Technická Univerzita v Liberci

Fakulta strojní **Katedra aplikované kybernetiky**

Prof. Ing. Miroslav Olehla, CSc.

Počítače a programování – VD

Computers & programming - VD

Studijní podklady

Liberec 2011

© Prof. Ing. Miroslav Olehla, CSc.
Lektoroval: Doc.Ing. Libor Tůma, CSc.

ISBN 978-80-7372-730-7

OBSAH:

Předmluva:

Předkládaný učební text je určen studentům 1. ročníku fakulty strojní v Liberci a Mladé Boleslavi jako **pomůcka a doplněk přednášek** při studiu předmětu "Počítače a programování". Hlavní důraz je kladen na praktické zvládnutí algoritmizace formou vývojových diagramů a usnadnit tak přechod k využívání strukturovaného programovacího jazyka, například TURBO PASCAL. Snahou je provést skripta ve formě studentům co nejbližší a jsou proto opatřena značným počtem příkladů VD.

1. Úvod

Vývojový diagram je symbolický algoritmický jazyk, který se používá pro názorné zobrazení algoritmu zpracování informací a jeho používání je vhodné zejména u začínajících programátorů. Symboly vývojových diagramů představují grafické značky přesně definovaného významu. Pomocí vývojových diagramů (VD) lze pro studenty, kteří se nesetkali s některým programovacím jazykem, usnadnit pochopení sestavení programů - algoritmizace. **Po pochopení principů algoritmizace je vhodnější zapisovat programy přímo ve zvoleném programovacím jazyce bez využívání VD**. Poznamenejme, že u vybraných VD je uvedeno ekvivalentní vyjádření algoritmu v programovacím jazyce Pascal.

2. Jednoduché příkazy

2.1Základní značky u vývojových diagramů

Sestavení algoritmu řešení úlohy na počítači je základní a rozhodující tvůrčí činnost při programování. Algoritmizace úlohy tvoří hlavní, nikoli však jedinou etapu programování. Před vlastní algoritmizací úlohy, ať již vědecko technické nebo ekonomické, je třeba úlohu matematicky formulovat, vytvořit vhodný model či zvolit vhodnou numerickou metodu. Po sestavení algoritmu je pak třeba přepsat algoritmus do zvoleného programovacího jazyky. Na prohřešky proti pravidlům pro zápis programu nás upozorní překladač (zjednodušeně počítač). Logické chyby, které způsobí jiný průběh výpočtu, než jaký požadujeme, je třeba hledat ověřováním programu, tzv. laděním. V této fázi porovnáváme výsledky ověřovacích výpočtů s výsledky známými a odstraňujeme příčiny neshod úpravami algoritmu a programu.

Věnujeme nyní pozornost algoritmizaci jednoduché úlohy, sečtení několika čísel. V této kapitole sice ještě čtenář nezná některý programovací jazyk, ale dostačuje, aby si pamatoval příslušné ekvivalenty k algoritmům ve tvaru vývojového diagramu. Po pečlivém studiu této kapitoly by měl být již schopen sestavit, i když ne příliš „elegantně“, většinu jednoduchých úloh.

Stanovme si za úkol určit součet pěti čísel, např. 3 + 2 +1 + 7 + 4. Je zřejmé, že v případě určení součtu jiných pěti čísel, např. 5 + 8 + 12 + 7 + 5, je nutné výpočetní postup sčítání opakovat. V případě, že bychom tuto sumaci prováděli často, bylo by vhodnější nepoužívat kapesní kalkulátor, nýbrž počítač, a sumaci provádět na proměnných, např. následovně: $S = A + B + C + D + E$. Místo konkrétních čísel jsme použili (podobně jako v matematice) proměnné, jimž musí být před zahájením výpočtu přiřazeny hodnoty. Proměnnou chápeme jako paměťové místo, které má své jméno, a do něhož lze uložit hodnotu.

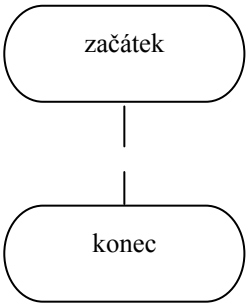
Na proměnnou A, B, C, D, E bychom uložili vždy požadované hodnoty, které máme zpracovat. Výpočetní postup – algoritmus – bychom zapsali např. takto:

1. Na proměnnou A ulož, dosaď, přečti číslo 3
2. Na proměnnou B ulož číslo 2
3. Na proměnnou C ulož číslo 1
4. Na proměnnou D ulož číslo 7
5. Na proměnnou E ulož číslo 4
6. Sečti obsahy proměnných $A + B + C + D + E$ a výsledek ulož na proměnnou S
7. Vytiskni (zobraz na obrazovce) obsah proměnné S, což je výsledek.

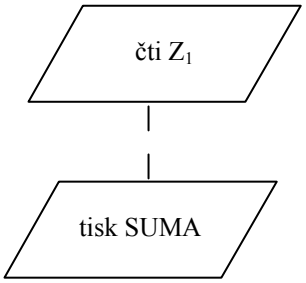
Popis algoritmu není příliš přehledný. Proto byly navrženy určité postupy, zlepšující přehlednost a usnadňující vyjádření výpočetního postupu. Jednou z možností je použití vývojových diagramů.

Vývojový diagram začíná značkou označující začátek diagramu a končí značkou označující konec. Přerušení vývojového diagramu a pokračování na jiném místě musí být označeno spojkou.

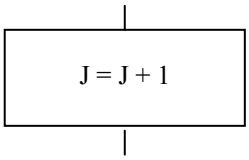
příkaz pro začátek a konec



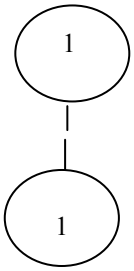
příkaz pro čtení a tisk



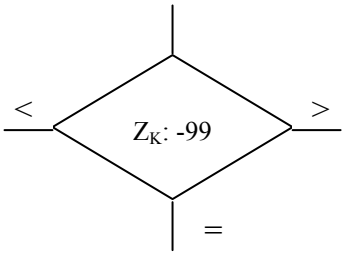
příkaz přiřazení



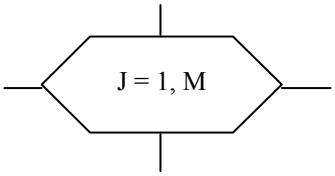
Spojka - přechod na část programu



podmíněný příkaz

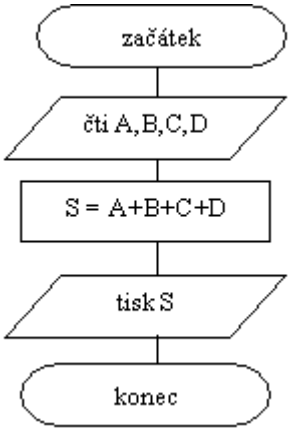


modifikace



2.2Možnosti součtu prvků řady čísel

Náš dřívější příklad součtu obsahů pěti proměnných má pak tento tvar:



Data programu jsou čtena v pořadí 3, 2, 1, 7, 4.

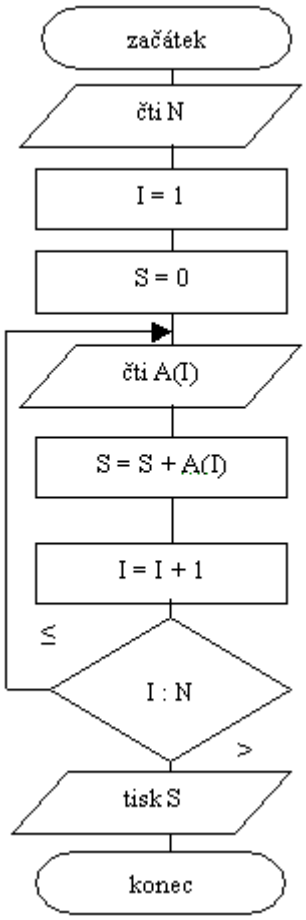
Počítač v tomto případě postupuje stejným způsobem, jako je zobrazeno ve vývojovém diagramu. Provede první příkaz, tj. čtení na proměnnou *A*, *B*, *C*, *D*, *E*.

Dalším příkazem je sečtení obsahu proměnných *A*, *B*, *C*, *D*, *E* a uložení výsledku na proměnnou *S*. Konečným příkazem je vytištění obsahu proměnné *S*, kde je uložen výsledek součtu obsahu proměnných *A* až *E*.

Pomocí uvedeného postupu je možno sečíst pětici jakýchkoli čísel. V případě, že je požadováno sečíst jiný počet čísel, např. šest čísel, deset čísel, tři čísla, atd., bylo by nutné vývojový diagram, resp. program, vždy přepsat. Je však možno použít indexovaných proměnných a vývojový diagram má pak tvar uvedený na následujícím VD.

Základní myšlenka spočívá v tom, že do proměnné *S*, které jsme na počátku přiřadili počáteční hodnotu 0, postupně v cyklu přičítáme hodnoty proměnných prvků vektoru *A*, tj. a_1, a_2, …, a_n.

Data jsou čtena v pořadí 5, 3, 2, 1, 7, 4, tzn., že v daném případě se přiřadí na $N = 5$, na $A(1) = 3$, $A(2) = 2$, $A(3) = 1$, $A(4) = 7$, $A(5) = 4$.

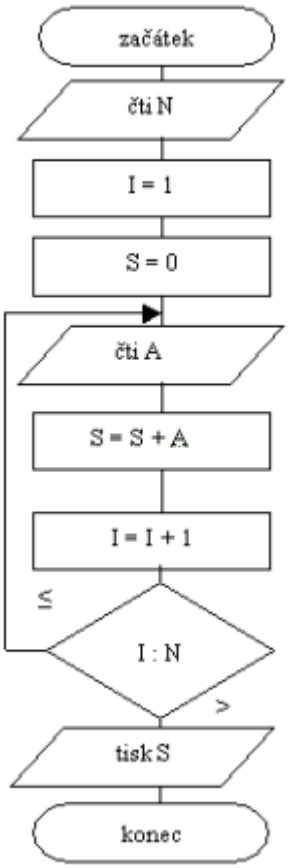



```
var N,I:integer;
S:real;
A:array [1..10] of real;
label L1;

begin
read (input,N);
S:=0;
I:=1;
L1: read(input,A[I]);
S:=S+A[I];
I:=I+1;
If I<=N goto L1
writeln(output,‘SUMA: ‘,S);
end.
```

nebo použitím modifikace:

```
var N,I:integer;
S:real;
A:array [1..10] of real;

begin
read (input,N);
S:=0;
for I:=1 to N do
begin
read(input,A[I]);
S:=S+A[I];
end;
writeln(output,‘SUMA: ‘,S);
end.
```

Pozornějšího čtenáře jistě překvapilo, proč je vlastně použita paměť pro všechny prvky pole *A*. Ve skutečnosti není pro takto zapsaný program zapotřebí ukládat a pamatovat celou řadu čísel, pak je možno algoritmu přepsat na následující tvar:

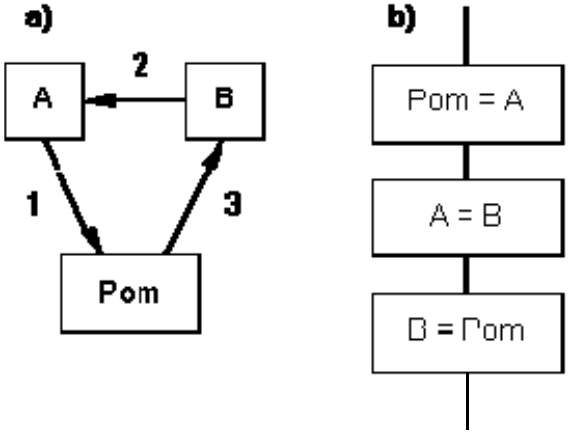


Proměnná *I* neslouží nyní k modifikaci (zvyšování indexu o 1), nýbrž jen jako počítadlo průchodu cyklem (několikrát opakovaná skupina příkazů).

```
var N,I: integer;
S, A: real;
label L1;

begin
read (input,N);
I:=1;
S:=0;
L1: read(input,A);
S:=S+A;
I:=I+1;
if i<=N goto L1;
writeln(output,‘SUMA: ‘,S);
end.
```

3. Výměna obsahu dvou proměnných

Operace je znázorněna na následujícím obrázku. V prvním kroku definujeme pomocnou proměnnou *Pom*, do které v prvním kroku přesuneme obsah proměnné *A*. Ve druhém kroku přesuneme do proměnné *A* hodnotu proměnné *B* a ve třetím kroku hodnotu z proměnné Pom přesuneme do proměnné *B*.

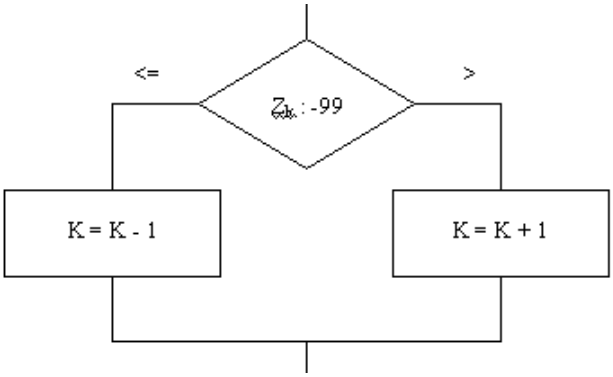


4. Strukturované příkazy

4.1 Větvení

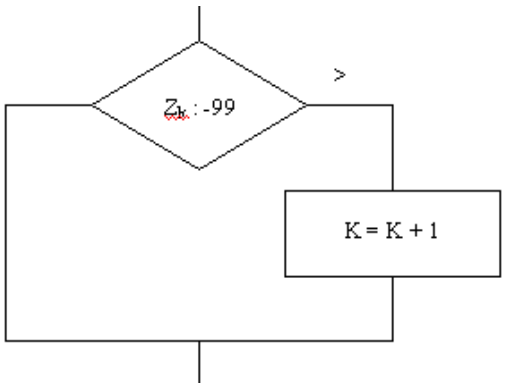
úplné

if Z[K]<=-99 then K:=K-1 else K:=K+1



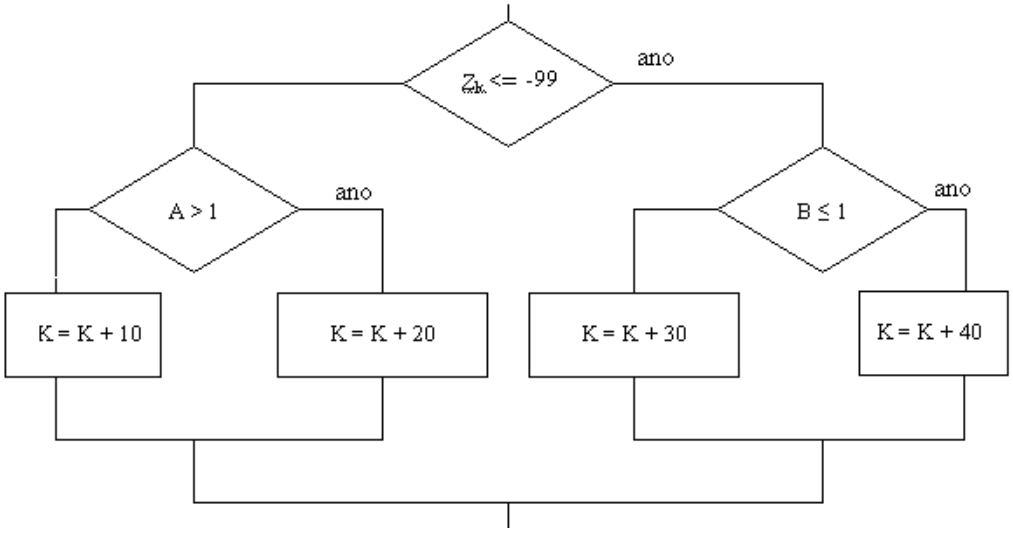
neúplné

if Z[K]>-99 then K:=K+1



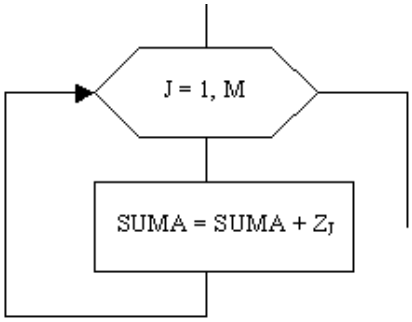
vnořené

if Z[K]<=-99 then if A>1 then K:=K+20 else K:=K+10 else if B<=1 then K:=K+40 else K:=K+30



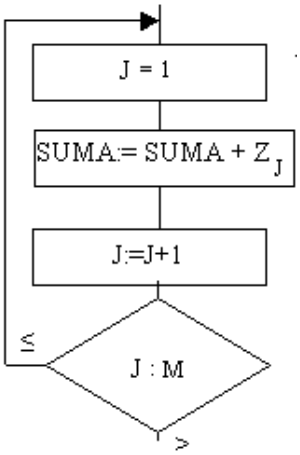
4.2 Cykly

cyklus s výčtem hodnot



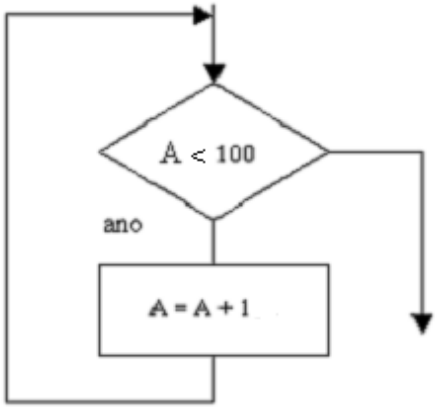

```
for J:=1 to M do
SUMA:=SUMA+Z[J];
```

Tento cyklus s výčtem hodnot nahrazuje méně přehledný zápis pomocí větvení (podmiňovacího příkazu)



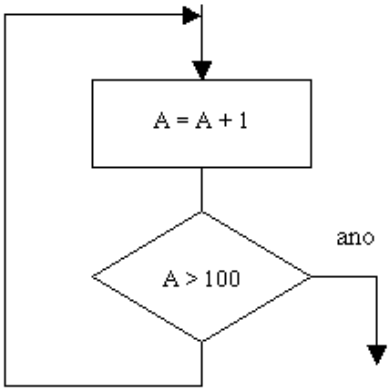

```
J:= 1;
L1: SUMA:=SUMA+Z[J];
J:=J+1;
if J<=M then goto L1;
```

cyklus s podmínkou na počátku




```
while A<100 do
A:=A+1;
```

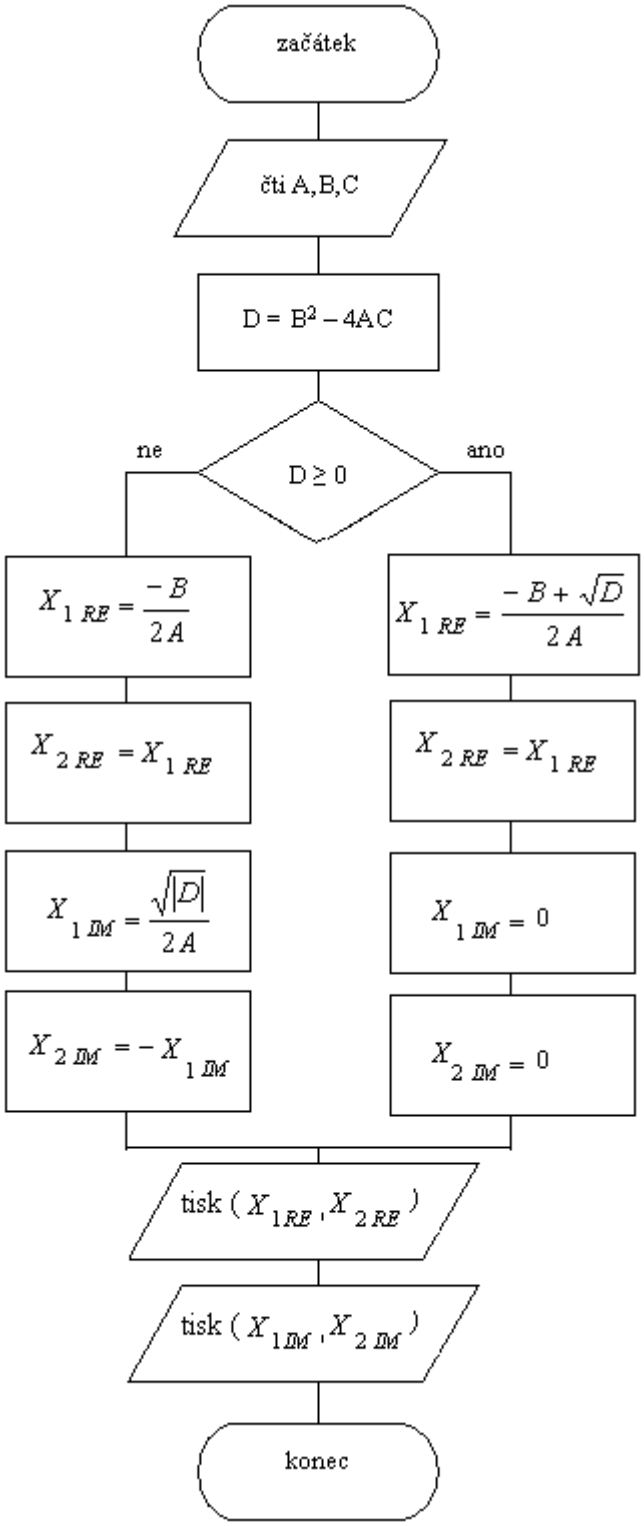
cyklus s podmínkou na konci




```
repeat
A:= A+1
until A>100;
```

5. Výpočet kořenů kvadratické rovnice

Sestavme algoritmus pro výpočet kořenů kvadratické rovnice $ax^2 + bx + c = 0$. Pro výpočet kořenů platí známý vztah: $X_{12} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$.

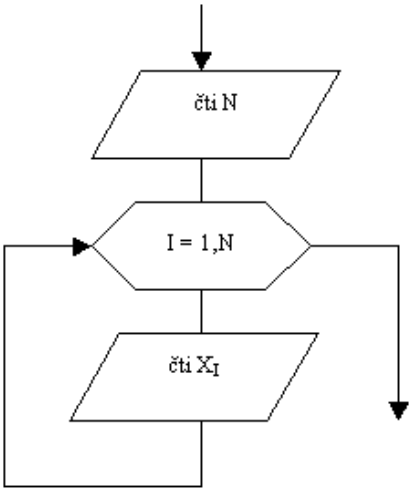


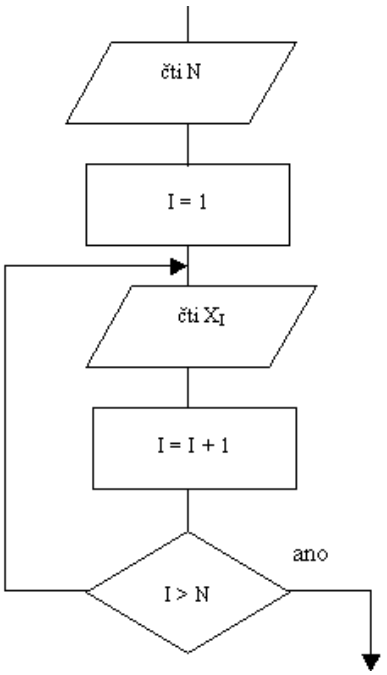

```
var A,B,C,D,X1RE,X2RE,X1IM,X2IM: real;

begin

 read(input,A,B,C);


D:=sqrt(B)-4*A*C;


if(D>=0) then


begin
   X1RE:=(-B+sqrt(D))/2*A);X1IM:=0;
   X2RE:=(-B-sqrt(D))/(2*A); X2IM:=0;
 end

else


begin
   X1RE:=-B/(2*A); X2RE:=X1RE;
   X1IM:=sqrt(abs(D))/(2*A);        X2IM:=-
X1IM;
  end;


writeln('X1RE: ',X1RE, 'X1IM: ',X1IM,
         'X2RE: ',X2RE , 'X2IM: 'X2IM);


    end.
```

6. Čtení posloupnosti dat

6.1 Čtení dat v posloupnosti o N prvcích

Posloupnost má pak obecný tvar: N, X_1, X_2, …..X_N. První se přečte hodnota počtu prvků řady a uloží do proměnné N. Příkazem cyklu s parametrem I, s krokem 1 a konečnou hodnotou N budou postupně data ukládána do proměnných X_1, X_2, ….., X_N. Parametr I bude tedy využit jako počítadlo průchodů cyklem a jako index prvků dané posloupnosti X.

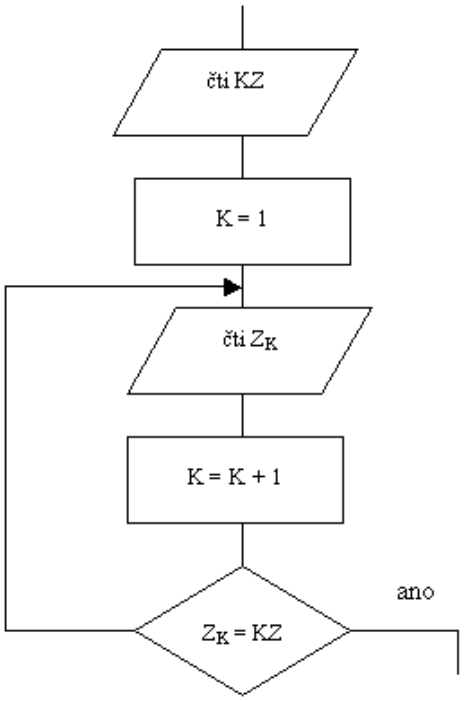




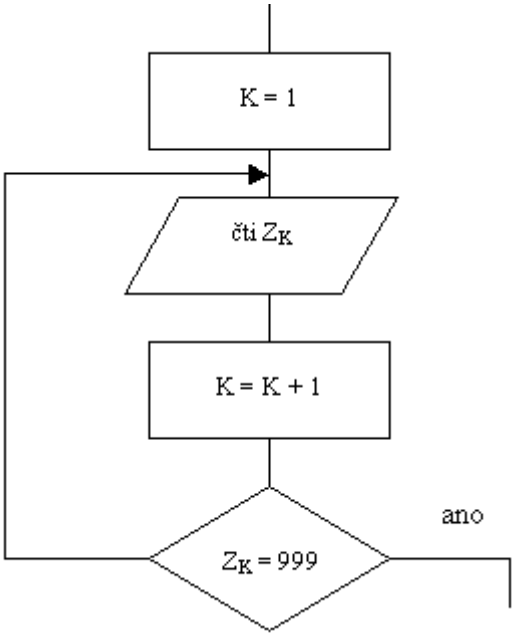
6.2 Čtení dat v posloupnosti s koncovým znakem

Tuto posloupnost označíme koncovým znakem tj. na konec posloupnosti přidáme předem definovanou hodnotu - koncový znak. Koncový znak nesmí mít hodnotu, která by se vyskytla ve čtené posloupnosti.

Koncový znak načten jako první hodnota

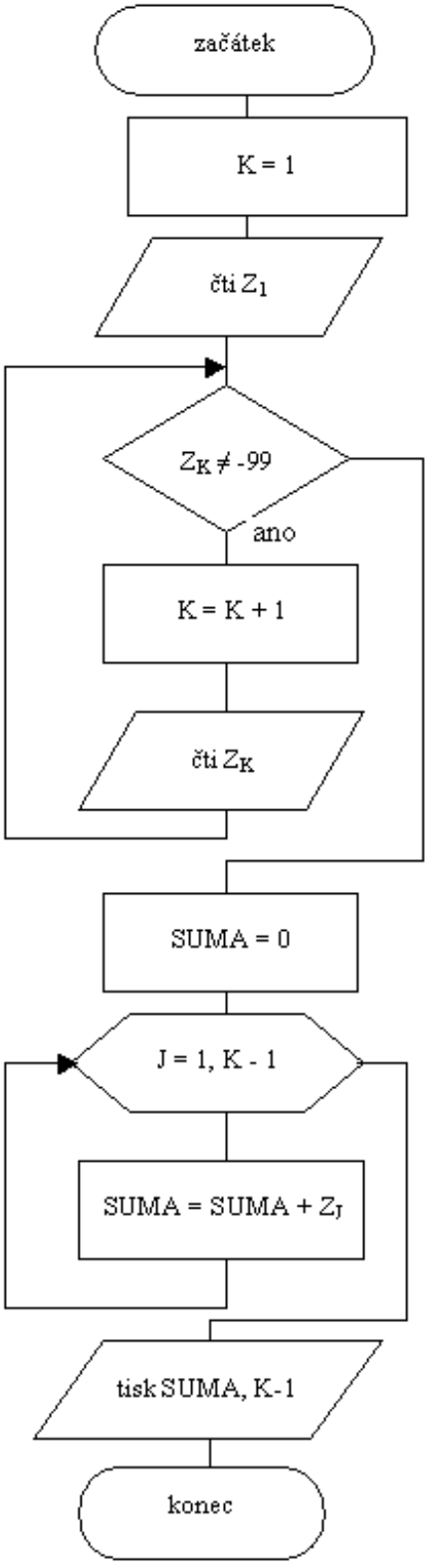


Koncový znak 999 je definován v programu



6.3 Čtení a součet prvků v souboru dat

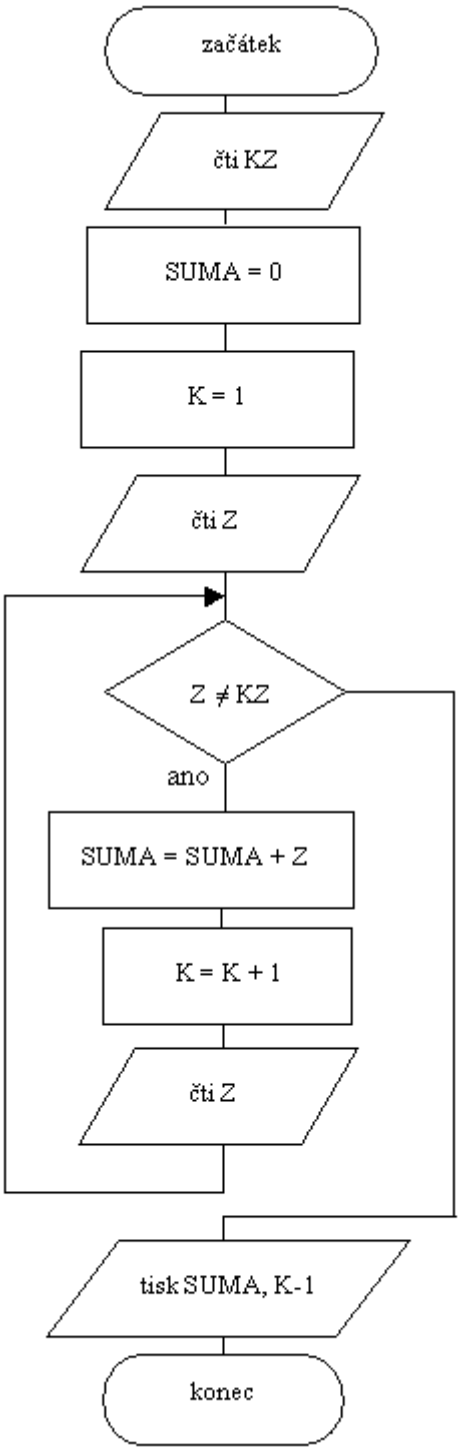
- Čtení souboru dat s pevně definovaným koncovým znakem a následným součtem, data zůstávají zachována.



U uvedeného vývojový diagram je definován koncový znak - hodnota -99. Čtení dat je prováděno pomocí příkazu cyklu kde jako výchozí hodnota počítadla cyklů je $K = 1$. První přečtená hodnota z daného souboru dat je uložena do proměnné Z_1 a pak následuje prověření shody přečtené hodnoty s definovaným koncovým znakem. Tento cyklus čtení vstupních dat pokračuje tak dlouho, dokud přečtená hodnota není rovna koncovému znaku. Protože tato poslední hodnota již do daného souboru nepatří, je nutné snížit hodnotu K, která udávala počet přečtených dat, o jednu. V daném případě je tato hodnota, udávající skutečný počet dat, uložena do proměnné M. Hodnoty vstupního souboru zůstávají uloženy v poli **Z** pro případné další zpracování.

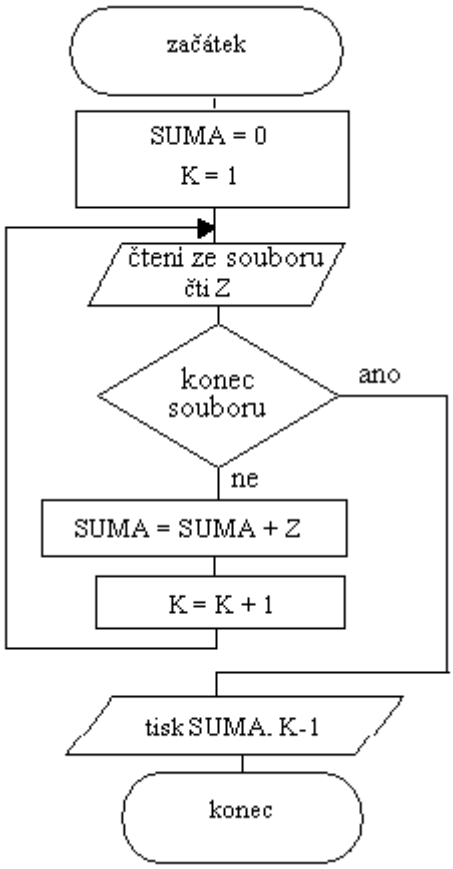
```
var K,J:integer;
   SUMA:real;
   Z:array[1..100] of real;
   label L1;
begin
K:=1; read(Z[1]);
L1:if Z[K]<> -999 then begin
                K:=K+1;
                read (Z[K]);
                goto L1;
                end;
 SUMA:=0;
 for J:=1 to K-1 do
 SUMA:=SUMA+Z[J];
 writeln(SUMA,'  ',K-1)
 end.
```

- Čtení dat se zadaným koncovým znakem a současným vytvářením součtu, data se pro další použití nezachovávají.



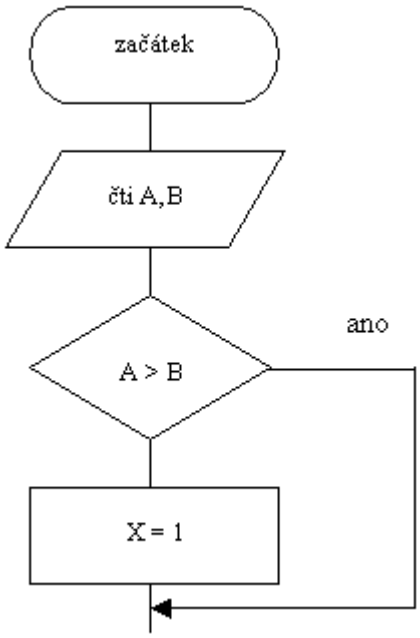
Na VD je koncový znak *KZ* načten jako první hodnota, která má význam koncového znaku. Další přečtená hodnota představuje první hodnotu zpracovávaného souboru. Protože nepředpokládáme uchovávání dat pro případné další zpracování, je použita jednoduchá proměnná *Z*. V následném cyklu s podmínkou na počátku je postupně vytvářen součet a čteny další hodnoty vstupního souboru. Tento cyklus se opakuje tak dlouho pokud přečtena hodnota není rovna koncovému znaku. Při splnění uvedené podmínky se cyklus ukončí a vytiskne se hodnota získaného součtu.

Poznámka:
V případě zápisu uvedeného algoritmu v programovacím jazyce, například v PASCALu, je možno využít příkazu „konec souboru“ *eof (jméno_souboru)* nebo „konec řádku“ *eoln (jméno_souboru),* respektive funkce *seekEof (x), seekEoln (x)* (pouze pro textové soubory, ignorují se tabulátory a mezery).

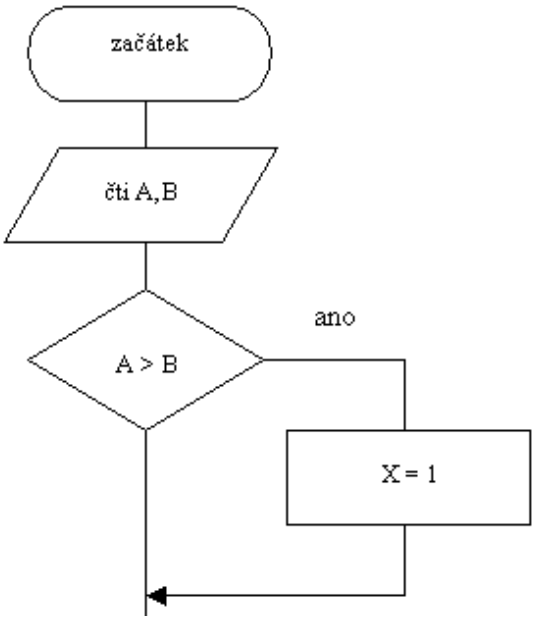


7. Příklad nevhodného zápisu VD

U následujícího schématu nelze realizovat strukturovaný program



Následující změnou lze dosáhnout, že program bude strukturovaný

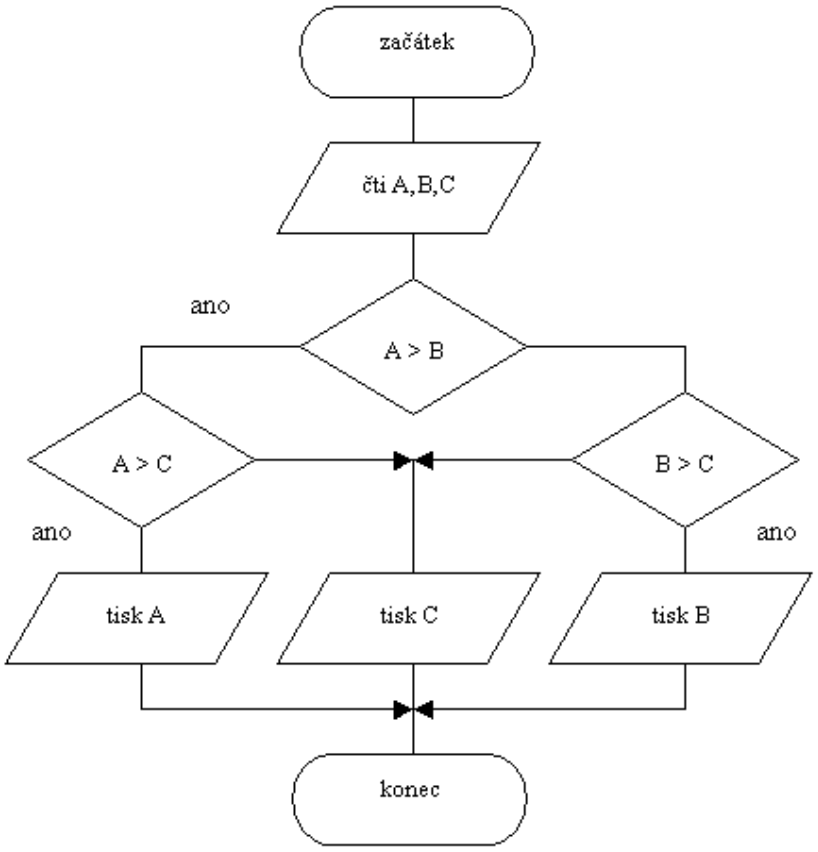



```
readln(A,B);
if A>B then goto L1 else X:=1;
L1:……..
```

```
readln(A,B);
if A>B then X:=1;
```

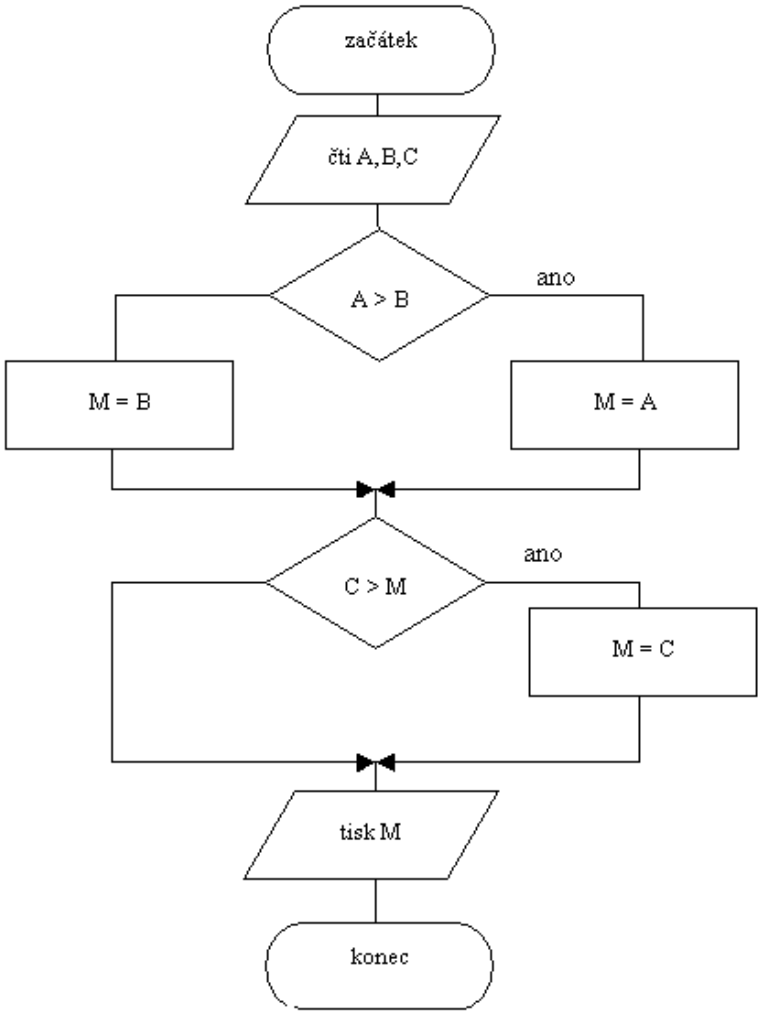
8. Vyhledání maximální hodnoty ze tří čísel

U následujícího schématu nelze realizovat strukturovaný program

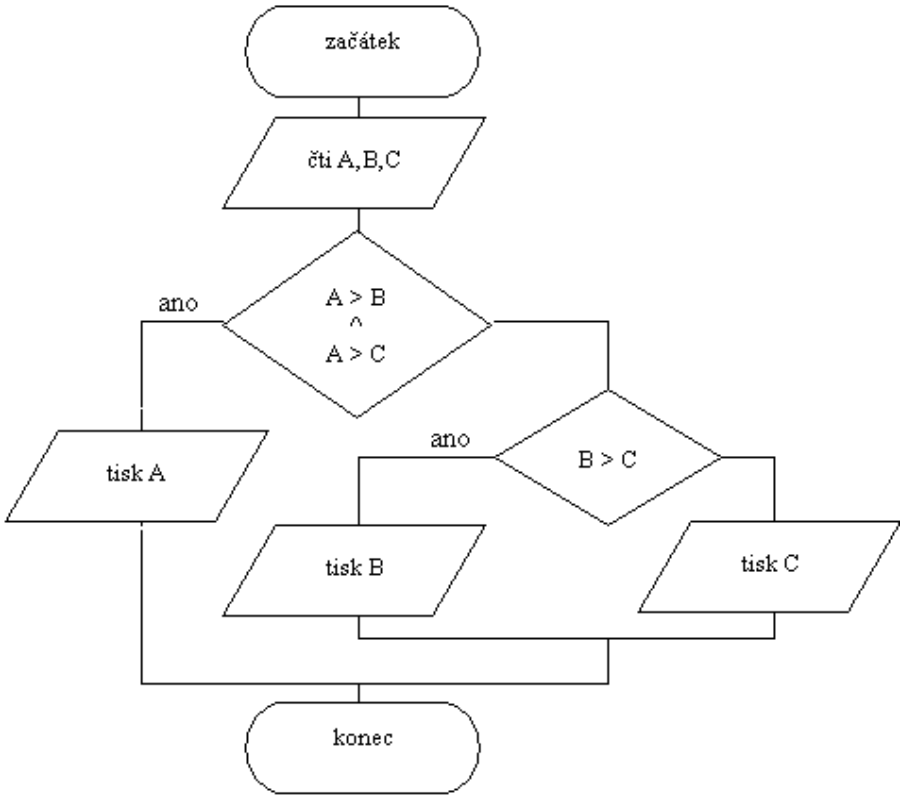



```
var A,B,C : integer;
label L1,L2;

begin
read (A,B,C);
if  A>B then    if A>C then
                  begin
                  writeln(A);goto L1
                  end
                else
                  goto L2
          else
                if B>C then
                  begin
                  writeln(B);goto L1
                  end
                else
                L2:writeln(c);

L1: end.
```

Následující změnou – u obou verzí - lze dosáhnout, že program bude strukturovaný



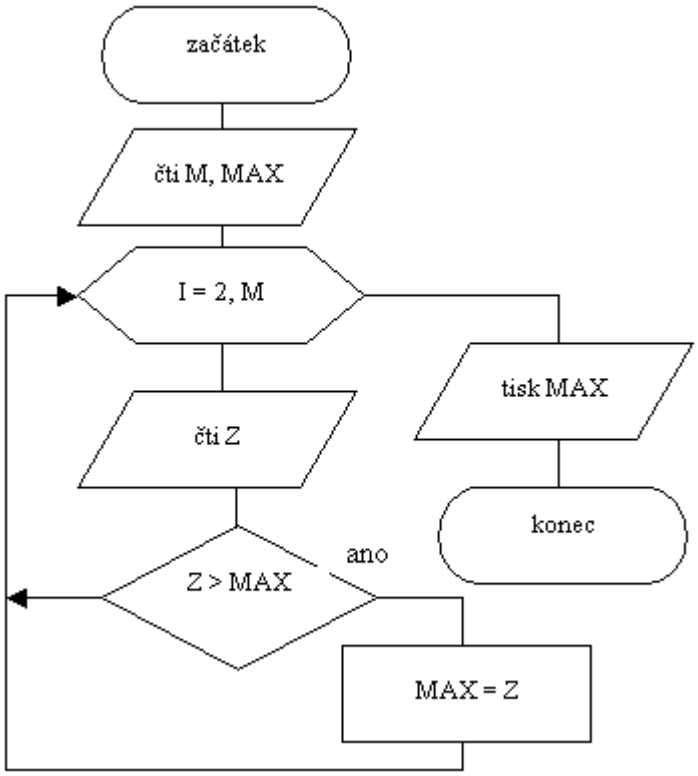

```
var A,B,C,M: integer;

begin
read (A,B,C);

if  A>B then   M:=A else M:=B;

if C>M then  M:=C;

writeln(M);

end.
```



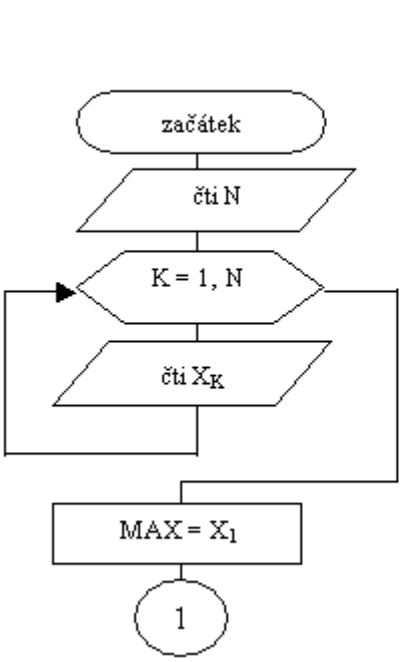

```
var A,B,C : integer;

begin
read (A,B,C);

if  (A>B) and (A>C) then
writeln(A)

else
if B>C then  writeln(B)

else writeln(C);

end.
```

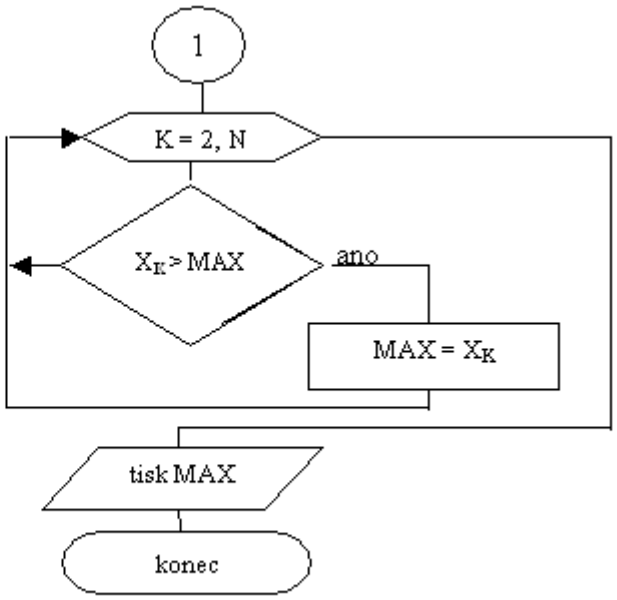
8.1 Nalezení maximálního čísla ze zadané množiny čísel

Čtení hodnot spojeno s výběrem, původní hodnoty prvků zadané množiny se nezachovají

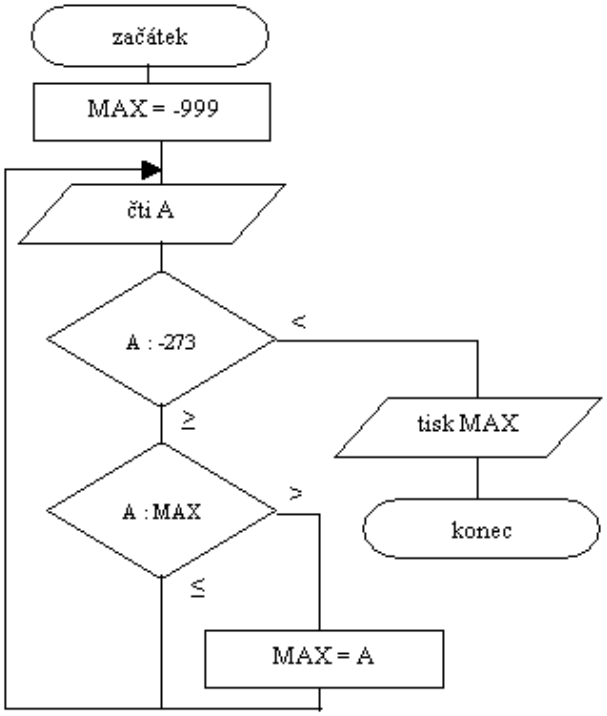


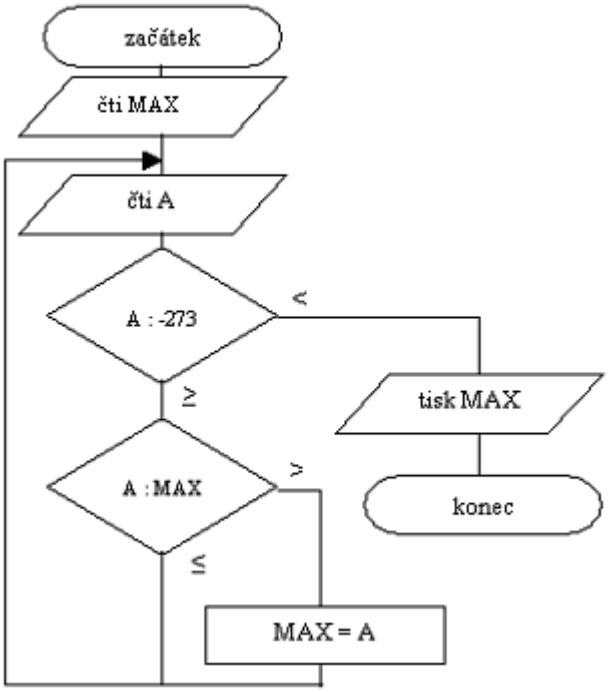
Jinou možností je oddělené čtení hodnot řady čísel X_K, původní hodnoty prvků zadané množiny se tímto zachovají



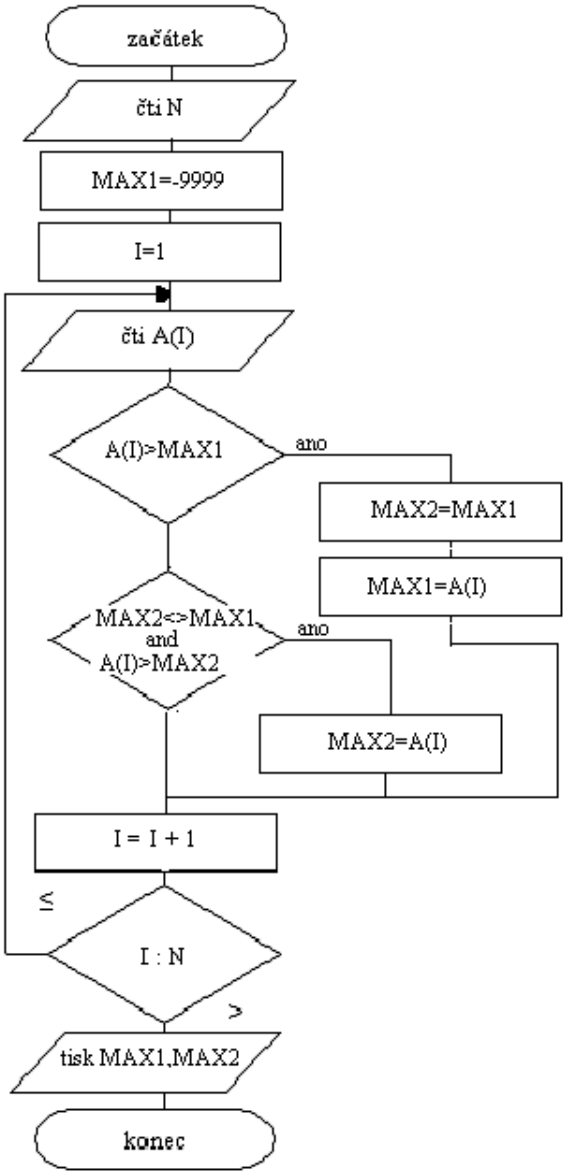


Možnost čtení s ukončením na koncovou značku –273 je uvedena v následujícím VD. Druhá varianta umožní volbu počáteční hodnoty *MAX*.





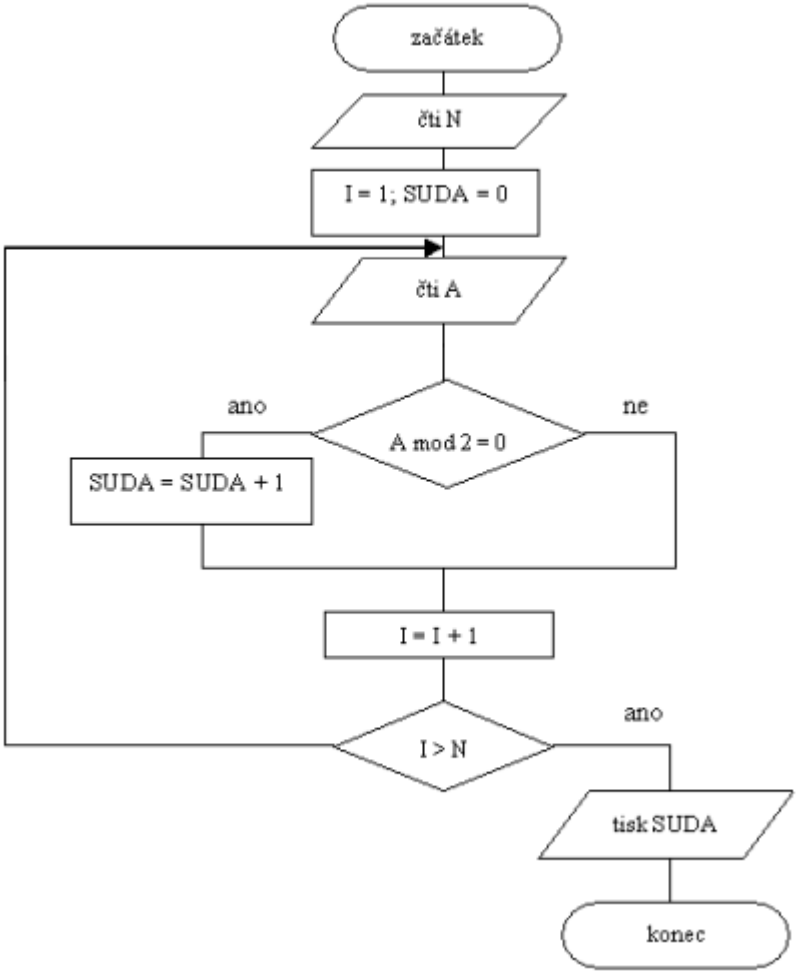
8.2 Nalezení dvou největších prvků z řady čísel




```
var N,I: integer;
   MAX1,MAX2:real;
   A: array [1..10] of real;

begin
max1:=-999;
read(N);

for I:=1 to N do
begin
  Read(A[I]);
  if A[I]>MAX1
  then begin MAX2:=MAX1;MAX1:=A[I] end
  else
if (MAX2<>MAX1) and (A[I]>MAX2) then
MAX2:=A[I];
end;

writeln(max1,max2);
end.
```

9. Načtení *n* celočíselných čísel a zjištění, kolik je sudých

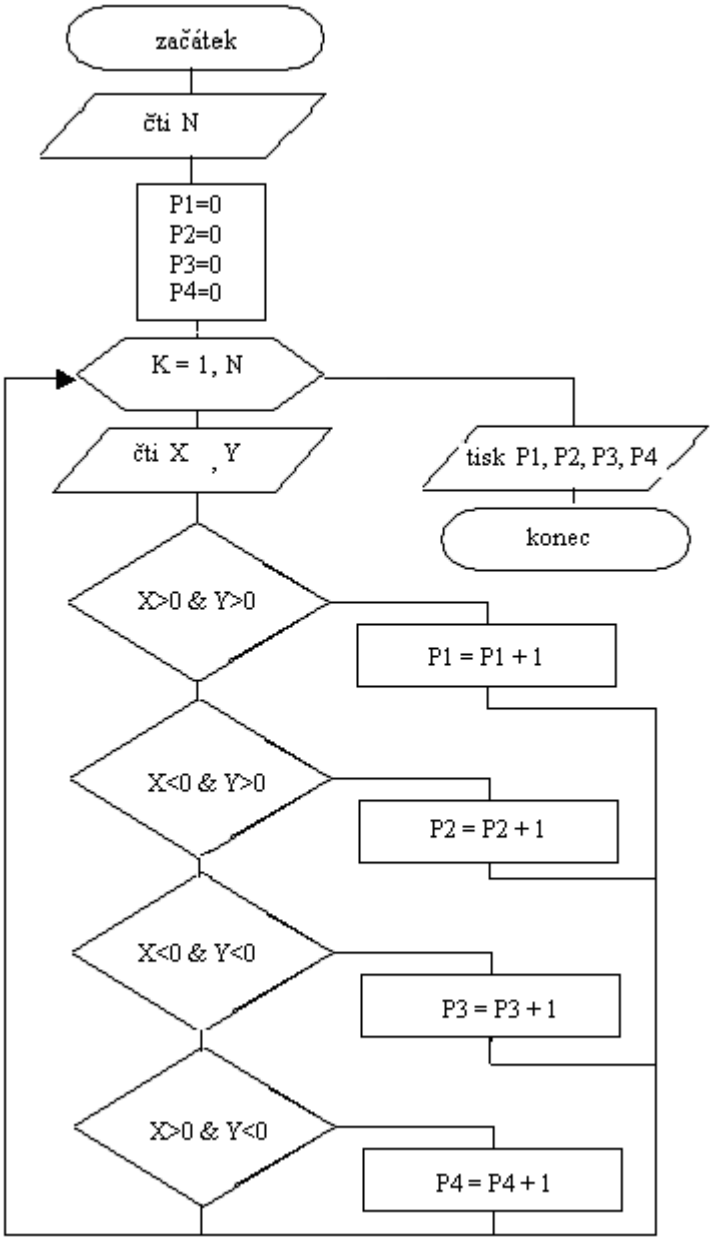
V bloku *A mod 2 = 0* možno použít *(A div 2)*2 = A* nebo *not (odd (A))*

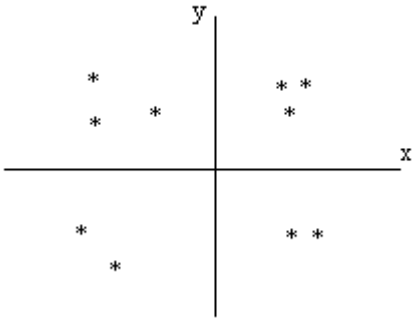


10. Určení počtu bodů

10.1 Určení počtu bodů v kvadrantu

Sestavme VD pro určení počtu bodů ležících v kvadrantu 1, Kvadrantu2, kvadrantu3 a v kvadrantu4. Předpokládejme, že souřadnice *x* a *y* nejsou nulové – body neleží na ose *x* resp. *y*.

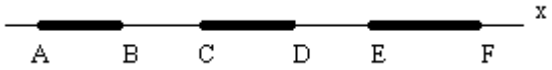


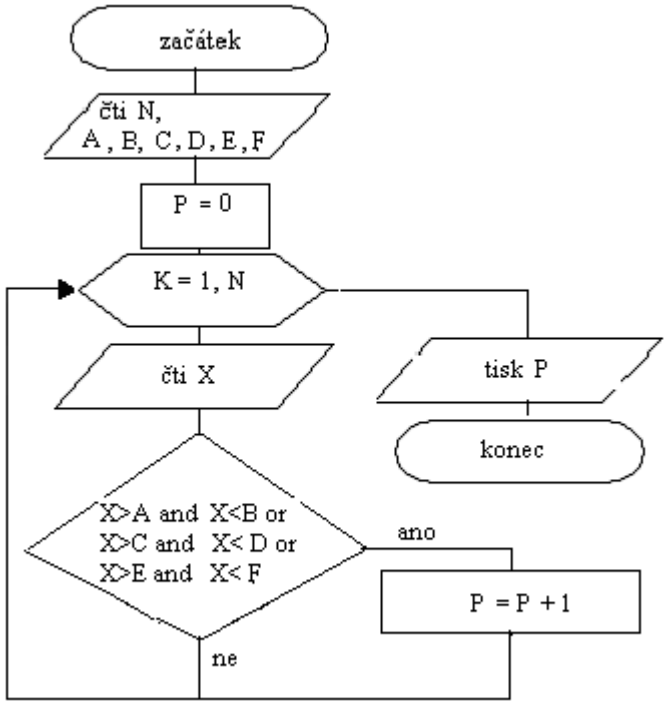



```
var N,K,P1,P2,P3,P4:integer; X,Y:real;
begin
read(input,N);
P1:=0; P2:=0; P3:=0; P4:=0;
for I:=1 to N do
begin
  read(input, X, Y);
  if (X>0) and (Y>0) then P1:=P1+1
  else if (X<0) and (Y>0) then P2:=P2+1
  else  if (X<0) and (Y<0)  then P3:=P3+1
  else P4:=P4+1
          {NE
          if (X>0) and (Y>0) then
          begin P1:=P1+1; goto L1 end;
          if (X<0) and (Y>0) then
          begin P2:=P2+1; goto L1 end;
          if (X<0) and (Y<0) then
          begin P3:=P3+1; goto L1 end;
          if (X>0) and (Y<0) then
          begin P4:=P4+1;
          L1: …….. }
          {nevhodně
if (X>0) and (Y>0) then P1:=P1+1;
if (X<0) and (Y>0) then P2:=P2+1;
if (X<0) and (Y<0) then P3:=P3+1;
if (X>0) and (Y<0) then  P4:=P4+1 }

end;
   writeln(output,
   'počet v 1. kvadrantu:' , P1,
   'počet ve 2. kvadrantu:' , P2,
   'počet v 3. kvadrantu:' , P3,
   'počet ve 4. kvadrantu:' , P4)
        end.
```

10.2 Určení počtu bodů na ose.

Sestavme VD pro určení počtu bodů ležících na ose x v daných intervalech A-B, C-D, E-F, dle následujícího grafu






```
var N,K,P:integer;
    X,A,B,C,D,F: real;

begin
read(N);
read (A,B,C,D,E,F);
P:=0;
for I:=1 to N do
        begin
        read(X);
        if ((X>A) and (X<B))  or  ((X>C) and (X<D))   or  ((X>E) and (X<F))   then P:=P+1
        end;
writeln(`pocet bodu v intervalu:`, P:5)
end.
```

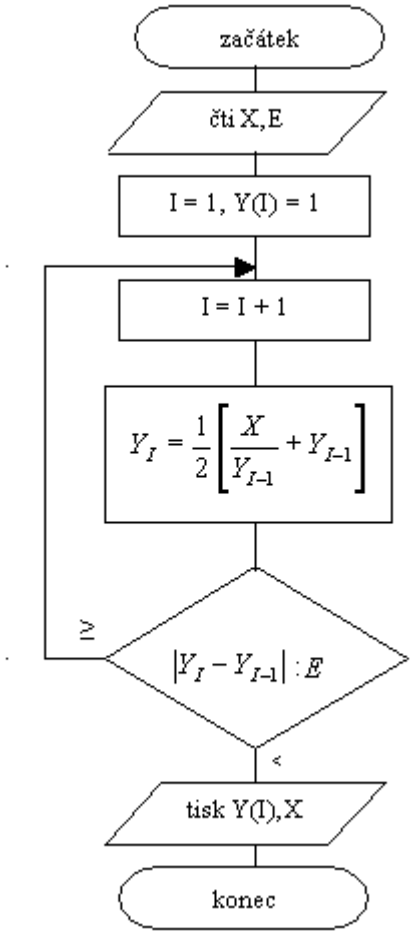
Poznámka:
Pozor na závorkování, chybný je zápis

- if (X>A) and (X<B) or (X>C) and (X<D)   or (X>E) and (X<F)   then P:=P+1
- if X>A and X<B or X>C and X<D   or X>E and X<F   then P:=P+1
- if (X>A and X<B) or (X>C and X<D)   or (X>E and(X<F)   then P:=P+1

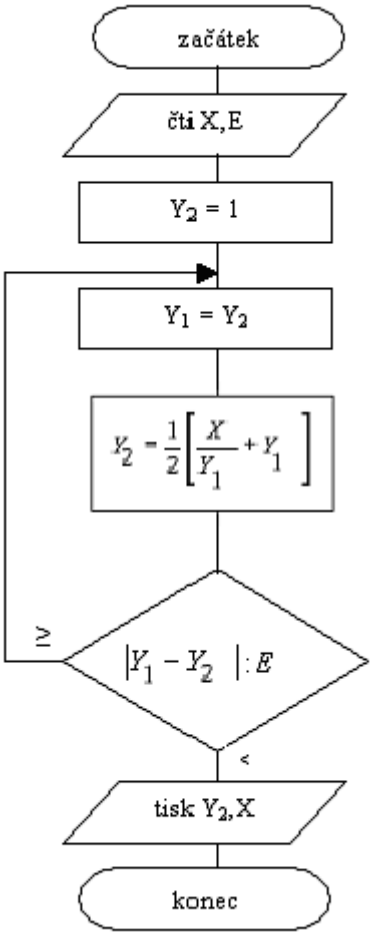
11. Rekurentní metody

U rekurentních metod jde o algoritmy, ve kterých jsou použity rekurentní vzorce, podle kterých lze vypočítat libovolný člen zkoumané posloupnosti na základě předcházejících členů. Při tom první člen, resp. několik počátečních členů musí být zadáno. Jednou z možností využití rekurentních metod je určení druhé odmocniny pomocí Newtonovy metody vedoucí k řešení vztahu $y_0 = 1; \quad y_i = \frac{1}{2}\left[\frac{x}{y_{i-1}} + y_{i-1}\right]$ pro $i = 1, 2, 3, ..$ se zvolenou přesností ε.

V prvním případě není sestavený algoritmus správný. Je sice přesným vyjádřením rekurentních vztahů, ale indexování y_1 , y_2 , y_3 , ,.. může vést k problému deklarace počtu členů posloupnosti y_i. Konvergence $|y_i - y_{i-1}| \le \varepsilon$ může nastat po 10 krocích, ale také po 1000 krocích.

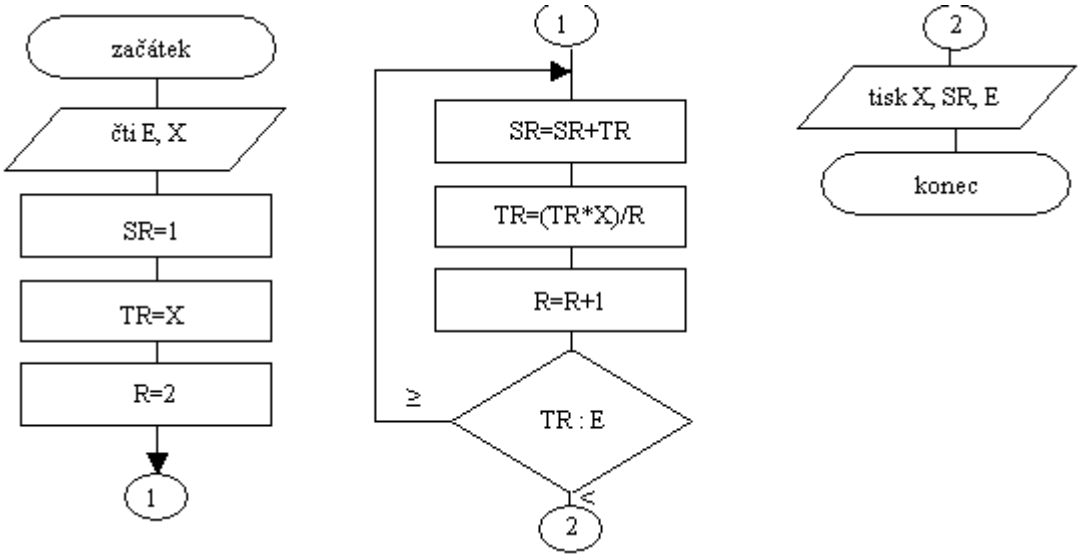


Správný je proto následující VD:



12. Výpočet exponenciální funkce

Výpočet exponenciální funkce pro daný argument X a pro předepsanou přesnost výpočtu ε řešíme řadou $e^x = \sum \frac{x^n}{x!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + ... + \frac{x^i}{i!}$ a výpočet ukončíme pro $\frac{x^i}{i!} \leq \varepsilon$.



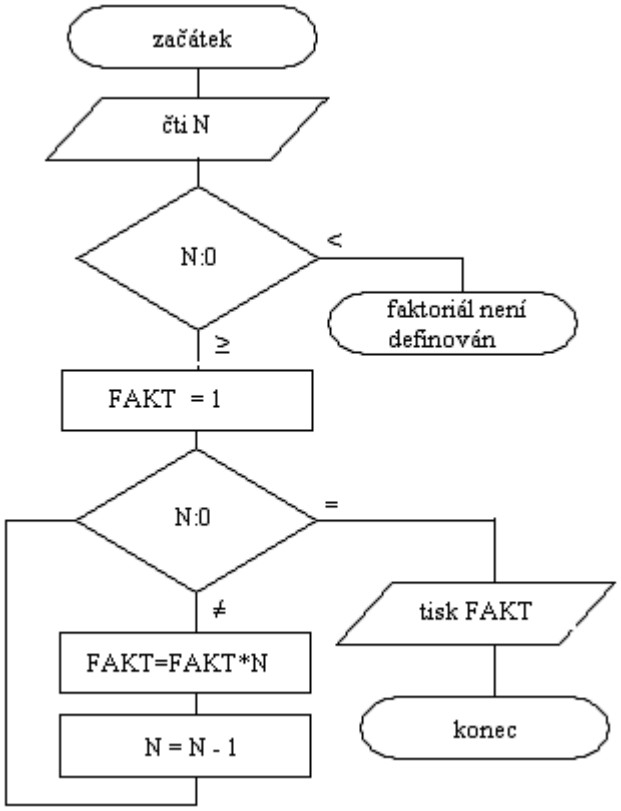

```
var E,TR,X:real;
    SR,R:real;
    label L1;
begin
write('zadej Eps a X');
read(E,X);
SR:=1;
TR:=X;
R:=2;
L1:SR:=SR+TR;
TR:=(TR*X)/R;
R:=R+1;
if TR>=E then goto L1;
writeln('X=',X:7:2,'  exp(X)=',SR:7:2,'  pro eps=',E:5:4);
end.
```

Program může být výhodněji zapsán strukturovaně:

```
var E,TR,X:real;
    SR,R:real;
begin
write('zadej Eps a X');
read(E,X);
SR:=1;
TR:=X;
R:=2;
repeat
SR:=SR+TR;
TR:=(TR*X)/R;
R:=R+1;
until TR<E;
writeln('X=',X:7:2,'  exp(X)=',SR:7:2,'  pro eps=',E:5:4);
end.
```

13. Výpočet faktoriálu čísla

Faktoriál je formálně definován pro čísla celá kladná vztahem $N! = N.(N-1).(N-2). .. .3.2.1.$ Faktoriál nuly je roven 1, faktoriál záporných čísel není definován. Výpočet je prováděn tak, že použijeme proměnnou *FAKT*, jejíž počáteční hodnota je rovna jedné. Uvedenou proměnnou *FAKT* násobíme zadanou hodnotou *N* a výsledek uložíme opět na *FAKT*. Proměnnou *N* snížíme o jedničku a provádíme předchozí operaci. Tuto opakujeme, až hodnota *N* bude rovna nule. Proměnná *FAKT* obsahuje hodnotu faktoriálu původně zadaného čísla *N*.



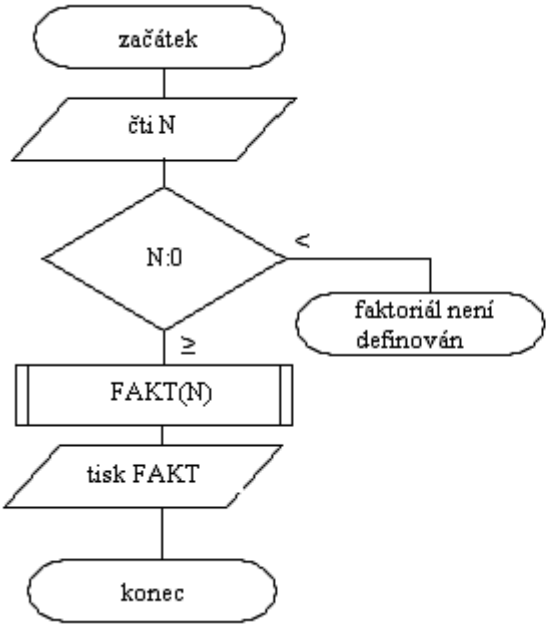

```
var N, N1:integer;
 FAKT:longint;
begin
read(N);
N1:=N; {pomocná N1 pro tisk}
if N< 0 then writeln('faktorial neexistuje')
        else begin
             FAKT:=1;
             while N<>0 do
                   begin
                  FAKT:=FAKT*N;
                   N:=N-1;
                   end;
writeln('faktorial ',N1:5,' je ',FAKT:10);
end.
```

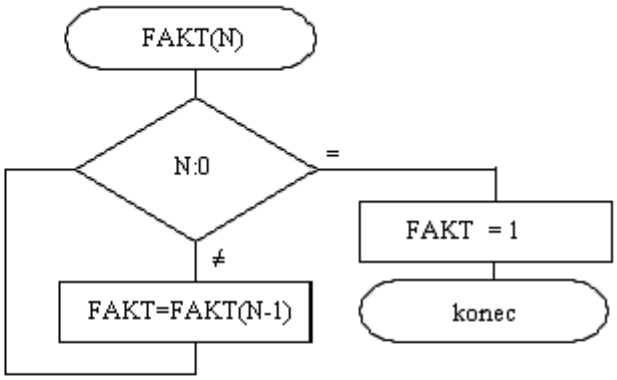
Program lze zapsat rovněž následujícím, méně vhodným způsobem

```
var N:integer;
FAKT:longint;
label L1;
begin read(N); write('faktorial ',N:5);
    if N< 0 then writeln('faktorial neexistuje');
    FAKT:=1;
    L1:if N<>0 then
                 begin FAKT:=FAKT*N;
                 N:=N-1;  goto L1;
                end
           else writeln(' je ',FAKT:10);
end.
```

14. Rekurze

Rekurze se velmi často využívá při syntaktické analýze textů formálních jazyků. Rekurzivní postup lze například použít pro výpočet faktoriálu *N*!. *N*! = pokud *N* > 2 pak (*N* - 1)! * *N* jinak je to *N*. To znamená, že za jistých podmínek se dá faktoriál spočítat pomocí faktoriálu.



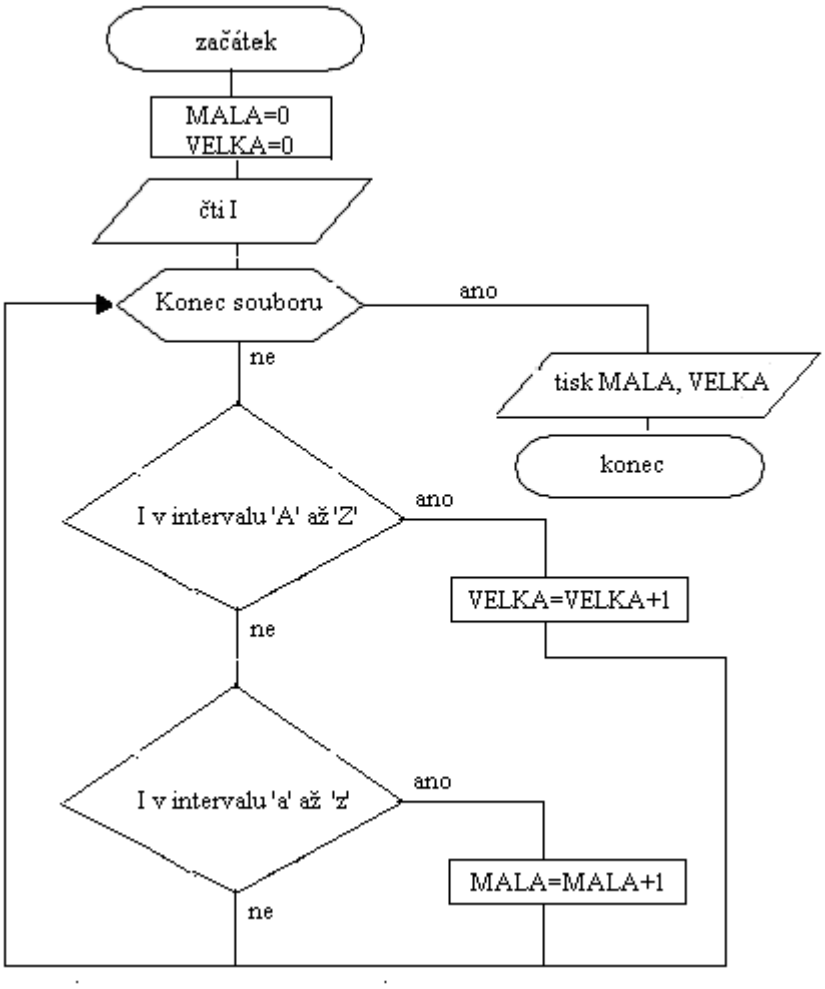



```
var N:integer;
        function FAKT(N:integer):integer;
        begin
        if N=0 then FAKT:=1
       else FAKT:=M*FAKT(N-1);
        end;

begin
write('ZADEJ N: ');
read (N);
if N<0 then writeln('faktorial neexistuje')
        else writeln('faktorial ',N:5,'je ',FAKT(N):5)
end.
```

Poznámka:
Existují i případy, kdy použití rekurze není vhodné. Jako příklad může sloužit Fibonacciho posloupnost. Rekurzivní algoritmus má exponenciální výpočetní postup, oproti tomu běžný algoritmus (počítající členy od začátku) má postup lineární.

15. Určení počtu symbolů.

Určeme počet malých písmen a velkých písmen v datovém souboru na příklad DATA.DAT.




```
Var I:char;
MALA,VELKA:integer;
F:text;

begin
assign(F,'C:\DATA.DAT');
reset(F);
MALA:=0;
VELKA:=0;

while not eof(F) do
  begin
  read(f,I);

      case I of
      'A'..'Z':VELKA:=VELKA+1;
      'a'..'z':MALA:=MALA+1;
      end

 end;

writeln('velka: ',VELKA);
writeln('mala: ',MALA);
readln
end.
```

16. Maticový počet

Maticí nazýváme *m.n* čísel, uspořádaných do obdélníkového schématu o *m* sloupcích a *n* řádcích.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & ..... & a_{1n} \\ a_{21} & a_{22} & ..... & a_{2n} \\ ..... & & & \\ a_{m1} & a_{m2} & ..... & a_{mn} \end{bmatrix}$$

při čemž a_{ij} (i=1, 2,..,n), (j=1, 2, …,m) jsou jednotlivé prvky dané matice. Je-li m=n nazýváme matici čtvercovou. Má-li matice čtvercová prvky a_{ii} (i=1,2,..,n) různá od nuly a prvky a_{ij} ($i \neq j$) rovny nule, pak tuto matici nazýváme diagonální maticí. Jednotkovou maticí nazýváme matici, kde prvky a_{ii} =1, (i=1, 2,..,n).

Pro čtvercovou matici můžeme zavést pojem determinant matice **A**, pro matici 2x2 a 3x3 můžeme determinant řešit velmi jednoduše pomocí Sarusova pravidla.

Pro n = 2 je tedy determinant roven rozdílu součinů prvků na hlavní a vedlejší diagonále:

$$\text{Det}\,\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = a_{11} \,.\, a_{22} - a_{21} \,.\, a_{12}$$

Pro n=3 je výhodné následující zápis:

$$\text{Det}\,\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad \text{Det}\,\mathbf{A} = \left[\begin{array}{ccc|cc} a_{11} & a_{12} & a_{13} & a_{11} & a_{12} \\ a_{21} & a_{22} & a_{23} & a_{21} & a_{22} \\ a_{31} & a_{32} & a_{33} & a_{31} & a_{32} \end{array}\right] =$$

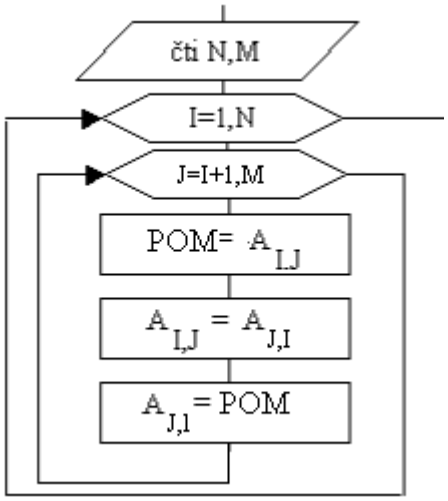
$$= (a_{11} \,.\, a_{22} \,.\, a_{33} + a_{12} \,.\, a_{23} \,.\, a_{31} + a_{13} \,.\, a_{21} \,.\, a_{32}) - (a_{31} \,.\, a_{22} \,.\, a_{13} + a_{32} \,.\, a_{23} \,.\, a_{11} + a_{33} \,.\, a_{21} \,.\, a_{12})$$

Pro n>3 je nutno použít jiného přístupu, například některou vhodnou eliminační metodou.

Transponovaná matice pro matici čtvercovou se získá překlopením prvků matice kolem hlavní diagonály, u obdélníkové matice je nutno zaměnit řádky za sloupce dle následujícího schématu:

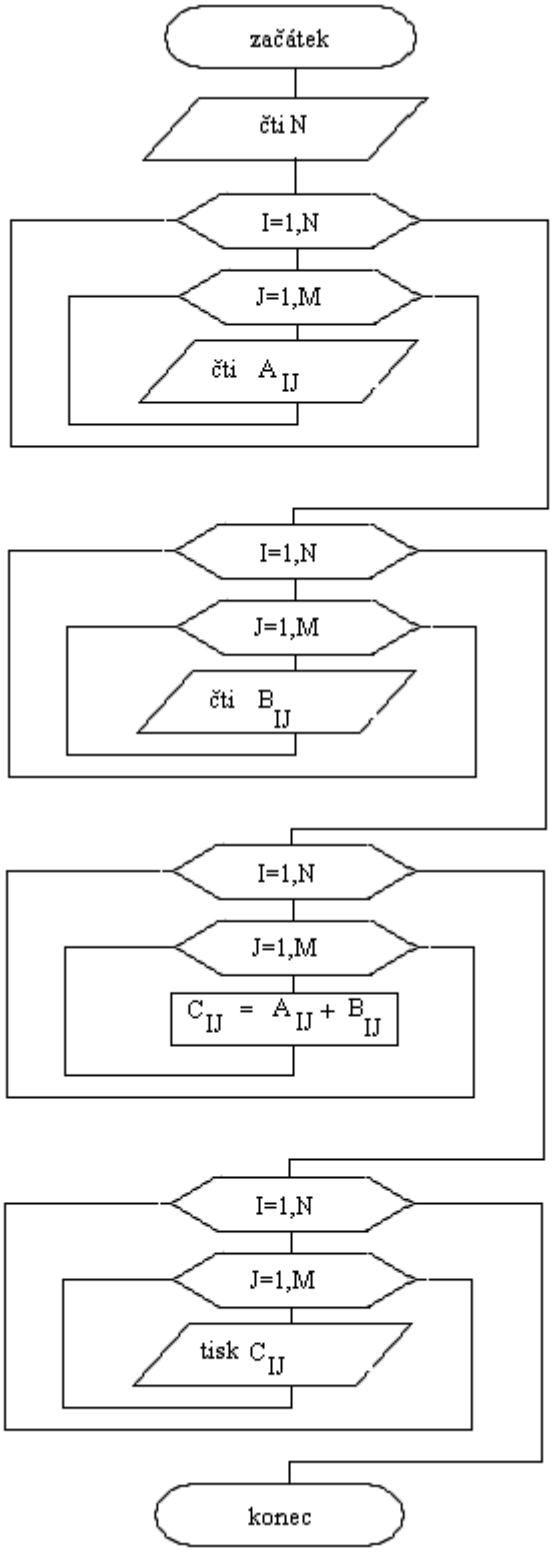
$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix} \quad \mathbf{A}^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} & a_{31} \\ a_{12} & a_{22} & a_{32} & a_{42} \\ a_{13} & a_{23} & a_{33} & a_{43} \end{bmatrix}$$

Pro čtvercovou matici je sestavení VD jednoduché:



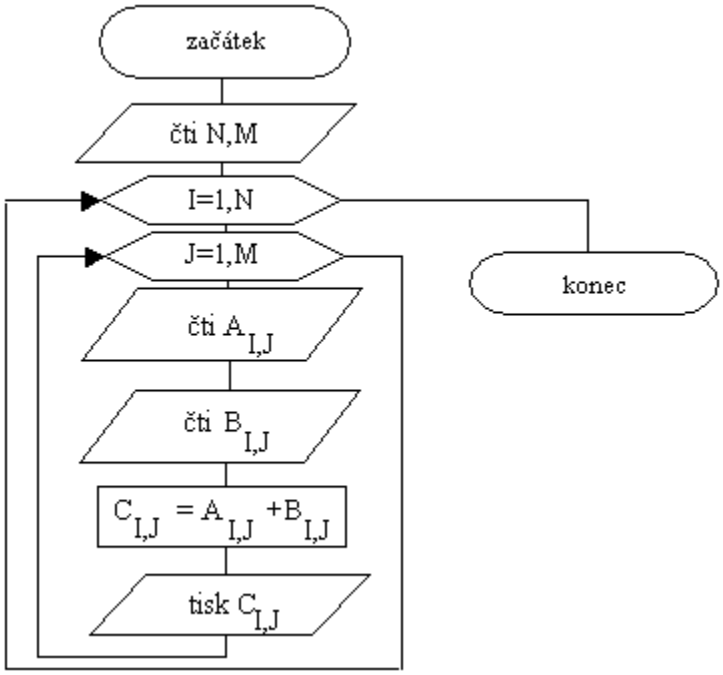
Typickým příkladem je sčítání matice **A** + **B**. Součtem dvou matic stejného typu nazýváme matici **C** téhož typu, jejíž prvky jsou rovny součtu odpovídajících prvků a_{ij} a b_{ij}, tedy $c_{ij} = a_{ij} + b_{ij}$.

$$\mathbf{C} = \mathbf{A} + \mathbf{B} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & ..... & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & ..... & a_{2n} + b_{2n} \\ ..... & & & \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & ..... & a_{mn} + b_{mn} \end{bmatrix}$$




```
var A,B,C: array [1..20,1..20] of real;
    N,M: integer;

begin
read(N,M);

for I:=1 to N do
for J:=1 to M do
read (A[I,J]);

for I:=1 to N do
for J:=1 to M do
read (B[I,J]);

for I:=1 to N do
for J:=1 to M do
C[I,J]= A[I,J]+B[I,J];

for I:=1 to N do
begin
for J:=1 to M do
write (A[I,J]);
writeln;{tímto příkazem zajistíme tisk matice po řádcích pro libovolný pocet sloupců}
end
end.
```

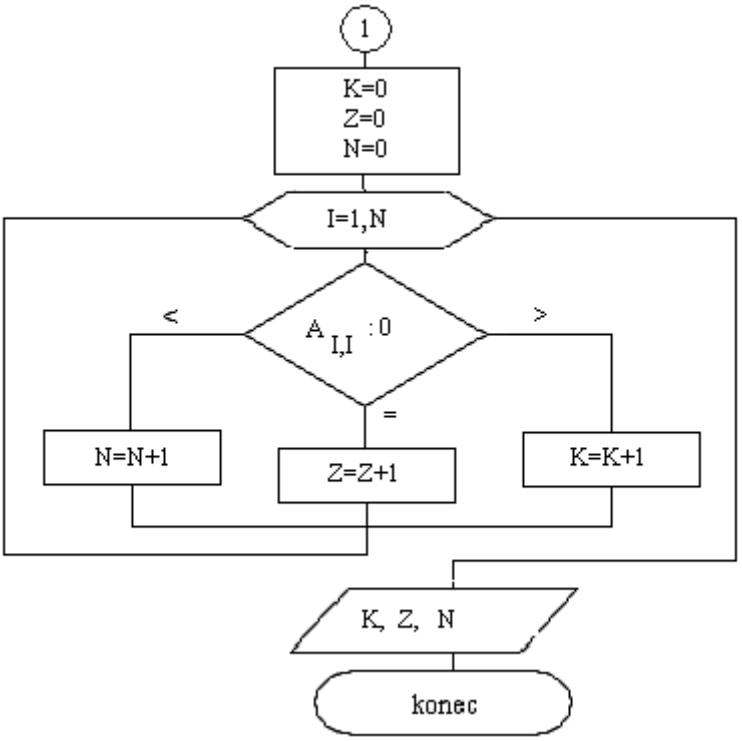
Program lze rovněž zapsat ne právě šťastným způsobem čtení prvků matic. V tomto případě je možno indexy *I,J* u matic **A**,**B**,**C** vypustit:




```
var A,B,C: array [1..20,1..20] of real;
     N,M: integer;
begin
read (N,M);
for I:=1 to N do
  begin
  writeln;
  for J:=1 to M do
      begin
      read (A[I,J];
      read (B[I,J];
      C[I,J]= A[I,J]+B[I,J];
      write (A[I,J]
      end
   end
end.
```

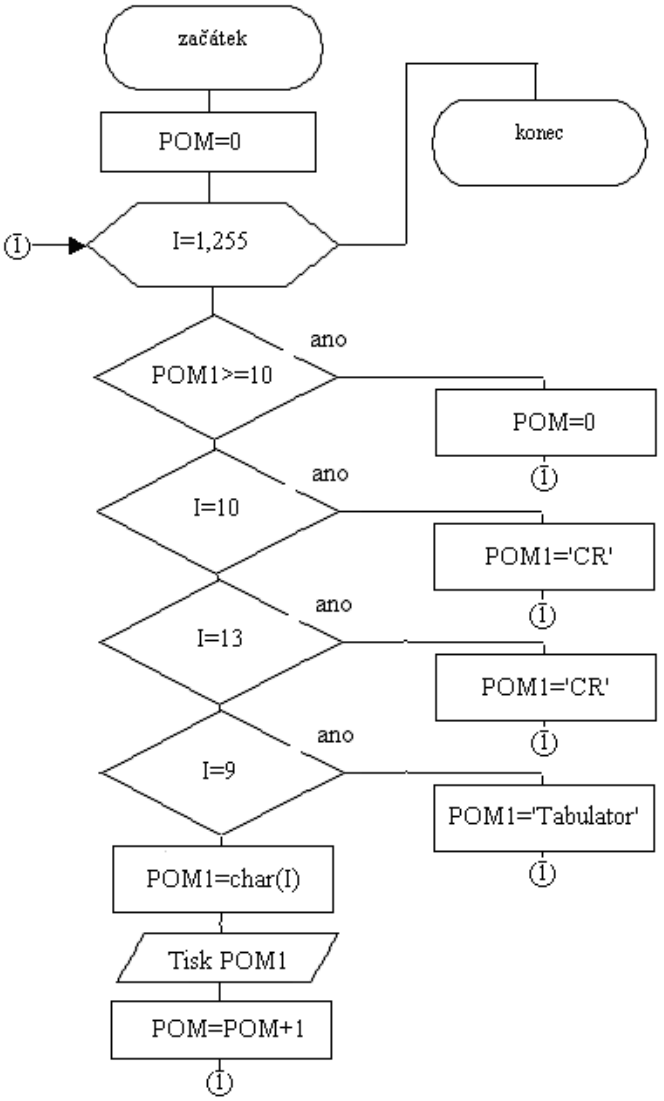
- Sestavme VD resp. program, který určí počet kladných, počet záporných a počet nulových prvků ležících na hlavní diagonále matice **A**. Poznamenejme, že prvky na hlavní diagonále mají stejné indexy, *I=J* a matice musí být čtvercová.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & ..... & a_{1n} \\ a_{21} & a_{22} & ..... & a_{2n} \\ ..... & & & \\ a_{n1} & a_{n2} & ..... & a_{nn} \end{bmatrix}$$



17. Vytvoření ASCII tabulky

Sestavme VD a program pro vytvoření ASCII tabulky (American Standard Cod for Information Interchange). Pro znaky *Nový řádek*, *Posun o řádek* a *Tabulátor* proveďme tisk CR, LF, TB.




```
var I,POM:integer;
POM1: string[2];
F:text;
begin
assign(F,'ascii.dat');
rewrite(F);
writeln(F);POM:=0;
for I:=0 to 255  do
begin;
if POM>=10 then
begin writeln(F);
POM:=0
end;
if I=10 then POM1:= 'lf'
else if I=13 then POM1:= 'cr'
else if i=9 then POM1:= 'tb'
else POM1:=char(I);
write(F,i:2,POM1:2);
POM:=POM+1;
end;
close(F);
end.
```

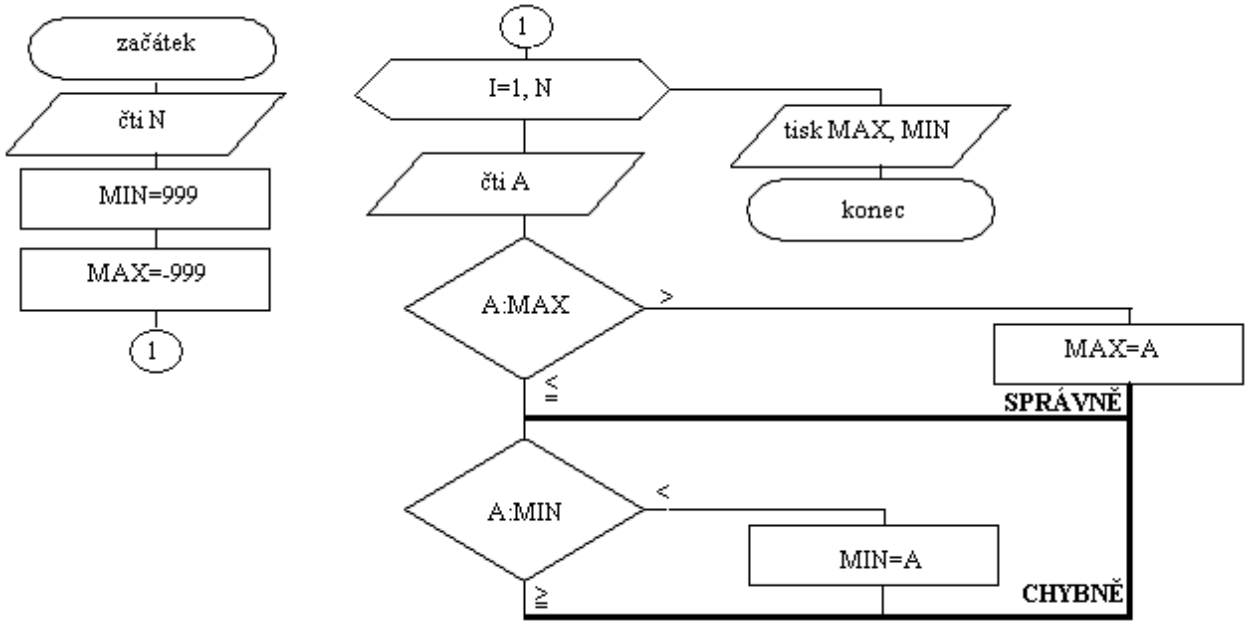
Výsledný soubor ascii.dat:

```
  0       1 ☺     2 ☻     3 ♥     4 ♦     5 ♣     6 ♠     7 •     8 ◘     9 ■
 10 lf  11 ♂  12 ♀  13 cr  14 ♫  15 ☼  16 ►  17 ◄  18 ↕  19 !!
 20 ¶    21 §    22 ▬    23 ↨    24 ↑    25 ↓    26 →    27 ←    28 ∟    29 ↔
 30 ▲    31 ▼    32      33 !    34 "    35 #    36 $    37 %    38 &    39 '
 40 (    41 )    42 *    43 +    44 ,    45 -    46 .    47 /    48 0    49 1
 50 2    51 3    52 4    53 5    54 6    55 7    56 8    57 9    58 :    59 ;
 60 <    61 =    62 >    63 ?    64 @    65 A    66 B    67 C    68 D    69 E
 70 F    71 G    72 H    73 I    74 J    75 K    76 L    77 M    78 N    79 O
 80 P    81 Q    82 R    83 S    84 T    85 U    86 V    87 W    88 X    89 Y
 90 Z    91 [    92 \    93 ]    94 ^    95 _    96 `    97 a    98 b    99 c
100 d   101 e   102 f   103 g   104 h   105 i   106 j   107 k   108 l   109 m
110 n   111 o   112 p   113 q   114 r   115 s   116 t   117 u   118 v   119 w
120 x   121 y   122 z   123 {   124 |   125 }   126 ~   127 ⌂   128 Ç   129 ü
130 é   131 â   132 ä   133 ů   134 ć   135 ç   136 ł   137 ë   138 Ő   139 ő
140 î   141 Ź   142 Ä   143 Ć   144 É   145 Ĺ   146 Í   147 ô   148 ö   149 Ľ
```

```
150 Ľ  151 Ś  152 ś  153 Ö  154 Ü  155 Ť  156 ť  157 Ł  158 ×  159 č
160 á  161 í  162 ó  163 ú  164 Ą  165 ą  166 Ž  167 ž  168 Ę  169 ę
170 ¬  171 ź  172 Č  173 ş  174 «  175 »  176 ░  177 ▒  178 ▓  179 │
180 ┤  181 Á  182 Â  183 Ě  184 Ş  185 ╣  186 ║  187 ╗  188 ╝  189 Ż
190 ż  191 ┐  192 └  193 ┴  194 ┬  195 ├  196 ─  197 ┼  198 Ă  199 ă
200 ╚  201 ╔  202 ╩  203 ╦  204 ╠  205 ═  206 ╬  207 ¤  208 đ  209 Đ
210 Ď  211 Ë  212 ď  213 Ň  214 Í  215 Î  216 ě  217 ┘  218 ┌  219 █
220 ▄  221 Ţ  222 Ů  223 ▀  224 Ó  225 ß  226 Ô  227 Ń  228 ń  229 ň
230 Š  231 š  232 Ŕ  233 Ú  234 ŕ  235 Ű  236 ý  237 Ý  238 ţ  239 ´
240 -  241 ˝  242 ˛  243 ˇ  244 ˘  245 §  246 ÷  247 ¸  248 °  249 ¨
250 ˙  251 ű  252 Ř  253 ř  254 ■  255
```

18. Nalezení chyby v programu

V některých případech je obtížné nalést chybu v sestaveném algoritmu. Uveďme celkem jednoduchý příklad pro nalezení maximálního a minimálního čísla (*MIX, MIN*) z dané posloupnosti čísel A_I, I=1 až N.

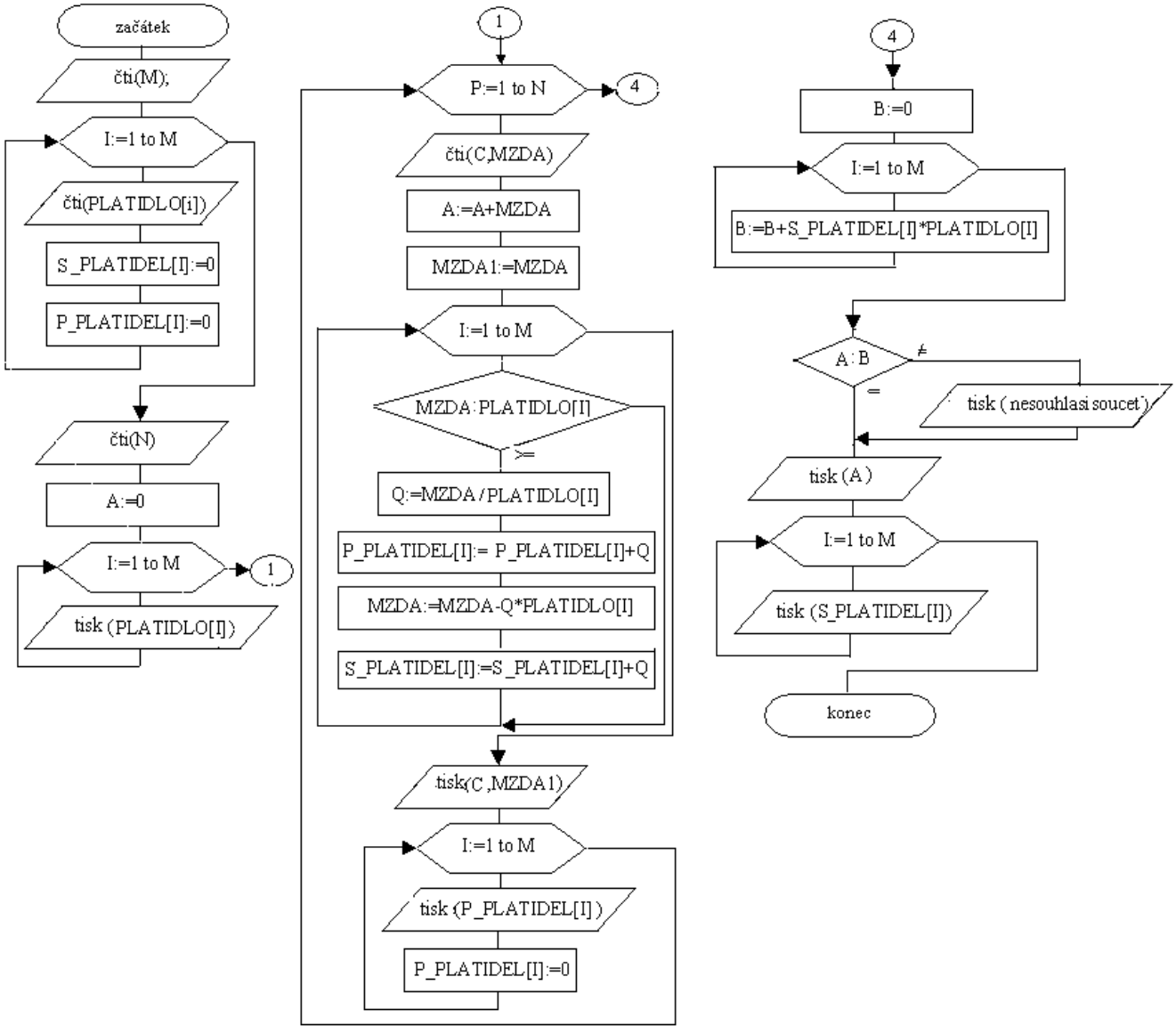


Program se zdánlivě jeví jako odladěný, chybu lze nalést pouze pro data, kde první prvek v řadě A_1 je minimální.

```
var N,I: integer;
    A,MIN,MAX: real;
begin
read (N);
MIN:=999;
MAX:=-999;
for I:= 1 to N do
begin
read (A);
if A>MAX then MAX:=A;   {chybně if A>MAX then MAX:=A
if A<MIN   then MIN:= A;                          else MIN:= A }
end;
writeln('maximalni prvek: ',MAX:10:4,'minimalni prvek:',MIN:10:4);
end.
```

19. Výčetka platidel

Podle vypočtených částek čisté mzdy zjistěme pro jednotlivé pracovníky minimální počty druhů jednotlivých platidel. Provedeme čtení počtu současných platidel *M*, jednotlivých platidel $PLATIDLO_I$ (například 5000, 2000, 1000, 500, 100, 50, 20, 10, 5, 1, 0.5). Dále přečteme počet zaměstnanců *N*, číslo zaměstnance *CISLO* a čistou mzdu *MZDA*. Minimální počty jednotlivých platidel označme $P\_PLATIDEL_I$. Počty jednotlivých platidel určíme postupným celočíselným dělením $MZDA/PLATIDLO_I$. Na *S_PLATIDEL* uložíme celkové součty jednotlivých platidel, na *B* uložíme kontrolní součet.



Na následujícím programu je zřejmý rozdíl v zápisu VD a vlastního zdrojového programu, například nutná deklarace všech proměnných, zápis textu pro orientaci při čtení dat, atd.

PROGRAM:

```
 var I,P,A,M,N,C,V,Q,V1,B:integer;
    L,J,K:array [1..25] of integer;
begin
   writeln('zadej pocet platidel  ');
read(M);
writeln('zadej',M,'x platidla od max. k min. ');
  for I:=1 to M do
  begin
  read(K[I]);
  L[I]:=0;
  J[I]:=0;
  end;
writeln('zadej pocet pracovniku  ');
read(N);
A:=0;
writeln('c.zam. plat pocet platidel');
  for I:=1 to M do
  write('         ',K[I]:5);
  writeln('zadej cislo a plat  ');
    for P:=1 to N do
    begin
    read(C,V);
    A:=A+V;
    V1:=V;
       for I:=1 to M do
       begin
       if V>=K[I] then begin
                  Q:=V div K[I];
                  J[I]:=J[I]+Q;
                  V:=V-Q*K[I];
                  L[I]:=L[I]+Q;
                  end;
       end;
     write(C:5,V1:5);
       for I:=1 to M do
       begin
       write(J[I]:5);
       J[I]:=0;
       end;
     end;
B:=0;
  for I:=1 to M do
  B:=B+L[I]*K[I];
if A<>B then
writeln('nesouhlasi kontrolni soucet');
writeln('celkove soucty  ');
write(A:5);
for I:=1 to M do
write(L[I]:5);
       end.
```

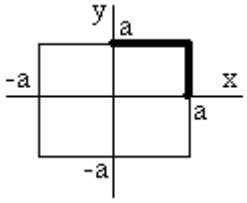
20. Literatura

1. OLEHLA,J., OLEHLA,M.: Basic u mikropočítačů. NADAS PRAHA 1988
2. BINZINGER, T.: Naučte se programovat Delphi. Praha, Grada 1999
3. JINOCH, J.,MÜLLER,K.,VOGEL,J.: Programování v jazyku PASCAL. Praha, SNTL 1986.
4. MOLNÁR, L. : Programovanie v jazyku PASCAL. Bratislava, ALFA 1987.
5. OLEHLA, M.: Počítače a programování . Turbo Pascal. Liberec, VŠST 1997, TU 2010.
6. OLEHLA, M.:Vývojové diagramy. http:// www.kky.vslib.cz/kky/textbooks, Liberec 2000
7. OLEHLA, M.: Počítače a programování, vývojové diagramy. Liberec, TU 2010.
8. TÖPFLEROVÁ D., TÖPFLER P. : Sbírka úloh z programování, GRADA Praha 1992
9. TAUFER, I., HRUBINA,K., TAUFER,J.: Algoritmy a algoritmizace – vývojové diagramy. Pardubice - Prešov - Hradec Králové 2001
10. http://lide.uhk.cz/home/pdf/ucitel/pujehlv1/www/vyuka/programovani/algoritmy/
11. http://www.mujweb.cz/pocitace/myprogram/Index_soubory/vyvoj.htm
12. Information processing - Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts. ISO 5807, 1985
13. http://pascal.webz.cz/index.php?part=down
14. http://cs.wikipedia.org/wiki/Rekurze

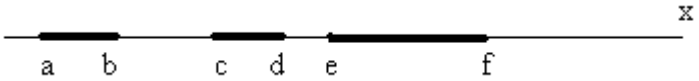
21. Příloha – Zadání příkladů

Jednoduché příklady

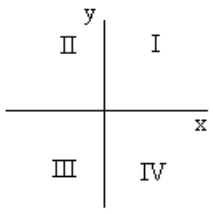
1. Sestavme VD resp. program pro určení přepony pravoúhlého trojúhelnika pomocí Pythagorovy věty, znám je rozměr odvěsny a,b ($c^2=a^2+b^2$)
2. Pro různé úhly γ_i, i=1,N máme vypočítat stranu obecného trojúhelníka, známe-li dvě strany a úhel jimi sevřený γ. K řešení použijeme kosinové věty ($c^2=a^2+b^2$ -2.a.b.cos γ)
3. Sestavme VD resp. program pro určení BMI pro zadanou váhu [kg] a výšku [m] a pro BMI <18,5 – tisk podváha, BMI >= 18,5 a BMI < 25 v normě, BMI > 25 a BMI < 30 nadváha, BMI > 30 obezita. BMI=hmotnost / (výška x výška)
4. Sestavme VD resp. program pro určení počtu bodů ležících uvnitř, na a vně kružnice o poloměru R. Počet bodů je N a jsou zadány souřadnicemi (x_i, y_i), i=1, N. Kružnice má střed o souřadnicích (0,0)
5. Sestavme VD a program pro určení povrchu a objemu koule.(S=4.¶.r^2, V=4/3. ¶.r^3)
6. Sestavme VD a program pro určení povrchu a objemu válce. *(*O=¶.r^2.v, P=2.¶.r.(r+v))
7. Sestavme VD a program pro určení obvodu a obsahu kruhu. (O=2.¶.r, P=¶. r^2)
8. Určeme počet bodů ležících uvnitř, na a vně v oblasti zadané následujícím grafem.



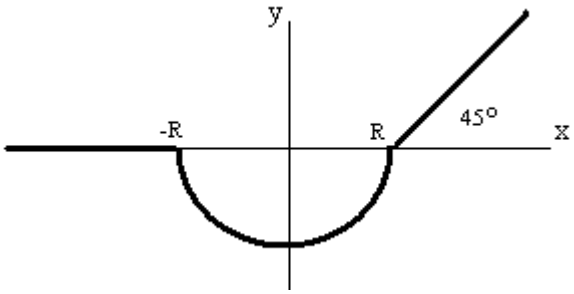
   Počet bodů je N a jsou zadány souřadnicemi (x_i, y_i), i=1, N
9. Sestavme VD resp. program pro vložení prvku a_k do řady a_i, i=1,N a vytvořme novou řadu a_i, i=1,N+1
10. Sestavme VD resp. program pro vyjmutí prvku a_k z řady a_i, i=1,N a vytvořme novou řadu a_i, i=1,N-1
11. Sestavme VD resp. program pro náhradu prvku a_k prvkem b v řadě a_i, i=1,N a vytvořme novou řadu a_i, i=1,N
12. Sestavme VD a program pro vytvoření řady kladných, řady záporných a nulových prvků z řady a_i, i=1,N
13. Sestavme VD a program pro vytvoření převodní tabulky z řady čísel zadaných ve stupních C na F. (F=C.(9/5)+35)
14. Určeme počet hodnot ležících uvnitř zadaných intervalů daných následujícím grafem



15. Určeme počet bodů ležících ve zvolených kvadrantech daných následujícím grafem



16. Určeme funkční hodnotu $f(x)$ pro zadané x_i , i=1,N dle následujícího grafu



Je dána řada celočíselných (reálných) hodnot ukončená koncovým znakem -999, který do řady nepatří.

17. Určeme dvě největší čísla z dané řady
18. Určeme počet kladných a počet nulových čísel z dané řady
19. Určeme aritmetický průměr z lichých čísel z dané řady
20. Je-li první číslo kladné, vytiskněte všechna kladná čísla, je-li záporné, vytiskněte všechna záporná čísla
21. Určeme pořadové číslo nejmenšího, největšího čísla z dané řady
22. Určeme, zda posloupnost čísel má větší počet kladných nebo záporných čísel
23. Určeme pořadí závodníků, jestliže výkon závodníka je hodnocen počtem bodů. V souboru dat je uvedeno číslo závodníka, jméno a příjmení a počet bodů.
24. Vypočítejme posloupnost Fibonacciových čísel. Tato posloupnost je dána součtem dvou následujících čísel. Prvé dva členy volíme obvykle rovny jedné, pak je řada 1 1 2 3 5 8 13 21 34 …

Je dána řada celočíselných hodnot, u řady je znám počet prvků řady *N*.

25. Určeme aritmetický průměr a rozptyl dané řady x_i

$$\bar{x} = \sum_{i=1}^{N} x_i / N$$

$$s^2 = \sum_{i=1}^{N} (x_i - \bar{x})^2 / (N-1)$$

26. Určeme dvě největší čísla z dané řady
27. Určeme počet kladných a počet nulových čísel z dané řady
28. Určeme aritmetický průměr z lichých čísel z dané řady
29. Je-li první číslo kladné, vytiskněte všechna kladná čísla, je-li záporné, vytiskněte všechna záporná čísla
30. Určeme pořadové číslo nejmenšího, největšího čísla z dané řady
31. Určeme, zda posloupnost čísel má větší počet kladných nebo záporných čísel
32. Sestavme VD resp. program, který sečte prvky posloupnost a_i, i=1,N s prvky posloupnosti b_i, i=1,N
33. Sestavme VD resp. program, který vytiskne původní posloupnost a posloupnost, která vznikne z původního vydělením každého prvku největším prvkem původní posloupnosti
34. Sestavme VD resp. program, který vytiskne původní posloupnost a posloupnost, která vznikne z původního vydělením každého prvku nejmenším prvkem původní posloupnosti
35. Sestavme VD resp. program pro součet prvků řady a_i, i=1,N

Aritmetické funkce

36. Sestavme VD resp. program, který vypočte $S = \sum_{i=1}^{N} (\sin x_i + 2.\cos x_i)$

37. Sestavme VD resp. program, který vypočte $S = \prod_{i=1}^{N} (\sin x_i + 2.\cos x_i)$

38. Sestavme VD resp. program, který vypočte $sum = \left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + ...\frac{1}{n}\right)$ rozvojem řadou s přesností ε

39. Sestavme VD resp. program, který vypočte $sum = \left(\frac{1}{1} + \frac{1}{2} - \frac{1}{3} + \frac{1}{4} - .....\frac{1}{n}\right)$ rozvojem řadou s přesností ε

40. Sestavme VD resp. program, který vypočte $sum = \left(\frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + ...\frac{1}{n!}\right)$ rozvojem řadou s přesností ε

41. Sestavme VD resp. program, který převede kartézské souřadnice na polární, je-li pól v počátku kartézské soustavy, pak $\begin{matrix} x = r\cos\varphi \\ y = r\sin\varphi \end{matrix}$. Například bod *M*, jehož polární souřadnice jsou $r = \sqrt{5};\ \varphi = 2{,}68$, má kartézské souřadnice (x = –2, y = 1)

42. Sestavme VD resp. program, který vypočte π rozvojem řadou s přesností ε

$$\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + ...\right)$$

$$\pi = 6\left(\frac{1}{\sqrt{3}} - \frac{1}{3\sqrt{3^3}} + \frac{1}{5\sqrt{3^5}} - \frac{1}{7\sqrt{3^7}} + ...\right)$$

43. Sestavme VD resp. program, který vypočte *arctg* (*x*) rozvojem řadou s přesností ε

$$\text{arctg}(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + ...$$

44. Sestavme VD resp. program, který vypočte *sin* (*x*) rozvojem řadou

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + ...$$

45. Sestavme VD resp. program, který vypočte *cos* (*x*) rozvojem řadou

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + ...$$

46. Sestavme VD resp. program, který vypočte e^x rozvojem řadou

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + ..$$

47. Proveďte výpočet třetí odmocniny čísla C iterační metodou pro zvolenou přesnost ε. Pro výpočet použijeme Newtonovu formuli

$$x_0 = C, \quad x_{i+1} = \left(\frac{C}{x_i^2} + 2x_i\right)\frac{1}{3}$$

Práce s dvourozměrným polem (matice, tabulka)

48. Určeme počet nulových a nenulových prvků matice **A** o rozměru N . M

$$\begin{bmatrix} a_{11} & a_{12}..... & a_{1N} \\ a_{21} & a_{22}..... & a_{2N} \\ ..... & & \\ a_{M1} & a_{M2}..... & a_{MN} \end{bmatrix}$$

49. Určeme determinant matice 2x2 a 3x3 pomocí Sarusova pravidla

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33}$$

50. Sestavme VD resp. program pro součet dvou matic **C**=**A**+**B**

$$\begin{bmatrix} c_{11} & c_{12}..... & c_{1N} \\ c_{21} & c_{22}..... & c_{2N} \\ ..... & & \\ c_{M1} & c_{M2}..... & c_{MN} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12}..... & a_{1N} \\ a_{21} & a_{22}..... & a_{2N} \\ ..... & & \\ a_{M1} & a_{M2}..... & a_{MN} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12}..... & b_{1N} \\ b_{21} & b_{22}..... & b_{2N} \\ ..... & & \\ b_{M1} & b_{M2}..... & b_{MN} \end{bmatrix}$$

51. Sestavme VD resp. program pro násobení matice konstantou **B**=**A**xK

$$\begin{bmatrix} b_{11} & b_{12}..... & b_{1N} \\ b_{21} & b_{22}..... & b_{2N} \\ ..... & & \\ b_{M1} & b_{M2}..... & b_{MN} \end{bmatrix} = \begin{bmatrix} a_{11}.K & a_{12}.K..... & a_{1N}.K \\ a_{21}.K & a_{22}.K.... & a_{2N}.K \\ ..... & & \\ a_{M1}.K & a_{M2}.K.... & a_{MN}.K \end{bmatrix}$$

52. Sestavme VD resp. program pro přičtení prvků matice ke konstantě K, tedy

$$\begin{bmatrix} a_{11} & a_{12}..... & a_{1N} \\ a_{21} & a_{22}..... & a_{2N} \\ ..... & & \\ a_{M1} & a_{M2}..... & a_{MN} \end{bmatrix} = \begin{bmatrix} a_{11}+K & a_{12}+K..... & a_{1N}+K \\ a_{21}+K & a_{22}+K.... & a_{2N}+K \\ ..... & & \\ a_{M1}+K & a_{M2}+K.... & a_{MN}+K \end{bmatrix}$$

53. Určeme dvě největší čísla z daného pole N . M
54. Určeme počet kladných a počet nulových čísel z daného pole N . M
55. Určeme aritmetický průměr z lichých čísel z daného pole N . M
56. Je-li v poli N . M první číslo kladné, vytiskněte všechna kladná čísla, je-li záporné, vytiskněte všechna záporná čísla
57. Určeme pořadové číslo nejmenšího, největšího čísla z daného pole N . M
58. Určeme, zda posloupnost čísel má větší počet kladných nebo záporných čísel
59. Určeme aritmetický průměr a rozptyl daného pole – matice o rozměru N . M
60. Určeme maximální a minimální prvek a jeho pozici v poli N . M
61. Určeme součet maximálních prvků z každého řádku (sloupce) matice N . M
62. Určeme počet kladných a počet záporných čísel z daného pole N . M
63. Určeme počet kladných čísel z daného pole N . M větších, než zvolené kladné číslo
64. Určeme indexy prvků matice **A**, které mají maximální a minimální hodnotu
65. Sestavme VD resp. program, který vytiskne původní posloupnost a posloupnost, která vznikne z původního vydělením každého prvku největším prvkem původní posloupnosti

66. Sestavme VD resp. program, který vytiskne původní posloupnost a posloupnost, která vznikne z původního vydělením každého prvku nejmenším prvkem původní posloupnosti
67. Sestavme VD resp. program, který vytiskne původní posloupnost a posloupnost, která se skládá z prvků větších nežli střední hodnota původní posloupnosti
68. Určeme maximální (minimální) prvek na hlavní diagonále čtvercové matice **A**
69. Určeme součet prvků na hlavní diagonále čtvercové matice **A**
70. Určeme indexy prvků matice **A**, které mají nulovou hodnotu
71. U čtvercové matice **A** vyměňme prvky řádků za prvky sloupců – transpozice matice
72. U obdélníkové matice **A** vyměňme prvky řádků za prvky sloupců – transpozice matice
73. U obdélníkové matice **A** vyměňme zvolený řádek *L* za jiný řádek *K*
74. U obdélníkové matice **A** vyměňme řádek s maximální hodnotou prvku na řádku za první řádek matice
75. U čtvercové matice **A** vyměňme řádek s minimální hodnotou prvku na řádku za poslední řádek matice
76. U čtvercové matice **A** vyměňme řádek s minimální hodnotou prvku na řádku za první sloupec matice
77. U obdélníkové matice **A** vyměňme sloupec s maximální hodnotou prvku ve sloupci za první sloupec matice
78. Sestavme VD resp. program, který určí počet kladných, počet záporných a počet nulových prvků ležících na hlavní diagonále matice **A**.
79. U obdélníkové matice **A** vyměňte zvolený sloupec *L* za jiný sloupec *K*
80. Sestavme VD resp. program, který vytvoří jednotkovou matici (matice s jedničkami na hlavní diagonále a nulami nad a pod hlavní diagonálou)

$$\begin{bmatrix} 1 & 0..... & 0 \\ 0 & 1..... & 0 \\ ..... & & \\ 0 & 0..... & ...1 \end{bmatrix}$$

Práce s textem

81. Sestavme VD resp. program, který určí počet číslic a písmen v daném souboru
82. Určeme počet malých a počet velkých písmen v daném souboru
83. Sestavme VD resp. program, který určí počet zvoleného znaku v daném souboru
84. Sestavme VD resp. program, který vypíše všechny slova začínající stejným zvoleným znakem v daném souboru
85. Určeme počet slov BEGIN v datovém souboru

Vyhledávání a třídění

Třídění patří k velmi často používaným způsobem zpracování dat. Provádí se podle zvolených položek – klíčů. Obdobně metoda hledání je častou operací, využívanou například pro kontrolu údajů v tabulce, převádění údajů do jiného kódu, atd. Vzhledem k tomu, že nemusí existovat hledaná věta, musí být zabezpečen alternativní způsob zpracování.

86. Vyhledejme zadanou hodnotu prvku v řadě a_i, $i = 1,N$ pro setříděný soubor pro větu katalogové číslo (klíč), cena. Hledání věty se urychlí tím, že určíme nejdříve blok, ve kterém se hledaná věta nachází a jakmile určíme blok, ve kterém se hledaná věta nachází, probíráme již větu po větě.
    Příklad:

60.10>	10	3,1	
	30	2,1	blok 1
	41	9,1	

60:50> 50 2,8
60:51> 51 8,8 blok 2
60:60= 60 6,8

……..

87. Vyhledejme zadanou hodnotu prvku v řadě a_i, $i = 1,N$ pro nesetříděný soubor pro větu katalogové číslo (klíč), cena. Větu vyhledáme porovnáním zadaného klíče věty s hodnotou klíče věty souboru. Pokud se prohledá soubor až do konce, aniž se nalezne hodnota hledaného klíče, indikujeme chybu.
Příklad:

Katalogové číslo	cena	
146	21,7	146:251 ≠
151	18,4	151:251 ≠
231	31,5	151:251 ≠
251	77,0	251:251 = nalezená věta

……..

88. Setřiďme podle velikosti datový soubor a_i, $i = 1,N$, setříděný soubor uložte do nového souboru. Třídění proveďme vyhledáním minimálního prvku v řadě a_i, a tento prvek uložíme do nové řady b_i (na místo maximálního prvku uložíme speciální značku, tím nebude tento prvek dále platný).

Příklad:

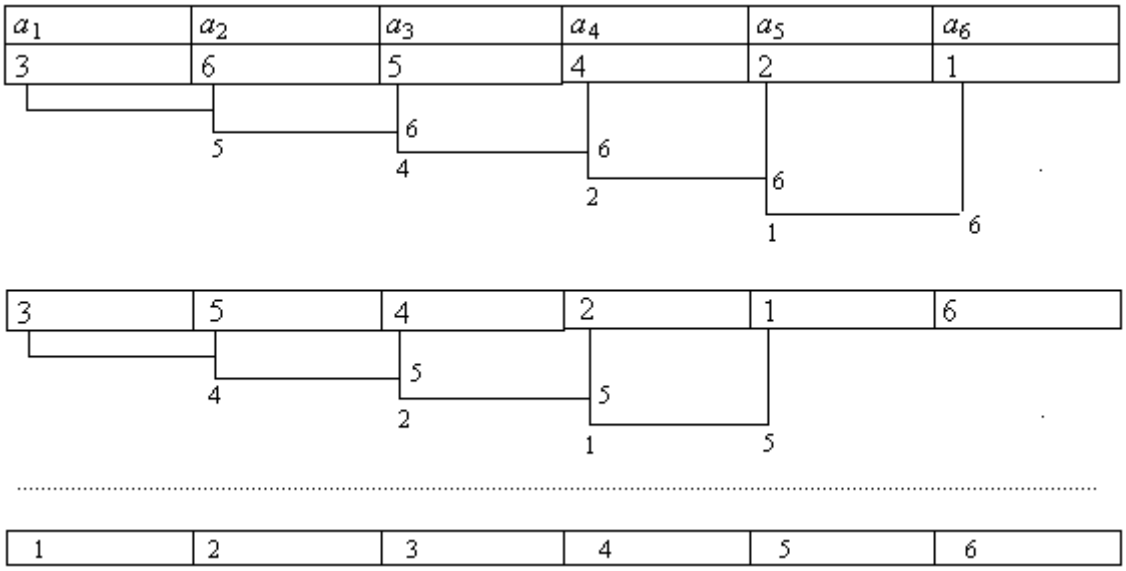
3	6	5	4	2	1
a_1	a_2	a_3	a_4	a_5	a_6

$b_1 = a_1$
$b_1 : a_{1\div6}$
$b_1 = a_6$
a_6= značka, například 9999
$b_2 = a_1$
$b_2 : a_{1\div6}$
$b_2 = a_5$
a_5= značka, například 9999
……

a_1	a_2	a_3	a_4	a_5	a_6
9999	9999	9999	9999	9999	9999

b_1	b_2	b_3	b_4	b_5	b_6
1	2	3	4	5	6

89. Sestavme VD resp. program pro setřídění řady čísel metodou transpozice sousedních prvků – metoda bublání. Třídění probíhá tak, že porovnáme vždy sousední prvky, tj a_i s a_{i+1}, i=1,2,..,N-1, k=1,2..,N-1. Je-li prvek $a_i > a_{i+1}$,provedeme vzájemné přemístění obsahu těchto prvků. Toto přerovnání provádíme tak dlouho, dokud nezjistíme, že při vzájemném porovnání již k přemístění obsahu v celé řadě nedošlo.
Příklad:

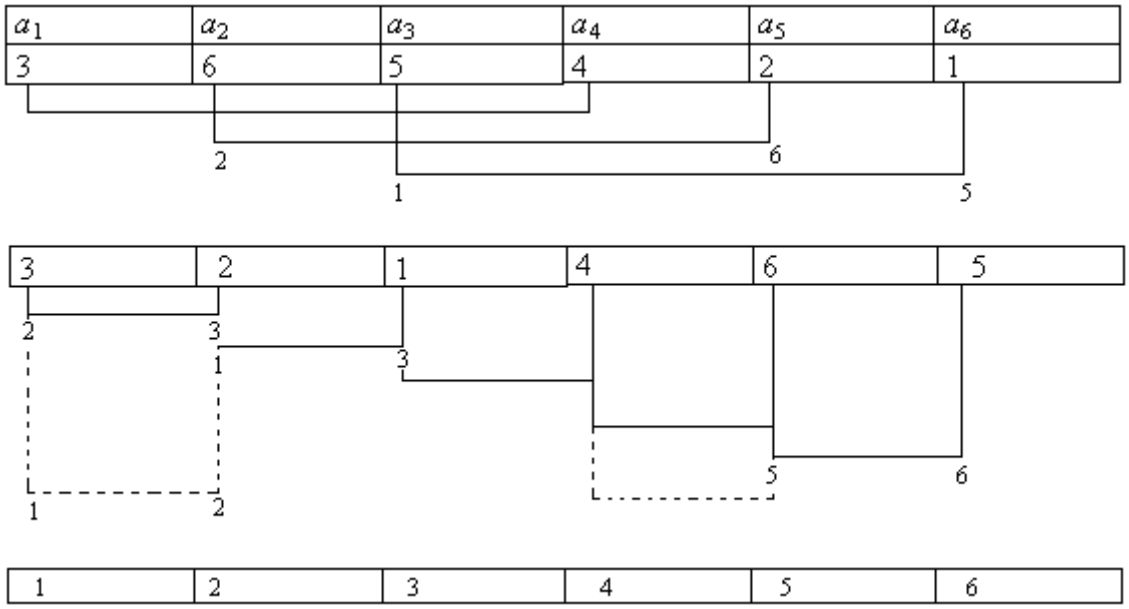


Program:

```
var i,j,n,pom,pom1:integer;
    a: array[1..500] of integer;
    label L1;
begin
     write('Zadej pocet prvku:');   readln(n);
     writeln('Zadej počet prvků: ');for i:=1 to n do
                                     readln(a[i]);
               for i:=1 to n do
          pom1=1;
          begin
               for j:=1 to n-i do
                   if a[j]<a[j+1] then
                      begin
                          pom:=a[j];
                          a[j]:=a[j+1];
                          a[j+1]:=pom;
                        pom1=0;
                      end;
           if pom1=1 then gozo L1;

     writeln('setříděná řada:');
     for i:=1 to n do
     write(a[i]:10);
     end.
```

90. Sestavme VD resp. program pro setřídění řady čísel metodou Shelovou. Tato metoda je podobná metodě bublání, začíná však porovnáním položek ve vzdálenosti *m*, to znamená, že položky, které nejsou na správném místě, se vyměňují mnohem rychleji než při jednoduchém třídění metodou bublání. Po každém průchodu celou řadou se hodnota *m* snižuje podle vztahu m $m_{i+1}=m_i/2$ a každá položka se porovná s položkou umístěnou ve vzdálenosti *m*. Jestliže má první položka vyšší hodnotu, provede se výměna. Porovnají se položky a_i s a_{i+m}, kde $m=n/2$, $n/4$,..Tento proces se opakuje, dokud existuje dvojice pro porovnání. V případě, že $a_i > a_{i+m}$, prvky se vzájemně přemístí, aby se nenarušilo dosud vytvořené setřídění.

Příklad:



Náročnější příklady

91. Sestavme VD resp. program pro výpočet diferenciální rovnice metodou Runge-Kutta. K řešení použijeme vztahy

$$Y_1 = Y_0 + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$

$$k_1 = fce\left(x_0, Y_0\right)$$

$$k_2 = fce\left(x_0 + \frac{h}{2}, Y_0 + \frac{1}{2}hk_1\right)$$

$$k_3 = fce\left(x_0 + \frac{h}{2}, Y_0 + \frac{1}{2}hk_2\right)$$

$$k_4 = fce\left(x_0 + h, Y_0 + hk_3\right)$$

Například uvažujme $y' = z + xy^2$, h=0,2, Y_0=1 a x_0 =0.

92. Sestavme VD resp. program pro numerický výpočet integrálu obdélníkovou metodou. Uvažujme spojitou funkci v uzavřeném interval <*a,b*> pro stejná dělení *h*. K výpočtu použijme vztah

$\int_a^b f(x)dx \cong h\sum_{i=1}^{n} y_i$ pro $h = \frac{b-a}{n}$, kde *n* je počet dělení, *n* +1 je počet souřadnic.

93. Sestavme VD resp. program pro numerický výpočet integrálu lichoběžníkovou metodou. Uvažujme spojitou funkci *f*(*x*) v uzavřeném interval <*a,b*> pro stejná dělení *h*.

K výpočtu použijme vztah

$$\int_a^b f(x)dx \cong h\left[\frac{y_1}{2} + \frac{y_{n+1}}{2} + \sum_{i=2}^{n} y_i\right] \text{ pro } h = \frac{b-a}{n}$$

94. Sestavme VD resp. program pro výpočet numerické integrace *f*(*x*) dle vztahu

$y_i' = \frac{-y_i + y_{i+1}}{h}$ pro *i* =1, 2, *n*, kde *n*+1 je počet bodů, ve kterých derivaci určujeme a

$h = \frac{b-a}{n}$.

95. Sestavme VD resp. program pro součin dvou matic **C=A** . **B**

$$\begin{bmatrix} c_{11} = a_{11.}b_{11} + a_{12.}b_{21} + .. + a_{1N.}b_{N1}; c_{12} = a_{11.}b_{12} + a_{12.}b_{22} + .. + a_{1N.}b_{N2}\ ;.....c_{11} = a_{11.}b_{1N} + a_{12.}b_{2N} + .. + a_{1N.}b_{NN} \\ c_{21} = a_{21.}b_{11} + a_{22.}b_{21} + .. + a_{2N.}b_{M1}; c_{22} = a_{21.}b_{12} + a_{22.}b_{22} + .. + a_{2N.}b_{N2}\ ;...c_{21} = a_{21.}b_{1N} + a_{22.}b_{2N} + .. + a_{2N.}b_{NN} \\ ..... \\ c_{N1} = a_{N1.}b_{11} + a_{N2.}b_{21} + ..a_{NN.}b_{N1}; c_{N2} = a_{N1.}b_{12} + a_{N2.}b_{22} + . + a_{NN.}b_{N2}\ ;.....c_{NN} = a_{N1.}b_{1N} + a_{N2.}b_{2N} + ..a_{NN.}b_{NN}. \end{bmatrix}$$