4.** - 6 250,
    - polovicu si nechá a 1 562 pošle **7.**.

      Otvorí sa ***Delenie***,
      - v prvom kroku má **7.** - 1 562,
      - potomkov nemá, nechá si všetko a procedúra končí, vracia sa späť ku **4.**
    - a pokračuje.
    - druhú štvrtinu pošle **8.**.

      Otvorí sa ***Delenie*** a
      - v prvom kroku má **8.** - 1 562,
      - potomkov nemá - procedúra končí a ide o úroveň späť - ku **4.**,
    - tu procedúra tiež končí a vracia sa ku **2.**,
  - tá poslednú štvrtinu pošle **5.**,

- otvorí sa delenie pre **5.**,
- **5.** si polovicu nechá,
- a druhú, keďže má len jedného potomka, pošle **9.**,
  - otvorí sa delenie pre **9.**,
  - potomkov niet, nechá si všetko a vráti sa o úroveň späť,
- **5.** podelila všetko, procedúra nemá čo ďalej konať - skončí a ide
- o ďalšiu úroveň späť

atď.

Toľko by na ozrejmenie práce rekurzie mohlo stačiť.

Všimnite si, že procedúra vždy skončí práve vtedy, keď dotyčný už nemá žiadneho potomka. Tento prípad - nazývaný **triviálny** - nesmie chýbať v žiadnej rekurzívnej procedúre. Inak by totiž vytváranie stále nových procedúr pokračovalo donekonečna a program by bol po zaplnení dostupnej pamäte násilne ukončený.

Technicky sa pre každú novovolanú procedúru berie časť z voľnej pamäte a procedúry sa ukladajú akoby do zásobníka. Keď procedúra skončí, časť zásobníka sa uvoľní a pokračuje sa ďalej.

Pozornejší si určite všimli, že v našom prípade ešte zostane polovica majetku aj Tasasamatarovi. Upravte procedúru tak, aby mu nezostalo nič. (Načo je človeku po smrti hmotný majetok?)

Tasasamataro ležiac v spacáku a hľadiac stareckými očami dookola pozoroval svojich drahých dedičov a v duchu sa zlostil:

- Takéto stvory som to ja vychoval. Ešte si tu polihujem, teším sa z teplých slnečných lúčov, a oni, darebáci, si už delia moje movitosti. Tak to teda nie!

I zodvihol sa, vybalil svoje storočné telo zo spacáka, ponaťahoval si údy a išiel sa prebehnúť na miestne futbalové ihrisko. Potencionálni budúci dediči vytreštili oči, spadla im sánka a len neveriaco čumeli na svojho praprazdravého prapraprapredka. A nechápali...

A Tasasamataro sa tešil zdravému životu, spomínal na časy, keď chcel svoj majetok rozdeliť medzi takých ničomníkov, a hľadal človeka, ktorému by ťažkou prácou nahonobené hmotné statky mohol venovať.

Cvičenie:

1. S použitím jedinej premennej sčítajte dve čísla.

Možno sa vám príklad zdá nezmyselný a neriešiteľný, ale s použitím rekurzie je úplne jednoduchý a práve na ňom možno pozorovať silu rekurzívnych procedúr:

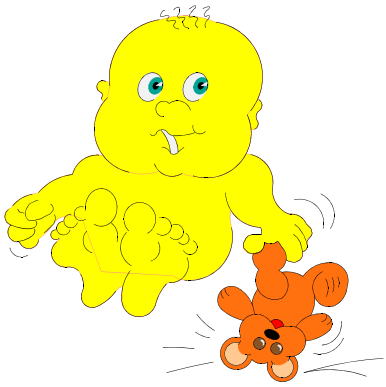
```
Program Sucet_s_jednou_premennou;
var a:integer;

Procedure Sucet(var a:integer);

begin
 a:=a-1;
 if a>0 then Sucet(a)
```

```
        else begin
         Write(' Zadaj druhe cislo ');
         ReadLn(a);
        end;
 a:=a+1;
end;


begin
 Write(' Zadaj prve cislo ');
 ReadLn(a);
 Sucet(a);
 writeln(a);
end.
```

2. Zamyslite sa nad programom a napíšte druhý, v ktorom budete rovnakým spôsobom počítať rozdiel kladných čísel.

3. Upravte program (za rovnakých podmienok), aby dokázal sčitovať kladné aj záporné čísla.

2.4 Geniálny Divoch (Backtracking)



V dávnej minulosti, ďaleko na juhu Európy, kde sneh poznajú len vďaka vysokým štítom hôr, kde Slnko svieti väčšinu dňa a väčšinu roka, sa v jednej kráľovskej rodine narodil syn. Chlapec veľký, silný, čulý a od svojho príchodu na svet veľmi zvedavý. Rodičia ho kvôli týmto pre novorodencov nezvyklým vlastnostiam nazvali Divochom.

Divoch rástol a verný svojmu menu vystrájal a divočil. Najprv len vo svojom krídle zámku, neskôr na celom kráľovskom dvore a po desiatich rokoch sa mu snažil vyhýbať každý človek z krajiny. Nebolo to preto, že by niekomu ubližoval alebo ho dokonca týral. Divoch bol skutočným divochom len v premýšľaní, myšlienkových pochodoch a pri riešení problémov. Každý, s kým sa dal do reči, sa postupne začal hanbiť a pripadať si pri tom desaťročnom nedochôdčati ako tupec.

A keďže sa s Divochom nechcel nikto baviť a každý sa mu vyhýbal, chudákovi chlapcovi nezostávalo iné, len aby sa staral o zábavu sám.

Ako áno, ako nie, dostala sa mu do rúk šachovnica s figúrkami, pergameny a hlinené doštičky s pravidlami a jednoduchými úlohami.

Jedna z nich napríklad znela:

Ako prejsť koňom po šachovnici tak, aby skočil na každé pole práve raz? (Riešte najprv len pre šachovnicu 5x5.)

Úloha sa na prvý pohľad zdá neriešiteľnou, no na pohľad druhý sa začína črtať riešenie. Budeme postupovať podľa náčrtku. Ak jazdec môže skočiť na pole v smere I., skočí, ak nie, pokúsi sa skočiť na pole II.. Ak ani to nie je možné (pole je mimo šachovnice alebo ho už navštívil), vyskúša III., IV. atď. Ak nemôže skočiť ani na VIII., dostal sa do slepej uličky a musí sa vrátiť.

	8		1	
7				2
		1		
6				3
	5		4	

V tom prípade vrátime jazdca o krok späť a z poľa, na ktoré sme sa vrátili, vyberieme ďalší smer (ak sme predtým skočili do smeru III., teraz sa vrátime a otestujeme smer IV.).

Takýmto spôsobom jazdec buď preskáče celú šachovnicu, alebo zistí, že to nejde (ak otestuje všetky možnosti a nájdu sa polia, na ktorých nebol).

1. Začneme napr. v rohu šachovnice (toto pole sa očísluje ako 1).
2. Jazdec skočí v smere I.
3. Políčko, na ktoré sa dostaneme označíme ako 2.
4. Skúmame smer I. - jazdec skočiť nemôže (je mimo šachovnice). Rovnako smer II. a smer III.
5. Skočíme v smere IV. a políčko označíme ako 3.
6. Ďalej (smery I., I., III., IV. nevyhovujú) skočíme v smere V..
7. Pokračujeme, kým sa máme kam pohnúť.

			2	7
	15	6	11	
		1	8	3
14	5	10		12
		13	4	9

8. Ak sa dostaneme do stavu, že niet kam pokračovať, vrátime sa o krok späť. Z políčka 14 sme sa naposledy pohli v smere I.. Bol nesprávny, skúmame smery II., III., ... VIII.. Ani jeden nevyhovuje - vrátime sa o pole nazad.

			2	7
		6	11	
		1	8	3
	5	10		12
		13	4	9

9. Z poľa 13 sme použili smer VII., skúmame VIII. Vyhovuje - posunieme v ňom koňa.

10. Pokračujeme, kým nie je poskákaná celá šachovnica.

Procedúra starajúca sa o skákanie by mohla vyzerať:

```
Procedure Skok(x,y:integer);

begin
 Cislo_skoku:= Cislo_skoku+1;  {globálna premenná
                               počítajúca skoky}
 Sachovnica[x,y]:= Cislo_skoku;       {označenie políčka, na
                               ktoré sa skočí}
 if Cislo_skoku<Sqr(Rozmer_sachovnice) then begin
                         {ak poskákaná celá šachovnica}
    if mozno(1| then Skok(x+1,y-2);
    if mozno(2) then Skok(x+2,y-1);
    if mozno(3) then Skok(x+2,y+1);
    if mozno(4) then Skok(x+1,y+2);
    if mozno(5) then Skok(x-1,y+2);
    if mozno(6) then Skok(x-2,y+1);
    if mozno(7) then Skok(x-2,y-1);
    if mozno(8) then Skok(x-1,y-2);
 end else Vypis_poskakanie;
end;
```

Volala by sa súradnicami miesta, na ktorom sa nachádza jazdec na začiatku.

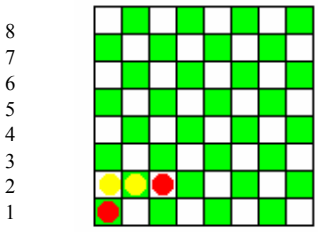
Metóda, ktorou sme riešili úlohu, sa nazýva ***backtracking*** a je to vlastne skúmanie všetkých možných stavov na všetkých možných úrovniach. Je to akoby **spätné prehľadávanie** - keď niekde neuspejeme, vrátime sa o krok späť a odtiaľ pokračujeme s novými skúsenosťami.

Ďalšia úloha:

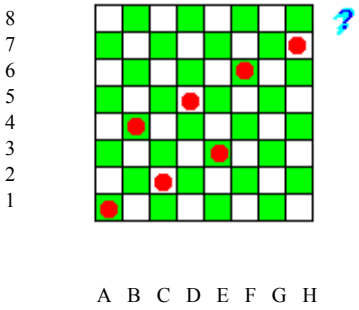
Rozložte osem dám na šachovnici tak, aby sa navzájom neohrozovali.

Ako na to?
Rovnakú otázku si položil aj Divoch a začal rozmýšľať. Prísť na systém je jednoduchšie ako len tak náhodne strieľať, ukladať, a potom kontrolovať.

Teda:



1. Prvú dámu uložíme do rohu.
2. Dámu v druhom rade tiež najprv uložíme na okraj šachovnice.
Tu však ohrozuje prvú dámu, preto ju posunieme o políčko ďalej. No opäť zistíme, že ohrozenie je akútne. Až keď je na treťom políčku (v šachovej terminológii C2), sú dámy neohrozujúce sa.
3. Rovnaký postup zopakujeme aj s dámami 3, 4, ... 7.



4. A prišli sme k problému s poslednou dámou...
Možno keby sme siedmu presunuli inam... Ale kam, keď sme už vyčerpali všetky možnosti?
... tak snáď šiestu, ... alebo až piatu?

Postupne by sme premiestňovali dámy asi takto: Ak dámu v skúmanom rade už nemáme kam uložiť (t.j. sme s ňou v poslednom stĺpci a ohrozenie pretrváva), položíme ju na začiatok šachovnice a vrátime sa o riadok (o dámu) späť a posunieme ju o políčko ďalej (ak sme na konci, ideme ešte o riadok nižšie atď.). Keď nájdeme pre niektorú dámu takú polohu, z ktorej nebude ohrozovať ostatné (na nižších riadkoch) a budeme pokračovať, až kým sa dámy neprestanú ohrozovať alebo nevyskúšame všetky možnosti ich rozloženia.

Divoch po troch dňoch bez jedla a spánku našiel 87 možností (polôh), ktoré vyhovujú zadaniu.

Napíšte program, ktorý usporiada dámy na šachovnici tak, aby sa navzájom neohrozovali.

Algoritmus bol už v podstate popísaný vyššie: nosnou časťou bude rekurzívna procedúra, ktorá bude mať na starosť buď vrátenie sa alebo pokračovanie rozkladania dám.

Napíšte program, ktorý nájde všetky dámske uloženia.

Po svojich úspechoch v šachu (Divoch sa neuspokojil s rozkladaním dám, ale na miestnom turnaji porazil všetkých renomovaných majstrov) sa mladý pánko rozhodol prejsť na zmysluplnejšiu činnosť a dal sa študovať (šťastný to človek).

Zaujímal sa najmä o mytológiu, geografiu, dejepis a alchýmiu.

Študoval mesiac, dva, pol roka a po roku už do vrecka strčil všetky múdre hlavy kráľovstva naraz.

Kdesi v tých múdrych knihách sa dočítal, že na neďalekom ostrove žije v obrovskom bludisku neporaziteľný netvor, ktorý dá svojim obetiam vždy pred zožratím šancu poraziť ho v logických problémoch. A hovorí sa, že ak ho niekto skutočne porazí, netvor zmizne a rozplynie sa v hmle.

Ďalej sa v knihe písalo, že ho najprv treba v bludisku nájsť, a to sa dosiaľ múdrym odvážlivcom nepodarilo. Všetci zablúdili v labyrinte chodieb a stien.

Kdesi pod čiarou bola ešte poznámka, že hviezdy hovoria o tom, že netvora porazí len potomok veľkého Ramakusa, prvého kráľa južnej krajiny.

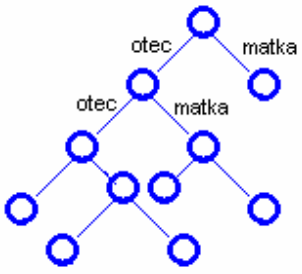
Divoch s otvorenými ústami čítal a čítal, s otvorenými očami sníval a sníval, a napokon začal aj rozmýšľať:

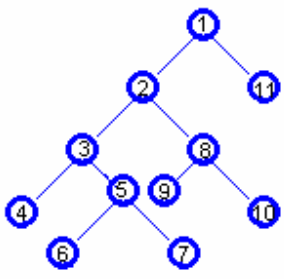
1. Musím byť potomkom Ramakusa,
2. Musím sa dostať na ostrov a nájsť netvora v jeho pelechu,
3. To už bude hračka. Určite som inteligentnejší ako nejaká hora šťachovitého mäsa v jaskyni.

A hneď začal svoje predsavzatia plniť. Ako prvý si zohnal rodokmeň. Ten bol však taký veľký, že po rozprestretí zabral celú podlahu prijímacej haly zámku. Divoch teda opäť musel nájsť spôsob, ako to celé prehľadať.

Vymyslel si taký systém, že pôjde od svojho otca na jeho otca atď. Keď dôjde na posledného otca, vráti sa k jeho synovi a pozrie sa na matku, jej otca, otca jej otca atď.

Teda v takomto poradí:





Napíšte program, ktorý pre zadaný rodokmeň zistí, či sa Ramakus v ňom nachádza alebo nie. Na naplnenie použite najjednoduchší spôsob.

Nedávno sme sa dozvedeli, ako s výhodou možno vytvárať nové prvky pomocou procedúry **new** a smerníkov. Tento príklad je na tento štýl práce ako stvorený. Pre reprezentáciu jednotlivých položiek v našom rodokmeni (budeme ho nazývať strom) bude najvhodnejšie použiť štruktúru **record**.

V podstate ide o úplne obyčajný binárny strom (štruktúru, v ktorej má prvok dvoch nasledovníkov), ktorý budeme prehľadávať systémom otec, pravý syn, ľavý syn.

Vytvoriť a naplniť binárny strom, by nemalo robiť problémy. Jedna jeho položka bude definovaná:

```
Type   Ppolozka = ^Polozka;
       Polozka = record
              Meno: string;
              Otec, Matka: Ppolozka;
       end;
```

a procedúra na prehľadávanie:

```
Procedure Hladaj (Otec: Ppolozka);

begin
 if Otec <> NIL then
  if Otec^.meno = 'Ramakus' then Vypis ('Mam')
  else begin
     Hladaj (Otec^.Otec);
     Hladaj (Otec^.Matka);
  end;
End;
```

Táto procedúra je úplne najjednoduchšia a natoľko kostrbatá, že bude prehľadávať úplne celý strom, t.j. aj vtedy, keď už Ramakusa nájde.

Pozn: pre podrobnejšie vysvetlenie pozri **Binárne stromy**.

*Upravte procedúru **Hladaj** na funkciu, ktorá vráti hodnotu FALSE, keď Ramakusa niet, a TRUE, keď sa v rodokmeni nachádza. Keď ho nájde, nebude v prehľadávaní pokračovať.*

Či už vaše prehľadávanie dopadlo tak alebo onak, Divoch sa pri objavovaní svojich predkov dopracoval k pozitívnemu výsledku - Ramakusa našiel.

Povedal si teda, že hviezdam sa protiviť netreba, a začal organizovať výpravu. Nechal si naložiť 30 lodí vodou a potravinami, naverboval niekoľko stoviek námorníkov, vzal si šachovnicu a prehľadávanie ostrovov sa mohlo začať.

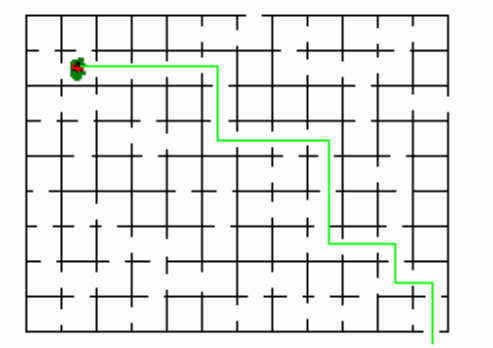
Po dvoch rokoch sa na obzore konečne objavil pahorok, ktorý súhlasil s náčrtkom v starej knihe hovoriacej o netvorovi.

Divoch nechal lode zakotviť a začal rozmýšľať, ako sa v neznámom bludisku zorientovať.

Za zábradlím sa preháňali ryby, v podpalubí šramotili potkany a nad loďou krúžili čajky. A práve tie mu vnukli myšlienku: najprv treba zistiť, ako vlastne vyzerá pôdorys bludiska, potom nájsť správnu cestu, no a napokon sa len riadiť zistenou trasou.

Pôdorys umelého bludiska teda už nebol problém zistiť. Divoch dal nachytať niekoľko stoviek čajok, priviazal im na chrbty sieť a zo vzduchu sa mu podarilo nakresliť plán labyrintu so všetkými priechodmi, múrmi a stenami.

Teraz už bolo treba len nájsť najkratšiu (alebo aspoň nejakú) cestu, aby sa dostal k oblude spiacej kdesi v útrobách.



Ako postupovať?

1. Vstúpiť do bludiska.
2. Ak možno - ísť dopredu,
   ak nie - doľava,
   ak nie - doprava,
   ak nie - vrátiť sa.
3. S dodržaním tejto postupnosti sa dá dostať až k netvorovi - ak cesta naozaj existuje.

Napíšte program, ktorý vám v zadanom bludisku nájde cestu od vchodu až k netvorovi, ktorý kdesi drieme. (Miesto jeho odpočinku sa zadáva pri vstupe).

Bludisko možno reprezentovať viacerými spôsobmi, no najnázornejšia je matica, ktorá obsahuje všetky miestnosti a priechody medzi nimi.

```
Type Miestnost = record
           Hore, Dole, Doprava, Dlava: Boolean;
     end;

var Bludisko: array[1..x , 1..y] of Miestnost;
```

Procedúra na prehľadávanie bude vyzerať asi takto:

```
Procedure Hladaj (x, y: integer, od: Smer);

begin
 if Bludisko[x, y] = Netvor then Skoncit:= TRUE;
 if not Skoncit then begin
 Case od of
   Sprava:begin
           Hladaj(x-1,y,Sprava);
```

```
            Hladaj(x,y+1,Zhora);
            Hladaj(x,y-1,Zdola);
          end;
   Zlava: begin
            Hladaj(x+1,y,Zlava);
            Hladaj(x,y+1,Zhora);
            Hladaj(x,y-1,Zdola);
          end;
   Zhora: begin
            Hladaj(x,y+1,Zhora);
            Hladaj(x-1,y,SPrava);
            Hladaj(x+1,y,Zlava);
          end;
   Zdola: begin
            Hladaj(x,y-1,Zdola);
            Hladaj(x-1,y,SPrava);
            Hladaj(x+1,y,Zlava);
          end;
  end else Vypis (x, y);
end;
```

... a čo robí táto divočina? Presne to, čo popisoval algoritmus vyššie. Pri príchode z ľubovoľného smeru zisťuje možnosti pohybu do zvyšných troch smerov. Pokiaľ by sa nerozlišoval smer, z ktorého sme prišli, mohlo by sa stať, že by sme sa zasekli na jednom mieste. Prečo?

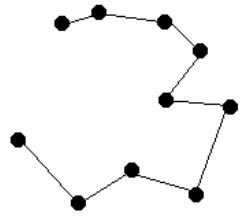
A prečo ***Vypis***? Vypíše cestu od netvora po východ. Posledné políčko, na ktorom je ešte ***Skoncit*** FALSE je to, kde sa rozvaľuje netvor - vypíše sa. Procedúra skončí a vráti sa na pole, z ktorého sme sa k netvorovi dostali. Keďže ***Skoncit*** je TRUE, opäť ho vypíše a ide o úroveň nižšie atď. Až von.

*Upravte procedúru **Hladaj** tak, aby sa našla najkratšia cesta.*

To znamená nájsť všetky cesty a tú najkratšiu si zapamätať.

Divoch našiel cestu, dostal sa do príbytku netvora a zistil, že prišiel neskoro. Netvora niekoľko mesiacov pred ním porazil miestny Hlúpy Jano, ktorý si teraz užíval s miestnou princeznou. Divochovi zostali len dračie kosti a nepríjemný pocit, že prišiel neskoro.

Unavený, naštvaný a pokorený sa vrátil domov a rozhodol sa vybudovať konečne v kráľovstve svojho otca cestu. Takú, ktorá by spojila všetkých 10 veľkých miest a vyšla pritom čo najlacnejšie.



Mal teda dve podmienky:

1. Aby sa z ľubovoľného mesta vedel dostať do iného po ceste.
2. Aby celá sústava 9 spojníc miest vyšla čo najlacnejšie.

Napíšte program, ktorý pre dané vzdialenosti miest určí najlacnejšiu (t.j. najkratšiu) sústavu cestnej siete. (Vstupom budú vzdialenosti miest, výstupom poradie, v akom treba cestu medzi nimi vybudovať.)

Pozn.: Vzájomné vzdialenosti miest vložte do matice.

Divochovi sa opäť podaril husársky kúsok, vybudoval, čo si predsavzal, a bolo to také dobré, že jeho dielo chválili všetky okolité národy, chodili k nemu po rady, odporúčania, vedomosti - a to mal ešte len okolo 17 pozemských rokov.

Využíval svoju múdrosť, radil ostatným a pomaly sa prestal zlostiť aj na svoju zbytočnú cestu za netvorom.

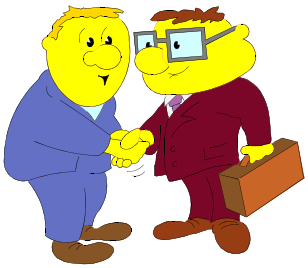
Cvičenie:

1. Napíšte program, ktorý bude schopný určiť najkratšiu trasu z jedného slovenského mesta do iného.
(Vzdialenosti je najpraktickejšie evidovať v matici a ešte praktickejšie uložiť ju na disk.)

2. Africká savana je plná nástrah a nebezpečenstiev. cesty sú skôr výnimkou ako pravidlom. Niektoré oblasti sú úplne odrezané od sveta. napíšte program, ktorý zistí pre zadanú maticu so spojnicami miest počet oblastí.
(Návrh: Ak je medzi mestami spojnica, je v matici reprezentovaná ako TRUE, ak nie FALSE. Hľadajte všetky spojenia zo všetkých miest).

3. V Číne pracujú poštári na veľkých vzdialenostiach, a preto sa vždy snažia roznášať poštu tak, aby nemuseli ísť po tej istej trase dva razy. Napíšte program, ktorý pre zadané mesto (s križovatkami a ich spojnicami) zistí, či ho možno prejsť tak, aby sa žiadna časť trasy neopakovala.
***Pozn.:** Ide o tzv. Eulerovský ťah.*

2.5 Slávybažný profesor (Binárne stromy)



Profesor Konôpka, učiteľ dejepisu, dostal pozvánku na Stredoeurópsku konferenciu historikov.

Keďže pozvánka prišla (pravdepodobne zásluhou pôšt) len deň pred akciou, profesor začal ihneď mierne panikáriť a zhromažďovať časti odevu, v ktorých sa bude prezentovať na ďalší deň.

S oblekom a bielou košeľou neboli problémy, profesor ich mal na sebe. Kravatu po prehádzaní všetkých skríň objavil na stoličke uprostred izby.

Horšie to bolo s obuvou. Botasky, v ktorých normálne chodil do práce, boli už značne obnosené. Po dlhom premýšľaní si však spomenul, že v pivnici má sviatočné topánky, v ktorých promoval. Prepletúc sa pomedzi pavučiny, staré noviny, prázdne fľaše a plné sudy ich našiel. Keď z jednej vyhodil myšiaka a z druhej pavúčiu rodinku, zistil, že sú celkom vyhovujúce. Po dlhšom leštení vyzerali naozaj ako zo škatuľky. Zostalo už len napísať prejav či prednášku...

Pri poslednej vete zastihlo profesora ráno. Zívajúc opustil svoj domov, zívajúc nastúpil do lietadla a zívajúc v ňom aj zaspal. Po prebudení už boli v dohľade veže konferenčného mestečka. Profesor si pripomenul hlavné body prednášky, vystúpil a ponáhľal sa na miesto určenia.

Konferencia bola poučná a zaujímavá, prednášky na slušnej úrovni a Konôpka bol spokojný.

Po namáhavom dni, ako to už býva, nasledoval spoločenský večierok, na ktorom sa pretriasali dojmy z prežitého dňa.

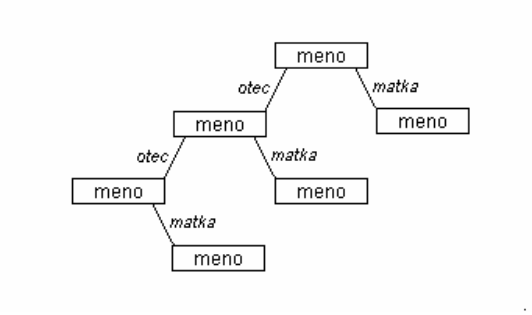
Všetko bolo v poriadku, až kým sa nezačal preberať pôvod účastníkov. Všetci sa chválili svojimi predkami, rozoberali minulosť svojho rodu snáď až do desiateho prapredka. Len profesor Konôpka sa po niekoľkých všeobecných odpovediach vytratil. Bolo mu nepríjemné, že zo svojich predkov poznal len rodičov a starú mamu. Pri neustálej pracovnej vyťaženosti mu totiž nezostal čas na to, aby sa zaoberal sebou samým.

Hneď po návrate domov začal túto chybu naprávať. Zahrabal sa na mesiac do archívu a keď odtiaľ vyšiel, poznal všetkých svojich predkov hádam až po Samovu ríšu.

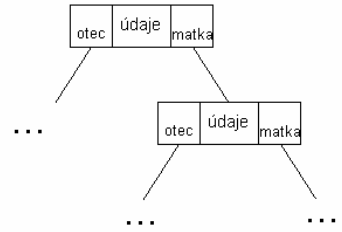
No profesorovi to stále nestačilo. Chcel mať maximálny prehľad, a preto sa rozhodol zveriť svoj rodokmeň počítaču.

Napíšte program, ktorý bude schopný zaznamenať všetkých priamych predkov (t.j. otec, matka - bez súrodencov). Bude schopný pridávať ich a vypísať rodičov pre zadaného človeka.

Ako vyzerá rodokmeň?



Zasa máme binárny strom, v ktorom si môžeme okrem mena pamätať aj dobu, v ktorej predkovia žili, ich majetok (bydlisko), povolanie atď.



Potom:

```
Type Rodic = ^Clovek;
     Clovek = record
            Meno, Majetok, Povolanie: string;
            Rok_narodenia, Rok_smrti: word;
            Otec: Rodic;
            Matka: Rodic;
     end;
```

Premenné ***Otec*** a ***Matka*** ukazujú na rodičov (otca a matku) človeka, ktorého meno obsahuje príslušný **record**. Títo ukazujú opäť na **record *človek*,** ktorý má tiež otca a matku:

Procedúra pre pridanie nových rodičov by mohla mať takúto schému:

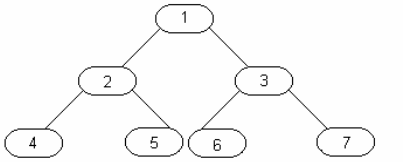
1. prečítaj meno,
2. nájdi ho v strome,
3. prečítaj rodičov,
4. zaraď ich do stromu.

S bodmi 1, 3 a 4 by nemali byť problémy.

Na nájdenie prvku v strome sa v mohutne prevažnej väčšine používajú rekurzívne algoritmy. Napr.:

```
Procedure Hladaj (Koren: Clovek);
                  {Koreň je ukazovateľ na človeka, medzi
                  ktorého predkami budeme hľadať}

begin
  if Koren. Meno = Hladany then
     Zapamataj_si_smernik_ktory_nanho_ukazuje
  else
```

```
      if Poznas_dalsich_potomkov then begin
            Hladaj (Koren.Otec);       {prehľadá
                                       otcových}
            Hladaj (Koren.Matka);      {a matkiných
                                       predkov}
      end;
end;
```



Vysvetlime si teraz činnosť procedúry. Majme rodokmeň. Máme zistiť, či sa v ňom nachádza meno šiesty (6.)

Smerník ukazuje nazačiatku na prvého. Postupujme podľa procedúry ***Hladaj***.

- Je ***Koreň.Meno*** hľadaný predok? Nie.
- Poznáme jeho predkov? Áno.
- Hľadáme v predkoch z otcovej strany, do procedúry vstupuje smerník na **2**.

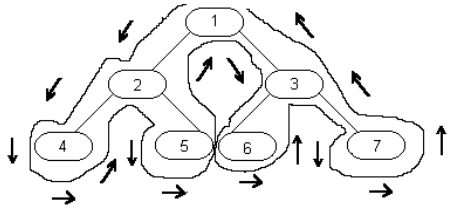
Ani **2** nie je hľadaný objekt, má potomkov, a tak prehľadávame stranu otca.

- Do procedúry vstupuje **4**. Nie je hľadaný, ani nemá potomkov, procedúra skončí a ďalej sa pokračuje z miesta, odkiaľ bola volaná.
- Prehľadávajú sa predkovia z matkinej strany od **2**, teda **5.**
- Smerník s hodnotou **5** nemá potomkov, procedúra skončí.
- Boli vykonané všetky príkazy pre **2**, skončí procedúra aj preň.
- Pre **1** sme skončili volaním otca, voláme teda matku.
- Do ďalšej procedúry vstupuje smerník na hodnotu **3**.
- Stále nejde o hľadaného, tak voláme procedúru pre jeho otca, t.j. pre **6**.
- Do procedúry vstupuje **6** a my s údivom sledujeme, že podmienka je splnená. Zapamätávame si smerník na hľadaného **6** a
- kvôli vyprázdneniu pamäti (haldy) nechávame prísť procedúru až do konca.
- Ukazovateľ na hľadaného predka máme zapamätaný, a preto možno vykonať body 3 a 4. (Môžeme ďalej pracovať - vypísať údaje, rodičov, testovať atď.).

Prípad, že by hľadaný v strome nebol, môžeme vyriešiť tak, že pred volaním procedúry vložíme do smerníka, ktorý má ukazovať na nájdený prvok v strome hodnotu **NIL**. Ak sa hľadaný nenájde, neprebehne ani zapamätávanie smerníka naň v prvej podmienke, teda aj po skončení neúspešného hľadania tu zostane kontrolovacie **NIL**.

Rovnako, za pomoci **NIL**, možno v rodokmeni reprezentovať aj neexistujúcich (neznámych) predkov.

Naše prehľadávanie, ak ho upravíme do grafickej podoby vyzerá asi takto:



Prehľadávali sme štýlom DOM (dieťa, otec, matka) alebo v odbornej terminológii - otec, ľavý syn, pravý syn. Otcom prvku sa tu nazýva ten prvok, ktorý je graficky nad ním. Synom je prvok, na ktorý otec ukazuje.

Prehľadávanie možno, samozrejme, vykonávať aj ďalšími spôsobmi (záleží to len od napísania - upravenia - hľadacej procedúry: DMO, ODM, OMD, MOD, MDO).

Napr. pre ODM by procedúra znela takto:

```
Procedure Hladaj (Koren: Clovek);

begin
 if Existuje_otec then Hladaj (Koren.Otec);
 if Koren.Meno = Hladany then
           Zapamataj_si_smernik_ktory_nanho_ukazuje;
 if Existuje_matka then Hladaj (Koren.Matka);
end;
```

Rovnakým spôsobom možno rodičov aj vypisovať. Nájdeme smerník ukazujúci na človeka a jeho rodičov jednoducho vypíšeme.

Profesor sa týmto programom a hľadaním svojich predkov zaoberal dostatočne dlho a možno povedať, že jeho úsilie malo byť napokon odmenené odtiaľ, odkiaľ to vôbec nečakal.

Parlament totiž odhlasoval zákon o navrátení majetku pôvodným vlastníkom...

A profesorovi sa strašne páčili majetky niektorých jeho susedov...

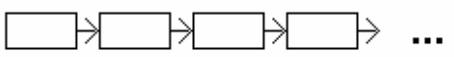
Napíšte procedúru, ktorá pre zadané meno majetku zistí, či ho niektorý z profesorových predkov nevlastnil.

Bohužiaľ, z tejto strany toho veľa nekvaplo, preto sa profesor rozhodol získať (vytlačiť) zoznam všetkých majetkov a pozemkov, ktoré kedy jeho rodine patrili.

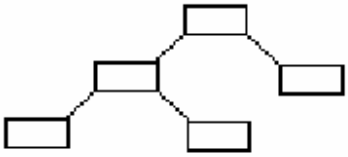
Napíšte procedúru, ktorá vypíše všetky majetky, ktoré profesorovej rodine patrili. Nech sa však žiaden z nich nevypíše viac ako raz.

Opäť prehľadávame celý strom a majetky ukladáme do osobitného zoznamu, ktorý vytvárame dynamicky (pomocou **new**). Vždy pred zaradením majetku do zoznamu skontrolujeme, či sa v ňom už nenachádza.

Zoznam majetkov sa dá robiť viacerými spôsobmi. Môže to byť úplne jednoduché zaraďovanie za sebou - do lineárneho zoznamu, kde každý prvok ukazuje len na svojho nasledovníka



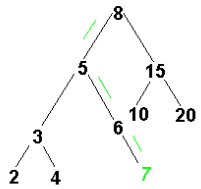
alebo



Pre tento strom platí, že ľavý nasledovník je menší a pravý väčší ako predchodca. Potom tu možno zoradiť prvky podľa abecedy a pri pridávaní, resp. kontrole, či už existuje, netreba kontrolovať celý zoznam, ale len postupovať:

1. Ak je nezaradený prvok menší, pozri vľavo
2. Ak je väčší, pozri vpravo,
3. Ak je rovný, skonči,
4. Ak nemáš s čím porovnať, tak ho zaraď (bude posledným - listom).

Napr.:
Zaraďme prvok 7:
7<8 - ideme doľava,
7>5 - doprava,
7>6 - doprava,
nie je žiaden prvok - ULOŽ.



Skúsme urobiť to isté pre 3:
3<8 - doľava,
3<5 - doľava,
3=3 - koniec.

Toto ukladanie má minimálne dve výhody:
1. Je o čosi rýchlejšie ako klasické - lineárne (vidieť to najmä pri veľkých množstvách údajov),
2. prvky možno vždy vypísať podľa abecedy.
Malou nevýhodou oproti lineárnemu ukladaniu je, že zaberá o čosi viac pamäti.

Profesor si začal nárokovať na majetok, ktorý vlastnili jeho predkovia, no neuspel. Veľkú väčšinu majetku totiž ešte v dobách rytierstva poprehrávali v kartách a poprepíjali, v dobe vojen a období temna boli povypaľované a zrovnané so zemou. A čo ešte zostalo, to sa v čase socializmu a ranného kapitalizmu rozkradlo.

Po týchto neveselých záveroch a zisteniach pán profesor Konôpka zanevrel aj na duševné aj na hmotné statky a oddal sa ďalšiemu skúmaniu svojho rodu.

Pre začiatok ho začali zaujímať prarodičia, ktorí žili za čias Mateja Korvína, potom Márie Terézie a Jozefa II. atď.

Napíšte procedúru, ktorá vypíše všetkých členov rodu, ktorí žili v zadanom období od-do čo len rok. (Žili v ňom aj vtedy, ak sa v hraničnom roku práve narodili alebo práve umreli.)

Pri neúnavnej práci zastihli pána profesora Vianoce. Vianoce, sviatky to pokoja, mu dali príležitosť zamyslieť sa nad sebou i nad svojím životom.

Každý večer profesor rozjímal niekoľko hodín.

Raz večer, pri jednom takom spytovaní si svedomia (asi v polovici), dospel, nevedno ako, až ku Guinessovej knihe rekordov. A začal rozmýšľať, ako by sa v nej mohol nechať zvečniť.

... A nápad prišiel. Najdlhšie žijúci človek mal 170 rokov. Ak bol niektorý z jeho predkov pred smrťou starší, dostal by sa do GKR. A spolu s ním aj meno objaviteľa - profesora Konôpku.

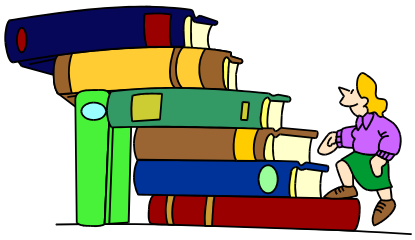
Napíšte procedúru, ktorá nájde najdlhšie žijúceho predka profesora Konôpku.

Táto akcia, napriek všetkým snahám, však profesorovi zápis do vytúženej knihy nepriniesla. Preto sa pokúsil zistiť najväčší vekový rozdiel medzi manželmi v rodine.

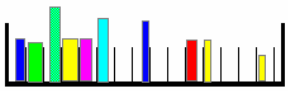
Napíšte procedúru, ktorá zistí najväčší vekový rozdiel medzi manželmi.

Tento pokus, rovnako ako tie predtým, stroskotal. Po krátkom uvažovaní pán profesor Konôpka predal počítač a kúpil si bicykel.

2.6 Poctivá knihovníčka (Triedenia)



Pani Moľová je vedúcou knižnice. Jej povinnosťou je starať sa o knihy, čitateľov, poriadok a manžela. Každý deň po pracovnej dobe musí zostať v knižnici a zoradiť knihy podľa evidenčných čísel. Deň čo deň je to úmorná práca, lebo stovky čitateľov rozhádžu všetko, čo sa len dá. Medzi odbornými medicínskymi príručkami možno nájsť prvý diel Winnetoua, druhý je vklinený medzi Kukučínove poviedky a Kurz číslicových počítačov a tretí robí spoločnosť Palculienke a leporelu Koho zjedla Mikimyška. Skrátka, nič nie je na svojom mieste a pani knihovníčka to musí dávať do poriadku, lebo nevďačná čitateľská verejnosť to tak vyžaduje. Musí byť poriadok a rýchly prístup ku knihám.



Všetky knihy sa zmestia do jednej špeciálnej police. (Špeciálna preto, lebo jednotlivé knihy sú pooddeľované úzkymi priehradkami.) Nemožno ich teda len tak jednoducho posúvať a tlačiť do police ďalšiu knihu, keď je plná a skracovať jej takto životnosť. (Dobre vymyslené!)

Čitatelia cez deň knihy poprehadzujú a pani Moľová ich večer musí ukladať. Má našťastie ešte jednu rovnakú prázdnu policu, kam knihy ukladá vzostupne (t.j. od najmenšej po najväčšiu).

V prvej polici, v ktorej je maximálny neporiadok, nájde najprv knihu s najnižším evidenčným číslom a uloží ju na začiatok prázdnej police. Potom vyhľadá knihu s najmenším evidenčným číslom zo zvyšných a vloží ju vedľa predošlej do naplňanej police. Pokračuje až do neskorého večera, kým nie je všetko pekne usporiadané.

Napíšte procedúru, ktorá bude simulovať činnosť pani knihovníčky. (Policu reprezentujte poľom, evidenčné čísla kníh generujte náhodne - môžu, samozrejme, existovať aj dve rovnaké knihy s rovnakým evidenčným číslom.)

```
Procedure Triedenie1;

begin
 for i:=1 to Pocet_knih do
     Nova_polica[i]:=Kniha_s_najmensim_ev_cislom;
 end;

Function Kniha_s_najmensim_ev_cislom: word;

begin
 i:=1;
 while Stara_polica[i]=0 do i:=i+1;
                  {číslom 0 sú kódované priehradky,
                  odkiaľ boli knihy už preložné}
```

```
    Najmensi_index:=i;
                  {zapamätáme si zatiaľ najmenšiu knihu}

    for j:=i+1 to Pocet_knih do

      if Stara_polica[j]<>0 then
       if Stara_polica[j]<Stara_polica[Najmensi_index]
           then
             Najmensi_index:=j;

  Kniha_s_najmensim_ev_cislom:=Stara_polica[Najmensi_index];
 Stara_polica[Najmensi_index]:=0;
                        {knihu vyberieme}
end;
```

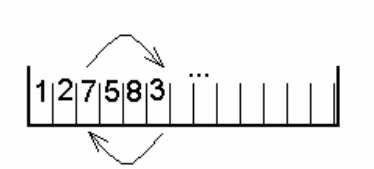
Chvíľu toto triedenie nádherne fungovalo, no v januári nasledujúceho roka, keď sa nakúpili ďalšie knihy, bola zaplnená aj druhá polica. Pani knihovníčka musela začať hľadať nové spôsoby usporadúvania.

Pokúsila sa to urobiť pomocou myšlienky z predchádzajúceho triedenia:

Rozhodla sa, že najprv vyhľadá knihu s najmenším evidenčným číslom a uloží ju na prvé miesto. Lenže, aby sa tam kniha zmestila, musí posunúť ostatné o priehradku ďalej. A to je jednak príliš namáhavé a jednak príliš zdĺhavé.

Po jednom obzvlášť dlhom večere sa so svojím problémom zverila manželovi - veľkému logikovi a lúštiteľovi krížoviek.

Ten jej poradil, aby len vymenila knihu s najmenším evidenčným číslom a knihu, na ktorej miesto ju chce uložiť:



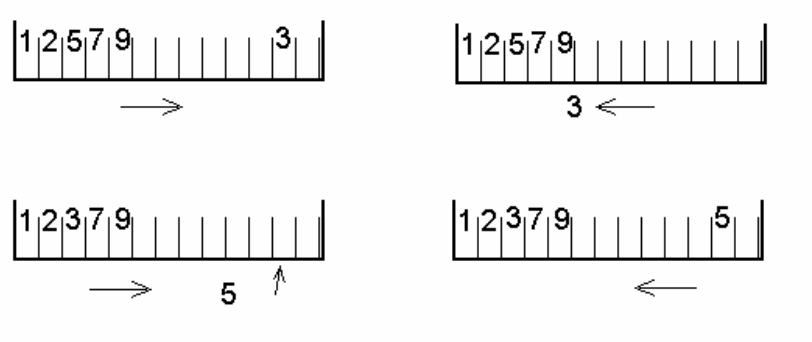
Namiesto 7 ide 3. Neposuniem kvôli tomu všetky knihy (to by som sa nadrel), ale len dve z nich vymením.

Napíšte procedúru, ktorá týmto spôsobom utriedi policu.

Pozn.: Toto triedenie sa nazýva MIN-SORT. Nájde najmenší prvok v ešte neusporiadanej časti poľa a správne ho zaradí.

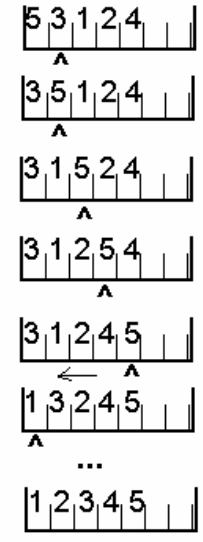
V zime býva klzko. Pani Moľová to nechcela zobrať do úvahy. Po troch dňoch ju prepustili z nemocnice. Ale len na vozíčku. Síce invalidka, ale do práce chodiť musela, lebo čitatelia boli hrozne hladní (samozrejme, po informáciách). Všetko bolo v poriadku, až na záverečné triedenie.

Bolo pre ňu totiž veľmi namáhavé kvôli každej knižke prejsť štyri razy popri policiach:



Začala hľadať spôsob triedenia, ktorý by ju stál menej pohybu.
Pani knihovníčka sa rozhodla ísť od začiatku a porovnávať dve susedné knihy.
Ak má kniha napravo menšie evidenčné číslo, tak ich navzájom vymení. Túto činnosť bude opakovať, až kým nebudú všetky knihy na svojich miestach.

Napr.:



Toto triedenie sa nazýva BUBBLE-SORT, lebo prvky akoby prebublávali, väčšie padajú hlbšie (dozadu), menšie sa derú do popredia.

Napíšte procedúru na Bubble-sort.

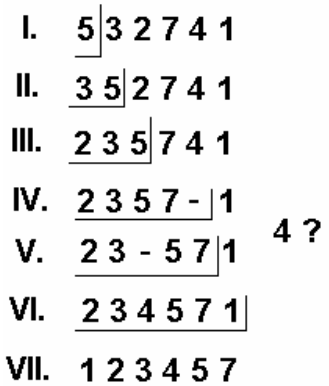
```
Procedure Bubble_sort;

begin
 for i:=1 to Pocet_prvkov do
  for j:=1 to Pocet_prvkov-1 do
   if Prvok[j] > Prvok[j+1] then begin
     pom:= Prvok[j];
     Prvok[j]:= Prvok[j+1];
     Prvok[j+1]:=pom;
   end;
end;
```

Algoritmus je najjednoduchší, no aj najpomalší, lebo beží aj vtedy, keď sú už všetky prvky usporiadané. Je vhodné pre každý cyklus (t.j. prebehnutie od prvého po posledný prvok) použiť ešte jednu premennú, ktorá si bude pamätať, či došlo k výmene. Ak nie, prvky sú už usporiadané a možno skončiť.

Pre pani Moľovú je zaujímavá aj alternatíva, že zamieňanie kníh bude robiť aj pri návrate na začiatok police - teda nepôjde naprázdno. (Znamená to dva cykly - jeden vzostupný a vzápätí druhý zostupný).

*Nájdite ďalšie urýchlenie (vezmite do úvahy cyklus s **j**).*

Po dvoch mesiacoch, keď sa tráva zazelenala, stromy zapučali, pani Moľová prišla o sadru a premiestňovanie jej prestalo robiť problémy. Pokúsila sa opäť triediť novým spôsobom.



Vezme prvú knihu a povie si, že táto je už na svojom mieste - správne uložená. Vezme druhú a porovná ju s prvou. Buď ju vymení a prvú posunie alebo ich nechá tak, lebo zatiaľ sú uložené správne.

Vezme prvú neutriedenú knihu (teraz tretiu) a nájde jej miesto v už utriedených. Knihy, ktoré majú väčšie evidenčné číslo ako zasúvané, sa posunú o jeden priečinok a kniha sa vloží.

Algoritmus sa opakuje do utriedenia všetkých kníh.

Triedenie sa nazýva INSERT-SORT (triedenie vkladaním).

Napíšte procedúru na Insert-sort.

Triedenie bolo sympatické pani Moľovej, ale výrazne nesympatické pánovi Moľovi, ktorému sa často ušla teplá večera až po Večerníčku, keď sa manželka vrátila domov. Rozhodol sa s tým niečo urobiť. Začal študovať odbornú triediacu literatúru a objavil...

...QUICK-SORT - rýchle triedenie, ktoré je založené na rekurzívnych metódach.

Odvtedy sú všetci spokojní...

Cvičenie:

1. Pokúste sa analyzovať Quick-sort. Spôsob, ktorým pracuje je rekurzívny a do úvahy sa berie nielen hodnota prvku, ale aj jeho poloha v zozname.

```
Program QuickSort;
var a:array[1..100] of integer;
    i,n:integer;
    k:char;

Procedure Tried(l,r:integer);
var i,j,x,w:integer;

begin
 i:=l;
 j:=r;
 x:=a[(l+r) div 2];
 repeat
  while a[i]<x do i:=i+1;
  while a[j]>x do j:=j-1;
  if i<=j then begin
   w:=a[i];
   a[i]:=a[j];
   a[j]:=w;
   i:=i+1;
   j:=j-1;
  end
 until i>j;

 if l<j then Tried(l,j);
 if i<r then Tried(i,r);
end;
```

```
begin
 Write(' Zadaj pocet prvkov : ');
 ReadLn(n);
 Randomize;
 for i:=1 to n do begin
     a[i]:=random(n);
     write(a[i],'  ');          {pole náhodných čísel}
 end;
 Tried(1,n);                    {volanie procedúry s celým zoznamom - prvý a posledný
                                  index}
 WriteLn;
 WriteLn;
 For i:=1 To n Do Write(a[i],'  ');
 ReadLn;
end
```

Tu končí univerzálna teória a začína prax konkrétneho jazyka. Všetky techniky potrebné pre zvládnutie algoritmizácie a všetky techniky potrebné všeobecne pre programovanie boli vyčerpané v predchádzajúcich kapitolách.

Snažili sme sa v nich podať suché, pre programátora i užívateľa na najnižšej možnej úrovni prijateľné riešenie problémov.

Predchádzajúce riadky sú síce podávané v programovacom jazyku Turbo pascal, no prechod na iné prostriedky je jednoduchý a prakticky bezproblémový. Veď techniky či algoritmy, akými sú rekurzia, backtracking, triedenia alebo binárne stromy, sa používajú temer vo všetkých viac alebo menej vyšších programovacích jazykoch.

Rovnako príbuzná je aj práca s konštrukciami typu záznam, zásobník alebo front.

Týmito riadkami dávame tenkú, bodkočiarkovanú čiaru za všeobecnými poznatkami potrebnými na zvládnutie algoritmizácie v globále a prechádzame na konkrétnejšie problémy, ktoré začínajúcim a občas aj viac skúseným programátorom spôsobujú vrásky.

Teraz, bohužiaľ, už nie je možné pracovať univerzálne, pretože prostriedky, ktoré budeme využívať, sú dosť konkrétne a často závisí len od jedinečnosti programátora, či a na akej úrovni sú k dispozícii.

Na prvom mieste sa budeme zaoberať grafikou, ktorá predstavuje ak už nie pre programátora, tak určite aspoň pre užívateľa sympatickejší a prijateľnejší komfort. Približuje abstraktný svet znakov a čísel normálnemu, medzi bežnými ľuďmi sa pohybujúcemu človeku.

Priblížime si aj niektoré techniky grafických editorov - samozrejme, zasa len na najnižšej úrovni, bez špeciálnych techník a urýchľovacích postupov.

Ďalším prvkom a možnosťou vyžmýkať z programovacieho jazyka to, čo ide, bude práca s animáciou - to, čo Turbo pascal nepodporuje, a to, čo sa dá za animáciu považovať len s najväčším sebazaprením.

Cez prácu s myšou v grafickom režime sa prenesieme na tlačenie v režime znakovom, prehľadávanie adresárov a diskov, ukrývanie a znovuobjavovanie sa kurzora.

Neodpustíme si trochu hluku pri práci so zvukmi a na záver sa pokúsime načrtnúť efektívne využívanie už existujúcich procedúr v pascalovských unitoch.

3. Grafika

3.1 Úvod do grafiky

V Turbo pascale nie je grafika (grafické príkazy) priamou súčasťou systému. Príkazy na kreslenie, zmenu farieb atď. sú uložené v unite ***graph***. Okrem toho treba zvoliť aj vhodný driver (ovládač). Driver je "program" (časť programu), ktorá sa natiahne do pamäti, zostane v nej rezidentná, a potom umožňuje zobrazovanie bodov na obrazovke (resp. "povie", kam treba na obrazovke nakresliť bod).

Drivery sa v Turbo pascale používajú kvôli tomu, že v praxi boli (resp. občas aj sú) rôzne grafické karty (čipy - alebo skôr dosky), vďaka ktorým je možné zobrazovanie na PC vôbec. Použitie driveru závisí od schopností grafickej karty počítača. Všeobecne platí, že pri schopnejšej karte (karte vyššej úrovne) je možné použiť ovládač karty nižšej kategórie.

Ak zabehneme trochu do hardware, pri zjednodušení sa grafické karty líšia jedine rozlíšením (počtom bodov na obrazovke) a počtom farieb, ktoré na nich možno zobraziť.

Ovládače poskytované Turbo pascalom sú od CGA cez Hercules až po SVGA.

Teraz, keď už máme (snáď) predstavu o tom, čo je ovládač a na čo slúži, môžeme opatrne pristúpiť k oživeniu grafickej obrazovky v pascale.

Prvá a dôležitá vec, bez ktorej sa nepohneme, je hneď na začiatku programu - treba prebudiť (inicializovať) ***unit graph***, ktorý (ako sme už spomínali) obsahuje všetky grafické príkazy. Teda:

```
Program Prvy_pokus;
uses graph;
```

Ďalej (už v tele samotného programu) zvolíme driver, s ktorým budeme pracovať, a inicializujeme ním grafiku.

Ak by nám neskôr kompilátor vyhlasoval chybu, že nemôže nájsť súbor ***graph.tpu***, treba v menu *Options-Directories*, v položke *Unit directories* nastaviť cestu do adresára, v ktorom sa nachádza (väčšinou /BP/UNITS).

Napr.:

```
var gd, gm: integer;
begin
 gd: = Detect;
 InitGraph (gd, gm, 'C: \TP\bgi');
```

Prvý riadok uloží do premennej ***gd*** číslo (konštantu 0) driveru na používanú grafickú kartu. Je to automatické detekovanie, automaticky sa určí najlepší možný. V prípade, že chceme použiť konkrétny driver nižšej úrovne, priradíme ***gd*** jeho číslo (1-10).

Do *gm* sa po použití funkcie ***Detect*** vloží grafický mód zvoleného driveru. Ten podrobnejšie hovorí - o čom inom ako o móde.

Driver (pre ***egavga.bgi*** - EGA alebo VGA karta) možno zvoliť pri móde 1 ako 320 x 200 x 256 farieb, pri móde 0 ako 640 x 480 x 16 farieb.

Pokiaľ sa do ***gm*** nevloží nič priamo, nadobúda hodnotu 0.

Procedúra ***InitGraph*** už vkladá zvolený driver vo zvolenom móde do pamäte. Tretím parametrom (okrem driveru a módu) je cesta k adresáru, v ktorom treba súbor s driverom hľadať. Štandardne býva uložený v adresári BGI (má koncovku *bgi*).

Štandardne dodávaných driverov je 10. Napriek tomu sa môže stať, že nám ani jeden z nich nevyhovuje, resp. máme definovaný vlastný. (V súčasnosti, keď väčšina PC má v sebe SVGA kartu, je dosť neefektívne a neefektné používať ovládač na EGA). Najčastejšie sa používa ***egavga.bgi***.

Ak chceme kresliť v 256 farbách alebo v rozlíšení 800 x 600 či
1024 x 768 bodov, určíme si vlastný užívateľský driver. V praxi ich je vytvorených už mnoho a kolujú v programátorskej verejnosti.

Schéma je asi takáto:

```
uses graph;

var autodetect: pointer;
 gd, gm: integer;

{$F+}
Function det: integer;

begin
 det: =2;          {640x480x256}
 {det: =3;         {800x600x256}
 {det: =4;         {1024x768x256}

end;
{$F-}

begin
 AutoDetect: =@det;
 GD: = InstallUserDriver ('SVGA256', AutoDetect);
 GD: = Detect;
 InitGraph (GD, GM, '\bgi');
end.
```

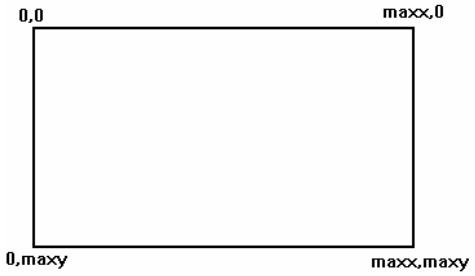
Inicializáciu máme za sebou, ***InitGraph*** inicializovaný, unit ***graph*** pripravený a môžeme pristúpiť ku kresleniu.

Najjednoduchšou činnosťou, ktorú môžeme vykonať, je vykreslenie (jediného) bodu na obrazovku.

Slúži naň procedúra:

```
PutPixel (x, y, farba);
```

Udáva súradnice bodu v kartézskej sústave a jeho farbu (väčšinou 0 - 15 s tým, že prvých 8 farieb je tmavých, ďalších 8 svetlých. Farby sa zadávajú buď číslom alebo konštantou - slovom (biela - white, žltá - yellow, ...).



Súradnice však nie sú definované tak, ako sme zvyknutí zo života (alebo skôr z matematiky). Vo verejnosti koluje historka, že tvorcovia Turbo pascalu mali hotový celý systém, keď si naraz ich šéf zmyslel rozšíriť ho o grafickú časť.

Väčšina programátorov bola už na dovolenke a tí ostatní v časovej tiesni dosť zbrklo navrhli systém a rovnako zbrklo ho aj dokončili. Túto zbrklosť okrem súradnicového systému vidieť aj pri práci s ***image*** (viď ďalšiu kapitolu).

Vráťme sa ale späť ku grafickým príkazom. Bod možno nielen vykresliť, ale neskôr aj zistiť jeho farbu.

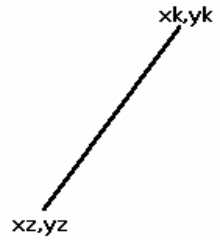
Na to sa používa procedúra:

```
Farba: = GetPixel (x, y);
```

Vedieť nakresliť bod a zistiť jeho farbu je to základné, čo potrebujeme a s čím by sme si vystačili. Turbo pascal nám však prácu uľahčuje poskytovaním ďalších príkazov (procedúr) na vykreslenie čiary, obdĺžnika, kružnice...

Čiara:

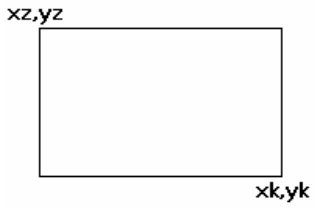
```
Line (xz, yz, xk, yk);
```



Parametre sú počiatočný a koncový bod v kartézskej súradnicovej sústave.

Obdĺžnik:

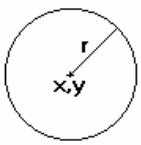
```
Rectangle (xz, yz, xk, yk);
```



Udávame protiľahlé rohy obdĺžnika: ľavý horný a pravý dolný alebo naopak.

Kružnica:

```
Circle (x, y, r);
```

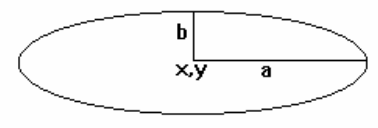


- x, y - stred kružnice,
- r - polomer.

Elipsa:

```
Ellipse(x,z,uz,uk,a,b);
```

Tu je to trochu zložitejšie; ***x*** a ***y*** udávajú stred elipsy, ***a***, ***b*** dĺžku hlavnej a vedľajšej polosi. Rozšírením sú parametre ***uz*** a ***uk***, ktoré hovoria o začiatočnom a koncovom uhle.

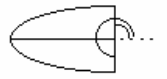


V praxi to znamená toto:

Ellipse (x, y, 0, 360, a, b);

Ellipse (x, y, 30, 90, a, b);

Ellipse (x, y, 90, 270, a, b);



Možno kresliť nielen elipsy, ale aj ich časti (na obrazovke sú elipsy zobrazené, samozrejme, vzhľadom na počítačovú súradnicovú sústavu s bodom 0, 0 vľavo hore). Uhly sa nezadávajú v radiánoch, ale v stupňoch.

Všetky útvary sa kreslia bielou farbou, pokiaľ táto nie je zmenená príkazom:

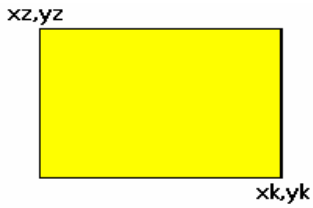
```
SetColor(Nova_farba);
```

Nastavenie farby zostáva rovnaké až po ďalšie použitie procedúry ***SetColor***.

Okrem kreslenia "prázdnych" útvarov možno pracovať aj s útvarmi s výplňou. Na to slúžia ďalšie príkazy.

Plný obdĺžnik:

```
Bar(xz,yz,xk,yk);
```



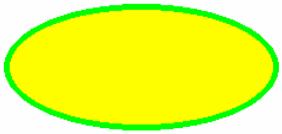
- nakreslí plný obdĺžnik bez obrysov (obruby).

Plná elipsa:

```
FillElipse(x,y,a,b);
```

- x, y - stred

- a, b - polosi
Obruba bude vykreslená farbou nastavenou pomocou ***SetColor***.



Výplne, rovnako ako farbu pri predošlých útvaroch, možno meniť a nastavovať.
Slúži na to príkaz:

```
SetFillStyle (Vzorka, Farba);
```

Druhý parameter udáva nám už známu farbu, prvý zobrazuje vzor výplne - od bodkovanej cez prečiarknutú až po prečiarkovanú krížom, samozrejme, všetko v rôznej hustote. Najpoužívanejšie vzorky:
1 - vyplní celý útvar jednoliatou nastavenou farbou,
0 - vyplní útvar farbou pozadia, bez ohľadu na farbu nastavenú druhým parametrom.

Pozadie v grafike nastavuje procedúra:

```
BackGround(Farba);
```

Ďalšou procedúrou grafického jadra pascalu je procedúra ***FloodFill***, ktorá vypĺňa oblasti nepravidelného tvaru.
Zadá sa jeden vnútorný bod nepravidelného útvaru a procedúra vyplní útvar až po jeho hranice.
Príkaz má tvar:


```
FloodFill (x, y, Farba);
```

kde ***x***, ***y*** sú súradnice vnútorného bodu a ***Farba*** je farba obrysov.

Výstraha:
- V prípade, že útvar nie je uzavretý, výplň obyčajne pretečie a pripraví nás o dosiaľ vykreslené objekty.
- Pri veľmi zložitých útvaroch sa stáva, že vypĺňanie buď neprebehne do konca alebo počítač proste zamrzne.

Samozrejmosťou pre prácu s grafickými prostriedkami je aj možnosť písať viacerými druhmi písma (fontami), centrovať ich, meniť veľkosť, farbu apod. Turbo pascal vo svojom štandarde ponúka 10 vektorových (tzv. oblých) fontov a jeden bitmapový - hranatý.

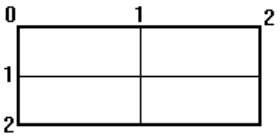
Meniť druhy a veľkosť písma možno procedúrou:

```
SetTextStyle(Font,Smer,Velkost);
```

Font je číslo (konštanta) fontu, ktorým chceme písať. Napr.:
0 - systémový (bitmapový),
2 - drobné písmo,
4 - gotické písmo,
10 - obrysové písmo atď.
Velkost - od 1 do 15 je dopredu definovaná veľkosť písma (pri každom fonte iná),
Smer - smer, v akom bude text zobrazený (priamo, zhora nadol, ...).

```
SetTextJustify (cx, cy);
```

nastavuje centrovanie písma. Je v rozsahu 0 - 2.



Napr.: ***SetTextJustify (1, 1)*** sa postará o to, aby sa centrovalo vzhľadom na stred.

Štandardné nastavenie je 0, 0.

Samotné vypísanie textu sa spúšťa príkazom:

```
OutTextxy(x,z,Text);
```

x, ***y*** - súradnice, vzhľadom na ktoré sa bude text umiestňovať,
Text - reťazec znakov, ktorý chceme vypísať.

Pri vypisovaní textov je často potrebné napísať aj číslo (vybrať ho z číselnej premennej alebo vypísať ako výsledok numerických operácií).
A tu je problém! ***OutTextxy*** totiž nedokáže vypísať samotné číslo ako číslo. Je toho schopný len v prípade, že číslo sa bude tváriť ako reťazec.
Tento problém sa rieši prevedením čísla na reťazec procedúrou ***Str*** a vypísaním čísla ako textu:

```
i: = Random (1000);
str (Retazec, i);
OutTextxy (x, y, Retazec);
```

A požadované číslo svieti na monitore nastavenou farbou...
Ďalším častým problémom je vypisovanie stavu počítadla na to isté miesto.
Riešenie typu:

```
i: =1;
while i<=max do begin
Str (i, Retazec);
OutTextxy (10, 10, Retazec);
i: =i+1;
end;
```

je úplne nevhodné, pretože po niekoľkých opakovaniach máme na obrazovke na mieste počítadla len farebný chaos. ***OutTextxy*** nemaže predtým zobrazený text, ako to robili procedúry ***Write*** alebo ***Gotoxy***.

Túto kolíziu treba riešiť systémom:

```
SetFillStyle (0, Pozadie);
i: =1;
while i<=max do begin
       Bar (0, 0, 20, 20);
       Str (i, Retazec);
       OutTextxy (10, 10, Retazec);
       i: =i+1;
end;
```

tzn. miesto, na ktorom bol vypísaný znak, vždy pred vypísaním ďalšieho vyčistite - položte naň obdĺžnik farby pozadia,

alebo

```
while i<=max do begin
SetColor (Pozadie);
OutTextxy (10, 10, Retazec); {zmaze povodny vypis}
i: =i+1;
SetColor (Pismo);
Str (Retazec, i);
OutTextxy (10, 10, Retazec);
end;
```

Prvý spôsob je rýchlejší, druhý "krajší".

Nemali by sme zabudnúť na zmazanie obrazovky, ktoré vykoná procedúra ***ClearDevice***.

A na záver práce v grafickom režime by bolo treba uvoľniť miesto v pamäti, ktoré zaberá nainštalovaný ovládač. Jednoducho:

```
CloseGraph;
```

3.2 Grafické editory

Okolo počítačovej grafiky, počítačových grafikov a grafických programov sa vo svete robí veľké haló. Grafické (animačné) programy sú schopné bez hercov vyprodukovať film, pridávať zvláštne efekty, spriesvitňovať objekty, dosádzať neexistujúce tvory.

Štandardom alebo skôr priekopníckym objektom sa stal v minulých rokoch Spielbergov Jurský park a pomerne nedávno film Maska s Jimom Carreym.

Napriek obdivuhodným výsledkom je to všetko len hra s bodmi, farbami a ich kombináciami.

Na vytváranie obrázkov či na ich úpravu sa používajú grafické editory.

Taký úplne obyčajný vám umožní kresliť body, čiary, kružnice, obdĺžniky, elipsy... Teda len to, o čom sme doposiaľ hovorili.

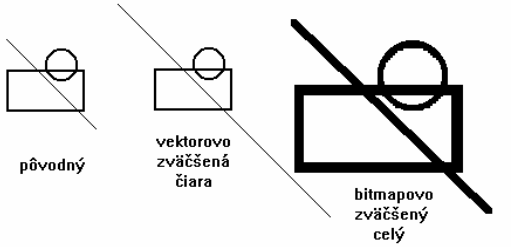
Okrem týchto triviálnych objektov je schopný kopírovania, zmenšovania, zväčšovania (lepší aj rotácie) - stále nič zložité.

A každý editor má ešte rad špeciálnych a špecifických funkcií a v sebe zabudovaných efektov (či vzorov), ktoré ho robia jedinečným a silným voči konkurencii.

Grafické editory ako také delíme na dva základné druhy:
- vektorové,
- bitmapové.

Vo vektorových pracujeme s objektami, v bitmapových s plochami. Čo to znamená?

Ak chceme vo vektorovom grafickom editore zväčšiť čiaru, klikneme na ňu myšou (alebo ju inak označíme) a natiahneme ju. Okolité objekty sa nemenia.



V bitmapovom editore vezmeme (označíme) výrez, v ktorom sa čiara nachádza, a zväčšíme ho spolu s pozadím.

Oba druhy majú svoje výhody i nevýhody. Naším cieľom nie je o nich polemizovať, len ich predstaviť.

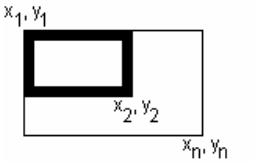
Úloha (konečne):
Napíšte program (bitmapový grafický editor), ktorý bude schopný kresliť na zadané miesta čiaru, kružnicu, obdĺžnik (plný i prázdny) a bude schopný zväčšovať, zmenšovať a rotovať vybrané obdĺžniky z obrazovky.

S prvou časťou úlohy (s vykresľovaním) by nemali byť žiadne problémy. Horšie to bude len s procedúrami zväčšovania, zmenšovania a rotovania.

Čo znamená bitmapovo zväčšiť body?

Je vidieť, že je to len zmena bodu na štvorec so stranou dlhou ako pomer zväčšenia.

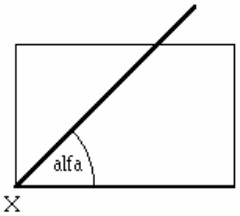
Povedzme, že máme zväčšiť výrez - obdĺžnik - takým spôsobom, ako je vyznačené na obrázku.
Ako postupovať?
Logické je odspodu nahor a sprava doľava tak, aby sme nepremazávali pôvodný obdĺžnik, kým budeme z neho čítať údaje.



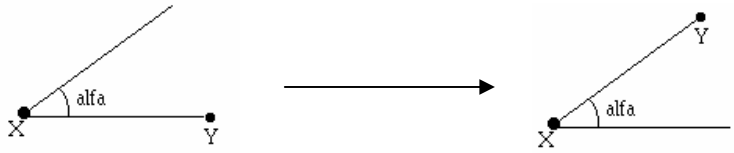
Procedúra bude vyzerať asi takto:

```
Procedure Zvacs (x1, y1, x2, y2, Pomer: integer);

Begin
 For i: =x2 downto x1 do
   For j: =y2 downto y1 do begin
     SetFillStyle (1, getpixel (i, j));
                  {nastaví farbu vytváraného štvorca na
                  farbu bodu, ktorý mu zodpovedá}
     Bar (x2+ (i-x1)*Pomer, y1+ (y2-j)*Pomer,
     x2+ (i+1-x1)*Pomer, y1+ (y2-j+1)*Pomer);
                                    {vykreslí štvorec}
     end;
end;
```

Rovnakým spôsobom možno vyriešiť aj zmenšenie.
Menším problémom je rotácia objektu okolo bodu. Táto činnosť sa vo väčšine bitmapových editorov taktne obchádza, pretože je jednak pomalá a jednak pri nej dochádza k značnému skresľovaniu pôvodných objektov.



Povedzme, že máme otočiť daný obdĺžnik (samozrejme aj s jeho vnútrom) okolo bodu X. Čo to znamená?
Ak otočíme bod Y, jeho súradnice sa zmenia:



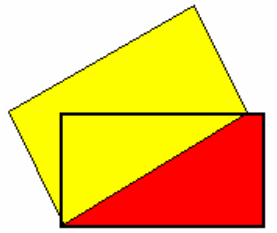
NY (x) = vzd (X, Y) . cos (alfa) + X (x)
NY (y) = vzd (X, Y) . sin (alfa) + X (y)

vzd (X, Y) je vzdialenosť bodov X, Y a z Pytagorovej vety vieme, že:

vzd (X, Y) = sqrt (sqr (X (x)-Y (x)) + (X (y) - Y (y))).

A z tohto vzťahu vieme určiť nové súradnice každého bodu obdĺžnika. Keď vystriedame všetky body, pôvodný obdĺžnik bude zrotovaný.

Tento spôsob bol najjednoduchším, ktorý sa ponúkal. A tým, že bol najjednoduchší, bol aj najnepresnejší. Ak sa pozorne zadívate na takto zrotovaný výrez, zistíte, že niektoré body sa pri rotácii stratili.

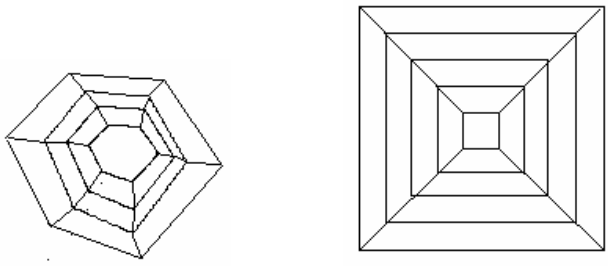


Takéto nedostatky sú, samozrejme, tiež riešiteľné:

Zistíme obrysy nového obdĺžnika a budeme ho vypĺňať tak, že budeme zisťovať farbu každého jeho bodu.

Cvičenia:

1. Nakreslite pavučinu.



2. Nakreslite pavúka a prispôsobte ho tak, aby sa menil podľa vstupných parametrov (dĺžka, výška...).

3. Nakreslite obdĺžnik rotovaný okolo stredu.

4. Urobte les. Stromy (polohu) generujte náhodne.

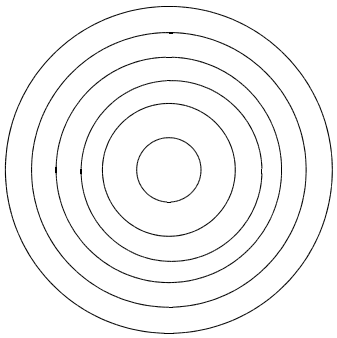
5. Pustite loptu zhora obrazovky nadol.

Pozn: *Pohyb guličky možno robiť tak, že ju mažete (prekresľujete) farbou pozadia a vykresľujete na novej pozícii.*

6. Napíšte program, ktorý bude:
a) pohadzovať guličku po obrazovke,
b) postavte jej do cesty raketu,
c) urobte z toho hru: raketou budete odrážať loptičku, keď vám spadne, máte o život menej,
d) upravte na hru pre dvoch hráčov.

7. Nakreslite tabuľku s daným počtom riadkov a daným počtom stĺpcov.

8. Urobte terč.

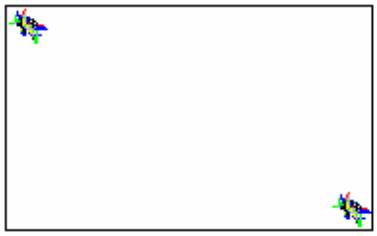


9. Nakreslite:

a) pyramídu, b) plot, c) slnko, d) hory, e)cencúle.

3.3 "Animácia" v Pascale

Ako už sám názov napovedá, budeme sa v tejto časti baviť o obrázkoch- ***images***. Príkazy na prácu s nimi sú súčasťou unitu ***graph***.



Nakreslite do ľavého horného rohu obrázok (pomocou známych grafických príkazov) a napíšte procedúru, ktorá ho skopíruje do pravého dolného rohu bod po bode.

Riešenie je jednoduché, no dosť zdĺhavé a pomalé. Treba totiž zisťovať farbu každého bodu a každý bod aj osobitne vykresľovať.

Oveľa jednoduchšie (elegantnejšie) to možno urobiť za pomoci procedúry:

```
GetImage (xz, yz, xk, yk, Pamat);
```

Procedúra si zapamätá vzhľad obrazovky na vyznačených miestach.
xz, ***yz***, ***xk***, ***yk*** - súradnice pravouhlého výrezu obrazovky, ktorý chceme uchovať,
Pamat - je buď pole (pevne dané), do ktorého sa uloží rozmer obrázku
a (zakódované) farby jednotlivých bodov,
- alebo ukazovateľ na časť pamäti, ktorú si predtým vyhradíme
asi takto:

```
Velkost_obrazku: = ImageSize (xz, yz, xk, yk);
GetMem (Pamat, Velkost_obrazku);
GetImage (xz, yz, xk, yk, Pamat^);
```

Rozoberme teraz túto trojriadkovú časť programu postupne:
ImageSize - zistí, akú veľkú pamäť budeme potrebovať na to, aby sme
boli schopní zapamätať si výrez obrazovky (xz, yz, xk, yk).
GetMem - procedúra, ktorá túto pamäť vyhradí.
- prvý parameter je smerník, ktorý na danú časť ukazuje, druhý
veľkosť, ktorá sa vyhradí a pamäť zaberie.
GetImage - parametre už poznáme, ale keďže posledný je smerník,
nesmieme zabudnúť na ^.

Dostali sme sa do stavu, keď obrázok máme uložený kdesi v pamäti a zostáva nám už len vykresliť ho (položiť) na vybrané miesto:

```
PutImage(xz,yz,Pamat,mod);
```

alebo

```
PutImage(xz,yz,Pamat^,mod);
```

x, ***y*** - bod, od ktorého začneme vykresľovať (ľavý horný roh nového umiestnenia pamätaného výrezu).
Nie je potrebné zadávať pravý dolný roh, pretože rozmery obrázku (obdĺžika) sú uložené v ***Pamat*** alebo kdesi v ***Pamat***^,
Pamat, ***Pamat***^ - už bolo vysvetlené,
mod - určuje, akým spôsobom sa bude obrázok vykresľovať, t.j. ako sa bude znášať s pozadím, na ktoré bude vykreslený.

Správanie sa voči pozadiu, samozrejme, nepostrehneme, pokiaľ obrázok budeme vykladať len na čierne (čisté) pozadie. Zmeny uvidíme, až keď bude pozadie mierne "začiarané".
0 - vykreslí obrázok presne v pôvodnom tvare,
1 - 3 - logické súčty, súčiny a iné operácie,
4 - inverzné zobrazenie pôvodného obrázku (t.j. čierna bude biela, modrá žltá...).

V predchádzajúcom texte sme vám ponúkli dva spôsoby na prácu s ***GetImage*** a ***PutImage***. Pre porovnanie ešte raz:

```
var Pamat: Array[1..max] of word;

begin
 GetImage (0, 0, 10 , 10, Pamat);
 PutImage (100, 100, Pamat, 0);
end;
```

alebo

```
var Pamat: Pointer;

begin
 Velkost: = ImageSize (0, 0, 10, 10);
 GetMem (Pamat, Velkost);
 GetImage (0, 0, 10, 10, Pamat^);
 PutImage (100, 100, Pamat^, 0);
 FreeMem (Pamat, Velkost);
end;
```

Prvý (statický) spôsob je oproti druhému (dynamickému) relatívne jednoduchší. Aspoň z hľadiska programátora. Druhý spôsob však šetrí pamäť. Vyhradí si ju procedúrou ***GetMem***, a keď obsah už nepotrebuje, uvoľní ju pomocou ***FreeMem*** pre ďalšie operácie.

```
FreeMem(Pamat,Velkost);
```

Pamat - smerník, pre ktorý bola pamäť vyhradená,
Velkost - pôvodná veľkosť vyhradenej pamäte.
Oba parametre sú tie isté ako pre ***GetMem***.

Pozn.: Pri pevnom vyhradení pamäte sa často stáva, že časť poľa je prázdna - nevyužitá zbytočne zaberá pamäť.

Napíšte program, v ktorom bude po obrazovke poletovať (náhodne alebo podľa nejakého algoritmu) lietajúci tanier. Nezabudnite na vykreslenie planét, hviezd, rojov meteoritov (tie môžu byť statické) ap.

Tento príklad už trochu zaváňa jednoduchou (primitívnou) animáciou. Ako na to?

```
begin
  Vykresli_pozadie;
  repeat
    Zapamataj_si_pozadie (x, y, x+20, y+20, pamat);
    Vykresli_UFO (x, y);
    Delay (10);
    Vrat_pozadie;
    Zmen (x, y);
  until KeyPressed;
end;
```

Vykresli_pozadie - procedúra, ktorá sa postará o spomínané planéty.

Zapamataj_si_pozadie - predtým, ako vykreslíme UFO, potrebujeme si zapamätať všetko, čo bolo za ním, aby sme po jeho posunutí sa na inú pozíciu dokázali všetko vrátiť do pôvodného stavu a zvýšiť tak dôveryhodnosť "animácie".

Vykresli_UFO - nakreslí samotné UFO, a to:

- buď ho vykreslíme ručne (pomocou čiar, elíps...),
- alebo:

1. Pred spustením cyklu niekde nakreslite UFO,
2. Vezmite ho do pamäti pomocou ***GetImage***,
3. Vždy ho v procedúre ***Vykresli_UFO*** vyložte (a vyskúšajte rôzne módy - čo to urobí).

Vrat_pozadie - položte späť to, čo ste si zapamätali. Ak použijete mód 0, čo sa väčšinou robí, dostanete pôvodné pozadie a zároveň prekreslíte aj UFO (zmažete ho).

Pohybový cyklus by mohol vyzerať napr. takto (ak UFO má rozmery 20 x 10 bodov):

```
Procedure Pohyb;
var  Pozadie, UFO: Array[1..5000] of word;
     x, y: Integerô

begin
  Nakresli_UFO (0, 0);
  GetImage (0, 0, 20, 10, UFO);
  ClearDevice;
  Vykresli_pozadie;
  x: = 0;
  y: = 50;
  repeat
    GetImage (x, y, x+20, y+10, Pozadie);
    PutImage (x, y, UFO, 0);
    Delay (10);
    PutImage (x, y, Pozadie, 0);
    x: = x+5;
  until x = 620;
end;
```

Ak si všimneme riadok s ***PutImage (x, y, UFO, 0)***, tak UFO sa vykreslí vždy do čierneho štvorca a zmaže všetko, čo sa v ňom pôvodne nachádzalo.

Tento problém (t.j. aby body, ktoré neboli nakreslené neprekresľovali pozadie) sa v Turbo pascale vyriešiť nedá. Je to výsledok už spomínanej zbrklosti autorov Turbo pascalu.

Možno ho obchádzať len rôznymi "barličkovými" spôsobmi, ktoré sú však pre každú situáciu špecifické. Univerzálnym riešením by bolo napísanie assemblerovskej rutiny (kvôli rýchlosti), ktorá by vykresľovala len body odlišné od farby pozadia. Ale to nie je problém pre nás (aspoň momentálne).

Pozn.: Zaujímavou možnosťou prekresľovania je aj to, že pri dvojnásobnom položení toho istého obrázku v móde 1 sa vráti to isté pozadie:

```
PutImage (x, y, Pamat, 1);      {vykreslí}
PutImage (x, y, Pamat, 1);      {zmaže}
```

Dvojnásobné vykreslenie sa vlastne zruší - premaže (a netreba si pamätať ani pozadie).

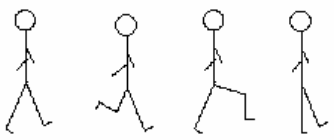
*Upravte procedúru **Pohyb** do "dynamického tvaru".*

Táto pseudoanimácia bola pomerne nedôveryhodná, pretože v podstate išlo len o nemenné posúvanie jediného objektu.
Ak chcete, aby animácia vyzerala dôveryhodne a "skutočne", musíte použiť minimálne 3-4 obrázky.

Napíšte program, ktorý ukáže idúceho človeka a nechá ho prejsť cez obrazovku zľava doprava.

Tu môžete naplno popustiť uzdu svojej fantázii a realizovať pohyby pomocou viacerých obrázkov.

Napr.:



alebo aj podrobnejšie...

Nechajte proti sebe bežať dvoch takýchto ľudí.

Po vyriešení týchto problémov sa dá povedať, že práca s obrázkami a jednoduchý pohyb je pre vás maličkosťou. Vážnejšie problémy však nastanú, pokiaľ chcete pomocou ***GetImage*** vziať do pamäti viac ako časť obrazovky, ktorá zaberá 64 kB (tzn. pri 16 farbách EGA je to asi tretina, pri 256 - šestina). Táto procedúra to prosto nevie. Musíte svoj veľký výrez rozdeliť na viac malých a tie brať po jednom.
Ak chcete ukladať nakreslené obrázky na disk, ponúka sa tu opäť viac možností. Tá najjednoduchšia je definovať pre obrázky pole a obrázky potom ukladať do súboru, ktorý je typu pole.

Môže to byť realizované napr. takto:

```
Type Pamat = Array[1..5000] of word;
var  Pam: Pamat;
     ff: File of Pamat;
```

```
Procedure Uloz (Obrazok: Pamat);

begin
  Assign (ff, 'obrazky.obr');
  Rewrite (ff);
  Write (ff, Obrazok);
  Close (ff);
end;
```

Procedúra uloží celý obsah poľa ***Pamat*** (aj ak je tam voľné, nevyužité miesto), t.j. aj s rozmermi obrázku.

Teraz ho možno hocikedy prečítať a vyložiť na obrazovku:

```
Procedure Citaj (var Obrazok: Pamat);

begin
  Assign (ff, 'obrazky.obr');
  Reset (ff);
  Read (ff, Obrazok);
  Close (ff);
  PutImage (10, 10, Obrazok, 0);
end;
```

Ak chceme uložiť celú obrazovku na disk, môžeme to robiť bod po bode jednoduchým pakovaním (zapamätáme si farbu a počet bodov za sebou, ktoré ju majú) alebo pomocou image-procedúr.

Tu už ale treba obrazovku deliť na niekoľko častí.

Cvičenia:

1. Napíšte procedúru, ktorá preklopí obrazovku okolo stredu podľa:
a) osi x, b) osi y.
Návod: Vymeňte vždy dva zodpovedajúce si riadky (stĺpce).

2. Napíšte procedúru, ktorá pri stlačení F2 uloží obrázok na disk, pri F3 vráti posledné uloženie.

3. Nechajte po obrazovke pohybovať viac guličiek (nejakým algoritmom) súčasne. V prípade zrážky nech sa efektne rozsypú.

4. Simulujte deň (t.j. nechajte slnko prejsť veľkým oblúkom cez obrazovku).

5. Simulujte padajúci dážď (kvapky) a sneh (vločky).

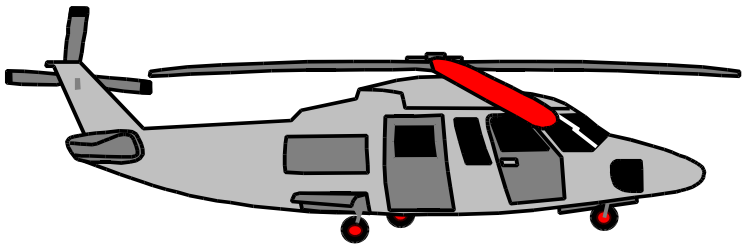
6. Pridajte k 5. aj vplyv vetra - šikmé padanie.

7. Napíšte vlastné grafické procedúry, ktoré upravia bod 0, 0
a) do ľavého dolného rohu,
b) do stredu obrazovky.

8. Nechajte po obrazovke lietať Windowsové okno.

4 Hlboké zábery

4.1 Brat génia (READKEY)



Ferko Mušlička je malý programujúci programátor. Je schopný vyriešiť každý matematický problém. A podľa toho by sa mohlo zdať, že je bezproblémovým človekom.

Lenže, ako to už obyčajne býva, zdanie zasa klame. Ferko má totiž mladšieho brata a práve pri ňom všetky trable začínajú. Mladší brat - krpec alebo podľa rodného listu Michal - je hrozne lenivý a okrem hrania sa s počítačom nie je ochotný nič robiť. Jeho prospech je nie na zaplakanie, ale skôr na potočenie (plakanie potokmi sĺz).

Ferko sa preto rozhodol napísať hru a pomocou nej prinútiť malého, aby konečne začal robiť čosi zmysluplné.

O čo pôjde? Krpec má rád lietanie, tak to treba vmontovať do nejakých lietajúcich predmetov... A po troch hodinách Ferka napadlo geniálne riešenie.

Helikoptéra letiaca nad morom, lesom alebo púšťou vypustí dvoch parašutistov, ktorí budú mať na sebe basketbalové dresy americkej NBA (t.j. čísla od 1 do 99). Po ich vyskočení helikoptéra vyvesí vlajku s počtovou operáciou (+, - , ., /) a Miškovým cieľom bude vyriešiť úlohu a napísať výsledok skôr, ako sa parašutisti dostanú na zem. Ferko rafinovane prišiel na to, že najlepšie bude urobiť z malého brata záchrancu oboch výsadkárov. Padáky sa im totiž otvoria až vtedy, keď Miško správne vyrieši príklad. Ak to nestihne pred dopadnutím nešťastníkov na zem alebo sa pomýli majú smolu - rozpľaštia sa na zemi.

Napíšte program namiesto Ferka (t.j.: vypustite parašutistov oblečených do dresov s náhodnými číslami a náhodne zvolenou operáciou).

Napísať program je úplne jednoduché.

Jedinou úlohou (a malým problémom) je postarať sa o to, aby parašutisti padali stále, t.j. aby program nečakal na zadávané čísla pomocou ***ReadLn***, aby parašutisti padali, padali, padali... a Miško mal stále šancu (do uplynutia časového limitu) napísať správny výsledok.

Na čítanie (zadávanie údajov do počítača) sa používal príkaz (procedúra) ***Read*** (***ReadLn***), ktorý ale potreboval údaje vždy potvrdiť stlačením *ENTER* a dovtedy prerušil vykonávanie programu.

My môžeme vyriešiť problém pomocou dvoch funkcií:

```
KeyPressed,
ReadKey.
```

Prvá je boolovská - jej hodnota je **TRUE**, ak sa stlačí kláves, a inak **FALSE**. Pravdivou zostáva, až kým sa všetky znaky pomocou ***ReadKey*** z buffera (textový zásobník - miesto, kam sa zapisujú stlačené klávesy a kde čakajú na prečítanie) nevyberú.

Druhá funkcia - ***ReadKey*** - je typu ***char*** a vracia znak, ktorý bol stlačený (pokúša sa vybrať z buffera prvý znak). Ak nebolo stlačené nič (buffer je prázdny), tak čaká na stlačenie klávesu. Funkcia vracia pre znaky a čísla samotný znak (ktorý bol stlačený) v rovnakej podobe, ako je zobrazený na klávesnici, a pre kontrolné (riadiace) klávesy (F1 - F12, šípky, Home...), kombinácie (Alt + kláves, Ctrl + kláves, ...) to má vyriešené opäť dosť divoko.

Pri prvom teste vráti funkcia ***ReadKey*** číslo 0, pri druhom nejaký znak (pre šípky napr. 'H', 'K', 'M', 'P').

Teda v programe, kde

```
...
a:= ReadKey;
a:= ReadKey;  ...
```

stlačíme písmeno (napr. 'X'), musíme stlačiť ešte čosi, lebo program má prázdny buffer a ***ReadKey*** z neho nemá čo vybrať.

Ak však stlačíme niektorý z riadiacich klávesov, do buffra sa vložia dva znaky (prvý je 0, druhý podľa situácie) a dvojsekvencia prebehne plynule.

Na to, aby sme zbytočne nečakali na stlačenie klávesu, sa používa test

```
if KeyPressed then a:=Readkey;
```

t.j. ak je stlačený kláves, načítaj ho do premennej ***a***, ktorá je typu ***char*** a možno s ňou podľa toho aj pracovať. Test je dobrý, ale vhodnejšie je:

```
if KeyPressed then begin
     a:=Readkey;
     if a = #0 then begin
            a:=Readkey;
            ...
            end else begin
            ...
     end;
end;
```

Táto časť otestuje aj riadiace klávesy.

Ak sa vrátime späť, k súrodencom Ferkovi a Miškovi, program by mohol vyzerať takto:

```
begin
  Vymysli_cisla;
  Vyhod_parasutistov;
  Koniec:=FALSE;
  Vysledok:='';
  repeat
    Pokles;
    if KeyPressed then begin
      a:=ReadKey;
      if  a=#0 then begin
            a:=readkey;
```

```
            a:=readkey;
       end else begin
            if ( (a=>'0') and (a<='9'))
                then Vysledok:=Vysledok+a;
                         {ak bolo tlačené číslo / znak
                         pridaj ho na koniec}
                                                                  if a=#13 then
Koniec:=TRUE;
                                        {#13=kód ENTER}
            Vypis_cislo;
          end;
       end;
   until Koniec or Parasutisti_na_zemi;

   val (Vysledok, Cvysledok, Kod);
   if Cvysledok = Spravne_riesenie then
                                Vystrel_padak
   else
                                Bez_padaka;
 end.
```

Program bol perfektne prepracovaný, grafické efekty a poškrekovanie parašutistov dokonalé, Miškovi sa to páčilo, čosi nové sa naučil, ale po čase ho začalo nudiť, že program musí spúšťať po každom zoskoku znovu.

Upravte program tak, aby sa opakoval, kým nebude stlačený ESC (kód #27) a na obrazovku vypisoval percentuálnu úspešnosť vyriešených príkladov.

Zasa bolo chvíľu všetko v poriadku - aspoň kým sa Miško učil počítať. Po niekoľkých dňoch bol však natoľko rýchly, že ho pomalé zosúvanie parašutistov začalo nudiť.

Napíšte (upravte) program tak, aby sa rýchlosť padania parašutistov zvyšovala podľa percentuálnej úspešnosti (t.j. pri
90 % úspešnosti - veľká rýchlosť, pri 5 % - takmer nulová).

Tieto úpravy urobili z Miška hviezdu (bol najrýchlejším počtárom na škole) a Ferkovi priniesli pocit uspokojenia (bol bratom hviezdy).

Cvičenia:

1. Doplňte do programu "help", t.j., aby sa pri stlačení F1 vypísal príklad ako v zošite - čísla parašutistov a operácia medzi nimi - a po stlačení ľubovoľného ďalšieho klávesu "help" zmizol.

2. Často sa stáva, že po skončení programu zostanú v bufferi nejaké znaky - občas to býva nepríjemné (ak je po návrate do NC v bufferi ENTER, spustí sa program znova ap.). Napíšte procedúru, ktorá pred skončením programu buffer vyprázdni.

4.2 Slávičí škrek (SOUND)

Aká by to bola hra bez škrekov, výkrikov, podmazu a upokojujúcich tónov?
Aké by to bolo programovanie bez zvukov?
Nanič!
Samotné PC síce nie je dobrým a kvalitným hudobným prístrojom, no po pridaní zvukovej karty sa stáva malým zázrakom. Produkované tóny sa vyrovnajú hociktorému symfonickému orchestru alebo rockovej kapele.
A ako je to s pascalom?
Rovnako ako s PC. Bez assemblerovských modulov je schopný vyprodukovať maximálne tak škrekotavý tón potácajúcej sa vrany.
Pre prácu so zvukmi ponúka Turbo pascal dva až tri príkazy - procedúry:

```
Sound(Vyska);
NoSound;
Delay(Pauza);
```

Sound - zapne generátor zvuku na zadanú frekvenciu. Je typu ***word***, takže od 0 do 65 535 Hz. Pre ľudské ucho je počuteľný len zvuk od 16 do 16 000 Hz, takže bohato stačí.
NoSound - zastaví jednotvárne pípanie.
Delay - nie je určená priamo pre zvuky. Je to len procedúra, ktorá zabezpečí prestávku na zadaný počet milisekúnd.

Výhodou procedúry ***sound*** je, že nečaká na vypnutie, ale po zapnutí generátora zvuku sa okamžite vykonávajú ďalšie príkazy (aj bez jeho vypnutia).

*Prispôsobte počítač tak, aby sa správal ako pseudoklavír. Tóny **c-c2** budú klávesy číslicového radu. Tón sa bude hrať tak dlho, kým bude kláves stlačený.*

Zaujímavé efekty sa dajú dosiahnuť súčasným "pustením" pípania dvoma smermi - oproti sebe. Napr.:

```
for i:=1000 to 2000 do begin
 Sound (i);
 Delay (1);
 Sound (3000-i);
 Delay (1);
end;
NoSound;
```

No pri pascalovskej práci s procedúrou ***sound*** platí viac než kdekoľvek inde, že skutočne kvalitný výsledok sa dá dosiahnuť len experimentovaním.

Napíšte program - hudobno-notový editor, ktorý vám umožní vkladať do notovej osnovy noty (rôznu výšku i dĺžku) a bude schopný "skomponovanú" skladbu uložiť aj na disk.

4.3 Stratený kurzor (Skrývanie kurzora)

Občas sa stáva, že pri práci so znakmi - v CRT móde - pôsobí nemiznúci kurzor veľmi rušivo (najmä tým, že sa zobrazuje vždy za posledným výpisom) a potrebujeme ho skryť - nezobrazovať.

Ďalší prípad, keď ho potrebujeme zmeniť, prichádza po stlačení *INSERT* - kurzor by sa mal zmeniť z klasickej čiarky na vysoký obdĺžnik.

Alebo, alebo, alebo...

Ako teda meniť kurzor?

Na prvý pohľad vyzerá univerzálny spôsob hrôzostrašne, no v skutočnosti je úplne jednoduchý. Využíva jednu zo služieb DOSu - ***Intr $10*** - kde sa do registrov vkladá tvar (veľkosť kurzora) a po vykonaní funkcie sa podľa nej kurzor aj zobrazuje. ***Intr*** je súčasťou unitu DOS.

```
uses Dos;
var Reg: Registers;

procedure SkryKurzor;

begin
 reg.ah:=1;
 reg.ch:=32;
 reg.cl:=0;
 Intr ($10, Reg);
end;

procedure Prepisovaci;

begin
 reg.ah:=1;
 reg.ch:=1;
 reg.cl:=14;
 Intr ($10, Reg);
end;

Procedure Klasicky;

begin
 reg.ah:=1;
 reg.ch:=13;
 reg.cl:=14;
 Intr ($10, Reg);
end;
begin
 {zavolajte si ľubovoľnú procedúru}
```

```
end.
```

Napíšte malý textový editor (stačí, keď bude pracovať len s jednou obrazovkou), ktorý bude schopný prepisovať a vkladať znaky a kurzor sa bude meniť podľa situácie. Samozrejmé je aj ukladanie na disk.

4.4 Vláda pohodlnosti (Prehľadávanie adresára)

Pri programovaní často potrebujeme získať od užívateľa meno súboru, s ktorým chce pracovať. Môžeme to urobiť tisícimi spôsobmi, no najčastejšie sú dva:
1. Necháme ho, úbožiaka, vypísať celú cestu od koreňového adresára až po hľadaný súbor bez toho, aby naisto vedel, či jeho súbor naozaj existuje,
2. Zobrazíme aktuálny stav (t.j. obsah aktuálneho adresára).

Prvý postup je jednoduchý, ale nedôstojný človeka, ktorý programuje viac ako dva roky.
Druhý je síce dôstojný, ale aj patrične náročný (najmä keď ho chceme urobiť dokonale a profesionálne - s možnosťou pohybu šípkami, *PgUp*, *PgDown*, *Home*, *End*...).

Prvá dôležitá vec, ktorú síce nebudeme (možno) priamo potrebovať, je zistenie aktuálnej cesty, ktorá je nastavená v DOSe.

```
GetDir(0,Cesta);
```

Procedúra do premennej ***Cesta*** vloží aktuálnu cestu na aktuálnom disku (parameter 0).

Ak chceme získať súbory obsiahnuté v ľubovoľnom adresári, použijeme procedúru

```
FindFirst(Charakteristika,Atributy,Info);
```

Charakteristika - cesta + typ súborov, ktoré majú byť hľadané (*.*, *.txt, *.pas...),
Atributy - atribúty vypisovaných súborov (ReadOnly, Adresáre...),
Info - záznam, do ktorého sa vkladajú údaje o súbore. Je štandardne definovaného typu ***SearchRec*** a obsahuje: meno, veľkosť, čas vzniku...).

Procedúra ***FindFirst*** nájde prvý súbor zodpovedajúci požiadavkám. Ak neexistuje, vygeneruje sa chyba ***DosError(18)***.

Ďalšie požadované súbory hľadá procedúra

```
FindNext(DirInfo)
```

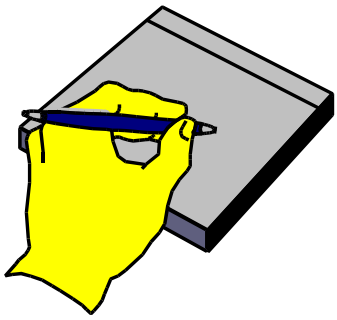
ktorá si pamätá nastavené parametre ***Findfirst***u.

Ako teda vypísať aktuálny adresár?

```
uses Dos;
var DirInfo: SearchRec;

begin
  FindFirst ('*.*', AnyFile, DirInfo);
  while DosError = 0 do begin
    WriteLn (DirInfo.Name);
    FindNext (DirInfo);
  end;
end.
```

Napíšte procedúru, ktorá vám dovolí "šantiť" po disku, t.j. dostávať sa do nadadresárov, podadresárov, prehliadať súbory a nazrieť do ľubovoľného z nich.

4.5 Ako sa neupísať k smrti (Unity)



Ak ste dlhšie programujúcim a väčšie programy píšucim programátorom, určite sa vám už stalo, že v dvoch úplne nesúvisiacich programoch ste použili rovnakú procedúru, podobnú funkciu, tie isté sekvencie a ... museli ste to isté písať dva razy.

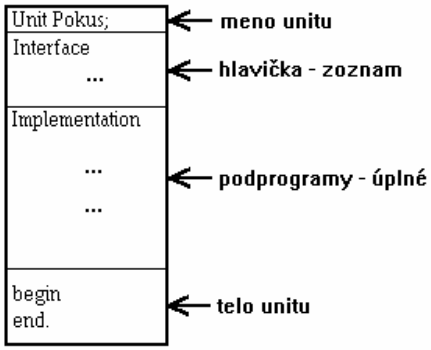
A ak ste veľké programy píšuci a stručnosti nie príliš holdujúci pascalisti, viete zasa, že pri programoch asi nad
4 000 riadkov, kompilátor vyhlasuje, že nie je schopný spracovať zdrojový text kvôli jeho veľkosti (nevojde sa do jedného segmentu - 64 kB).

Pre oba zdanlivo úplne odlišné problémy sa ponúka jediné a univerzálne riešenie - vytvoriť z procedúr programu ***unit***.

Unit je jednotkou (alebo skôr knižnicou), ktorá obsahuje procedúry a funkcie, ktoré možno volať (spúšťať) z ľubovoľného programu.

Ak máme napr. unit obsahujúci procedúry na načítavanie údajov, vykresľovanie a pohyb po menu, máme jednak silný prostriedok na vytváranie prostredia, jednak nástroj na veľké ušetrenie času.

Unit je obyčajný súbor procedúr (obyčajne napísaných v Turbo pascale). (Prácu s grafikou aj prácu s textom tiež zabezpečujú unity - ***graph*** a ***crt***.)



Skladá sa:

- z mena - názvu unitu,
- z hlavičky, ktorá obsahuje zoznam procedúr a funkcií (so všetkými parametrami), ktoré sú prístupné pre volanie z iných programov,
- z procedúr a funkcií v úplnom znení,
- z tela unitu, ktoré sa vykoná vždy, keď sa zavolá niektorý z podprogramov unitu. (Táto časť býva väčšinou prázdna.)

Ak chceme unit využívať vo svojom programe, musíme jeho meno uviesť do zoznamu *uses*ov.

pokus.pas:

Napr.:

```
  Program C1231;
  uses graph, crt, pokus;
  var a: integer;
```

Pozn.: Meno unitu a súboru, v ktorom je unit uložený, musia byť rovnaké. (Inak dochádza ku kolíziám.)

Napíšte unit pre načítavanie a vypisovanie reťazcov.

```
Unit Cit_Pis;

Interface

  Procedure Citaj (var Retazec: string);
  Procedure Pis (Retazec: string);
  Procedure Prazdny_trojriadok;

Implementation

Procedure Citaj;        {tu už netreba uvádzať parametre}

begin
 WriteLn (' Zadaj retazec ');
 ReadLn (Retazec);
 WriteLn (' Dakujem');
end;

Procedure Pis;

begin
 WriteLn ('**********************');
 Writeln (Retazec);
 WriteLn ('**********************');
end;

Procedure Prazdny_trojriadok;

begin
 WriteLn;
 WriteLn;
 WriteLn;
end;
```

```
begin
 {prázdne telo procedúry}
end.
```

Štruktúru ***unitu*** už poznáte. To je základ. Okrem procedúr a funkcií však možno v ***unite*** definovať aj typy a konštanty, ktoré bude využívajúci program potom poznať (napr. typ osoba, tabuľka...).

Vytvorte unit, ktorý bude schopný načítavať, vypisovať a spracúvať (prehľadávať, usporadúvať) zoznam hráčov futbalového klubu MUCHA.

V ***unite*** možno deklarovať (pomocou ***var***) aj vlastné (unitovské) globálne premenné, ktoré budú poznať všetky jeho procedúry, no tieto premenné nebudú dostupné programu, ktorý unit využíva.
Všetky ostatné prvky ***unitu*** sú, samozrejme, dostupné. A naopak - definície, deklarácie, procedúry a funkcie otcovského programu sú pre unit neznáme a neprístupné. Potrebné údaje sa dajú prenášať (a aj sa tak prenášajú) len v parametroch jednotlivých procedúr.
Každý program môže využívať ľubovoľný počet unitov. V prípade, že sa v dvoch unitoch vyskytne procedúra s rovnakým menom, sa patrí bližšie špecifikovať, ktorú procedúru žiadame:

`unit1.moja` alebo **`unit2.moja`**

V opačnom prípade (len ***moja***) sa vykonáva procedúra z unitu, ktorý je v zozname zapísaný ako prvý.

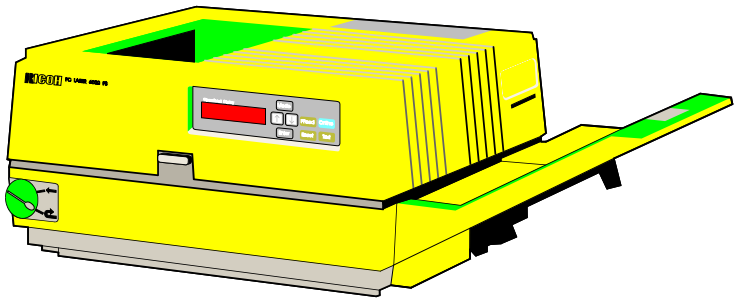
Vytvorte unit, ktorý bude schopný vytvárať a ovládať grafické - tlačítkové - menu. Parametrami možno voliť počet, typ, veľkosť, farbu, hrúbku tieňa, obrysy... tlačítok a vracať sa môže napr. kód zvoleného (vybraného) tlačítka.

Upravte unit tak, aby pri každej akcii zapískal.

Tu stačí do tela unitu vložiť príkaz zvuku - netreba prerábať všetky procedúry. Unit vykonáva najprv príkazy uvedené v tele a až potom sa pustí do prikázanej procedúry.

```
Unit Menu;

...
...
...

begin
 Sound (1000);
 delay (5);
 NoSound;
end.
```

4.6 Dajte to na papier (Tlač)



V pascale možno, okrem iného, aj tlačiť. Túto výhodu neocenia len tvrdohlaví programátori, ktorí pri vytváraní databázových systémov uprednostnia pascal pred profesionálnymi nástrojmi, ale aj bežný - normálny - programátor, ktorý potrebuje výstup namiesto na obrazovku poslať na tlačiareň.

Štruktúra tlače je jednoduchá:

```
uses Printer;
var  zn: integer;
     lst: text;

begin
 {$I-}
 Write (Lst, 'nazdar');
 {$I+}
 if IOResult<>0 then WriteLn ('Tlaciaren nekomunikuje');
end.
```

Riadok ***Write(Lst, 'nazdar')*** posiela na tlačiareň text. Zariadenie - tlačiareň - je definované v ukazovateli Lst. Takže programátor sa nemusí starať vôbec o nič. (Možno iba o kontrolu, či je tlačiareň k dispozícii - podobne ako pri súboroch - vypínaním a zapínaním ***$I***).

Upravte jednoobrazovkový textový editor tak, aby bol schopný vytlačiť na papier riadok, na ktorom sa nachádza kurzor.

Pozn.: Bežné tlačiarne - A4 - majú rovnakú šírku ako obrazovka - 80 znakov. V prípade, že posielame reťazce len príkazom ***Write*** (väčšinou - nie vždy), správajú sa rovnako ako obrazovka - po 80. znaku idú na začiatok ďalšieho riadku. Ak pošleme údaj cez ***WriteLn***, papier sa nastaví na začiatok nového riadku.

4.7 Bez klávesnice (Myš)

Myš - nástroj, ktorý mnohým uľahčuje prácu, ale aj nástroj, z ktorého majú neskúsení programátori panický strach.
A pritom je to také jednoduché...

Čo od nej potrebujeme?

1. Vidieť, kde sa nachádza,
2. Zistiť, či bola stlačená,
3. Zistiť, kde bola stlačená,

... a to je všetko. Ostatné doplnky sú už naozaj len luxusom.

O správu, kontrolu a prácu s myšou sa starajú procedúry prerušenia - ***Intr $33***.
A ako začať?
Najprv samozrejme inicializujeme grafiku.
Potom pripravíme na prácu myš. Znamená to vynulovanie (vyčistenie) registrov a nastavenie parametrov - RESET.

A môžeme začať:
- ukážeme myš užívateľovi - ***Ukaz_mys***,
- o pohyb myši sa stará systém, meniť súradnice pri pohybe zostáva na ňom,
- čakáme na stlačenie "očka" - ***Pravy***
Lavy
Stredny,
- a zistíme, kde bola myš stlačená - ***Kde_je_mys (x, y)***,
- ak chceme meniť obrazovku (kresliť na ňu), treba myš skryť - ***Zrus_mys***. Keby sme kurzor neodstránili, mohlo by sa stať (a veľmi často sa aj stáva), že ho prekreslíme a pri jeho ďalšom pohybe - odsunutí - vzniknú nepríjemné narušenia (grafické - chýbajúce body na mieste kurzora, farebný chaos...).

Ďalšie funkcie môžu byť napr.:
Nastav_mys (x., y),
Nastav_okno (xz, yz, xk, xy) - nedovolí kurzoru vyjsť z nastaveného okna
atď.

O ovládanie a prácu s myšou sa obyčajne starajú napísané unity, ktorých po svete koluje hrozne veľa a každý z nich je charakteristický väčšími alebo menšími službami - v podstate však ide vždy o to isté.

Na ilustráciu jeden zo základných:

```
unit Mysiak;
uses Dos;
var r: Registers;


Interface

Procedure Reset_mys;
Procedure Ukaz_mys;
Procedure Zrus_mys;
Procedure Okno_mys (x1, y1, x2, y2: integer);
Procedure Kde_je_mys (var x, y: integer);
Procedure Nastav_mys (x, y: integer);
Function Lavy: boolean;
Function Pravy: boolean;
Function Stredny: boolean;


Implementation

Procedure Reset_mys;
begin
 r.ax:=$0000;
 intr ($33, r);
end;

Procedure Ukaz_mys;
begin
 r.ax:=$0001;
 intr ($33, r);
end;

Procedure Zrus_mys;
begin
 r.ax:=$0002;
 intr ($33, r);
end;

Procedure Okno_mys (x1, y1, x2, y2: integer);
begin
 r.ax:=$0007;
 r.cx:=x1;
 r.dx:=x2;
 intr ($33, r);

 r.ax:=$0008;
 r.cx:=y1;
end;

Procedure Kde_je_mys;
begin
 r.ax:=$0003;
 intr ($33, r);
 x:=r.cx;
 y:=r.dx;
```

```
 end;

 Procedure Nastav_mys;
 begin
  r.ax:=$0004;
  r.cx:=x;
  r.dx:=y;
  intr ($33, r);
 end;

 Function Lavy: boolean;
 begin
  r.ax:=$0005;
 r.bx:=0;
  intr ($33, r);
  if (r.ax and 1)=1 then begin
  Lavy:=true;
  r.bx:=3;
  end
  else Lavy:=false;
 end;


Function Pravy: boolean;
begin
 r.ax:=$0005;
 r.bx:=1;
 intr ($33, r);
 if (r.ax and 2)=2 then begin
 Pravy:=true;
 r.bx:=3;
 end else Pravy:=false;
end;

Function Stredny: boolean;
begin
 r.ax:=$0005;
 r.bx:=2;
 intr ($33, r);
 if (r.ax and 4)=4 then Stredny:=true
 else Stredny:=false;
end;

begin
end.
```

Záver

Táto učebnica nie je klasickou učebnicou písanou so slovníkom cudzích slov. Snažili sme sa v nej priblížiť čo najbližšie ku generácii mladých, takmer úplne od základov začínajúcich programátorov orientovaných na pascal. Snažili sme sa poskytnúť dostatok príkladov na motivačnej úrovni nie pre vysokoškolákov, ale pre normálne, hravé, po rozptýlení túžiace ešte takmer deti. Snažili sme sa zaoberať aj typickými problémami, ktorým sa väčšina ľudí radšej vyhne, pretože ich nepozná a má z nich strach. Nejde tu len o železá (myš, tlačiareň), ale aj o problémy s dynamickými premennými, rekurziou, binárnymi stromami atď.
Možno sa celá učebnica nesie v tom duchu, že každý by mal byť programátorom, ale to je len zdanie. Nie každý na to má a nie každého to baví. Prvé zaujatie je však veľmi dôležité a snažili sme sa hlavne o to.
Veríme, že Turbo pascal (netradične a v príkladoch) aspoň niekomu niečo dá a tých niekoľko nocí sme nad ním nestrávili zbytočne.

Doplnkové cvičenia

1. Tetris

Zabezpečte: - otáčanie padajúcich útvarov,
- možnosť nastavenia viacerých úrovní obtiažnosti,
- zrýchľovanie podľa počtu poskladaných riadkov,
- evidovanie 10 najlepších výsledkov.
Rozšírenie: - určte ako cieľ hry odkrytie obrázku (pri každom poskladanom riadku sa časť obrázka odkryje).

2.Piškvorky

Zabezpečte: - možnosť hry pre 2-4 hráčov súčasne,
- možnosť nastaviť víťazný počet (3-7),
- schopnosť programu rozpoznať ukončenie a určiť víťaza
Rozšírenie: - ovládanie hry myšou,
- naprogramujte ako súpera počítač.

3. Puzzle

(Ide o hru, v ktorej máte k dispozícii hraciu plochu s rozmermi 4x4 štvorce. Obsahuje 15 štvorcov s číslami 1-15 a jedno prázdne miesto. Cieľom je poskladať štvorce v poradí

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 .)

Zabezpečte: - zamiešanie štvorcov z poskladanej polohy (pri náhodnom rozložení by sa mohlo stať, že naďabíte na situáciu, ktorá nie je riešiteľná),
- rozpoznanie úspešného konca,
- počítanie ťahov a evidovanie najlepších výsledkov.
Rozšírenie: - upravte ovládanie na myš,
- dovoľte vrátiť o ťah nazad.
Maximálne rozšírenie: - nájdite algoritmus, ktorý bude schopný rozhádzanú hraciu plochu poskladať.

4. ***Puzzle 2***

Pravidlá sú rovnaké ako v predchádzajúcom príklade, no neskladáte čísla, ale obrázok. Ten môže byť rozložený na ľubovoľný počet častí (rovnakého tvaru a veľkosti).
Pri skladaní poskytnite zmenšený návod v rohu obrazovky.

Zabezpečte: - ako v predchádzajúcom príklade,
- možnosť výberu z viacerých obrázkov (napr. načítaním z disku).
Rozšírenie: - ako v predchádzajúcom príklade.

5. *Človeče, nehnevaj sa.*

Zabezpečte: - grafické znázornenie hry,
- možnosť hrať pre 2-4 hráčov, prípadne proti počítaču.
- náhodné generovanie pri hádzaní kockou,
rozpoznanie konca.
Rozšírenie: - efektný súboj pri vyhadzovaní,
- zvuková kulisa pri hádzaní kockou.

6. ***Tenis***

(Na začiatku stačí, ak na každej strane obrazovky bude obdĺžnik (čiara), ktorý bude odrážať loptu.)

Zabezpečte: - počítanie správnych a nesprávnych "lôpt",
- rozdelenie na sety.
Rozšírenie: - hra proti počítaču (naprogramujte algoritmus, ktorý nebude neomylný ani neporaziteľný).

7. Ruleta

Zabezpečte: - grafickú hraciu plochu,
- možnosť zadávania nielen na čísla, ale aj na farby, paritu, stĺpec alebo riadok,
- kontrolu vašej peňaženky.
Rozšírenie: - grafické zobrazenie otáčanie hracieho kolesa.

****

8. Pomocník pri rozmiestňovaní nábytku v byte

Zabezpečte: - vytvorenie zoznamu nábytku,
- možnosť presúvania jednotlivých kusov,
- pohľad do bytu z pôdorysu,
Rozšírenie: - presúvanie nábytku myšou.

9. Slovensko-anglický slovník

Zabezpečte: - evidovanie slov v súbore,
- zoraďovanie podľa abecedy jedného z jazykov.
- vyhľadávanie zadaného slova,
- opravy, vsúvanie, mazanie slov.
Rozšírenie: - vyrobte procedúru, ktorá dokáže preložiť anglický test do slovenského (len v základnom tvare).

10. Program - diár.

Zabezpečte: - evidovanie mien, adries, telefónu,
- vyhľadávanie údajov,
- hodiny (ukazuje dátum a čas),
- budík (pípanie v nastavenú dobu).
Rozšírené: - diár na každý deň rozdelený na polhodiny.

11. Simulátor križovatky so semaformi

Zabezpečte: - grafické znázornenie križovatky,
- náhodný príchod áut za štyroch strán,
- ich pohyb podľa svetiel na semafore.
Rozšírené: - v prípade zápchy prestavenie semaforov podľa potreby.

12. Unit pre animáciu

Zabezpečte: - základné procedúry (vykreslenie, zmazanie, posun),
- pohyb obrázkov (aj viacerých súčasne),

13. Prehľadávač disku

Zabezpečte: - hľadanie zadaného súboru na celom disku,
- použitie masky pre hľadanie.
*Rozšírené: - prehľadávanie súborov typu **arj**, **rar**, **zip**.*

14. Viacsúborový editor

Zabezpečte: - zobrazenie dvoch súborov súčasne,
- možnosť ich editovania,
- kopírovanie častí medzi nimi,
- možnosť pridávať jeden na koniec druhého.

OBSAH