Praktická časť bakalárskej práce s názvom „**Vytvorenie zbierky riešených príkladov k výuke predmetu Počítače a algoritmizácia – internetový prístup**“

vypracoval: Peter Kostelník

študijný program: Hospodárska informatika

vedúci práce: doc. Ing. Anna Jadlovská, PhD.

konzultant: Ing. Štefan Jajčišin

Košice 2012

1. Sekvenčné algoritmy

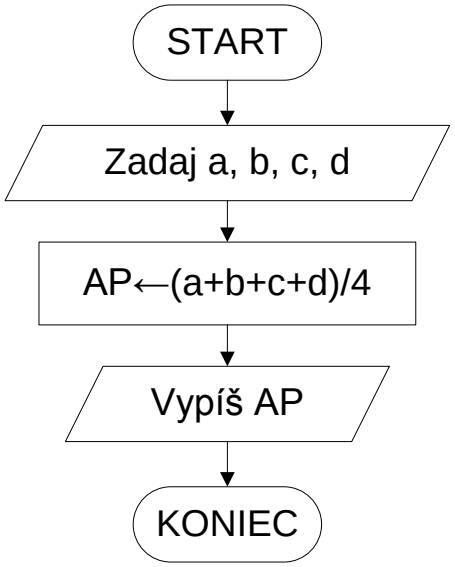
- najjednoduchšia forma algoritmov
- neobsahuje vetvenia ani opakovania
- jednotlivé kroky sa vykonávajú jednotlivo po sebe ako sú zobrazené vo vývojovom diagrame
- každý krok sa vykoná len raz
- nie je možné riešiť zložitejšie úlohy

1. Vytvorte algoritmus na výpočet aritmetického priemeru zo 4 čísel zadaných užívateľom.

Vstupné premenné: a, b, c, d
Výstupná premenná: AP

Analýza riešenia:
Užívateľ bude vyzvaný k zadaniu štyroch čísel postupne. Výpočet aritmetického priemeru spočíva zo sčítania všetkých čísel (a až d) a následne daný súčet vydelíme počtom zadaných čísel, v tomto prípade 4.

Slovný popis algoritmu:
1. krok: načítať hodnoty čísel a, b, c, d
2. krok: sčítať všetky čísla a vydeliť ich počtom (4)
3. krok: vytlačiť AP



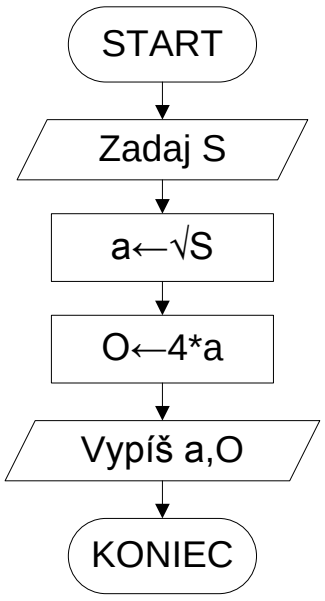

```
int main(void)
{
  float a,b,c,d;
  float AP;

  printf("Algoritmus pre vypocet AP 4 zadanych cisel \n\n");

  printf("Zadaj cislo a: ");
  scanf("%f",&a);
  printf("Zadaj cislo b: ");
  scanf("%f",&b);
  printf("Zadaj cislo c: ");
  scanf("%f",&c);
  printf("Zadaj cislo d: ");
  scanf("%f",&d);

  AP=(a+b+c+d)/4;

  printf("\nArit. priemer zadanych cisel je: %0.2f \n\n", AP);
  system("PAUSE");
  return 0;
}
```

2. Vytvorte algoritmus, ktorý zo zadaného obsahu štvorca vypočíta jeho obvod. Na konci vypíšte obvod štvorca aj dĺžku strany štvorca.

Vstupné premenné: S
Výstupné premenné: a, O

Analýza riešenia:
Keďže vzorec na výpočet štvorca je O=4*a je potrebné zo zadaného obsahu vyjadriť dĺžku strany a => a=√S. Po zistení dĺžky strany štvorca je obvod jednoducho vypočítaný podľa vzorca.

Slovný popis algoritmu:
1. krok: načítať obsah štvorca S
2. krok: výpočet strany a
3. krok: výpočet obvodu štvorca O
4. krok: vytlačiť a, O



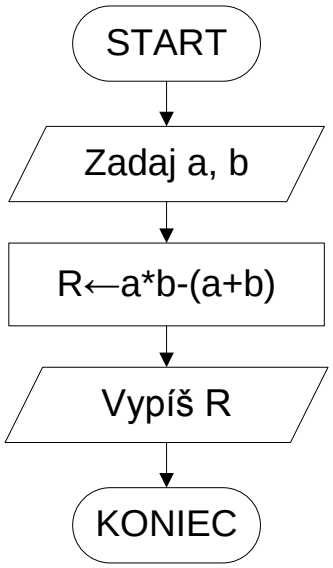

```
int main(void)
{
float a,S,O;

printf("Vypocet obvodu stvorca zo zadaneho obsahu\n\n");

printf("Zadaj obsah stvorca S v cm2: ");
scanf("%f",&S);

a=sqrt(S);
O=4*a;

printf("\nObvod stvorca so stranou a=%0.2f cm je %0.2f cm.\n",a,O);

system("PAUSE");
return 0;
}
```

3. Vytvorte algoritmus, ktorý vypíše rozdiel súčinu a súčtu dvoch zadaných čísel.

Vstupné premenné: a,b
Výstupná premenná: R

Analýza riešenia:
Po zadaní dvoch ľubovoľných čísel, program vypočíta ich súčin a následne od neho odčíta súčet týchto dvoch čísel. Výsledok uloží do výstupnej premennej R a vypíše na monitor.

Slovný popis algoritmu:
1. krok: načítať 2 čísla zadané užívateľom
2. krok: vypočítať rozdiel R=a*b-(a+b)
3. krok: vypísať R na monitore




```
int main(void)
{
  int a,b;
  int R;

   printf("Rozdiel sucinu a suctu cisel \n\n");

  printf("Zadaj 1. cislo: ");
  scanf("%d",&a);
  printf("Zadaj 2. cislo: ");
  scanf("%d",&b);

  R=a*b-(a+b);

  printf("\nRozdiel sucinu a suctu cisel %d a %d je %d. \n\n",a,b,R);
  system("PAUSE");
  return 0;
}
```

2. Algoritmy s vetvením

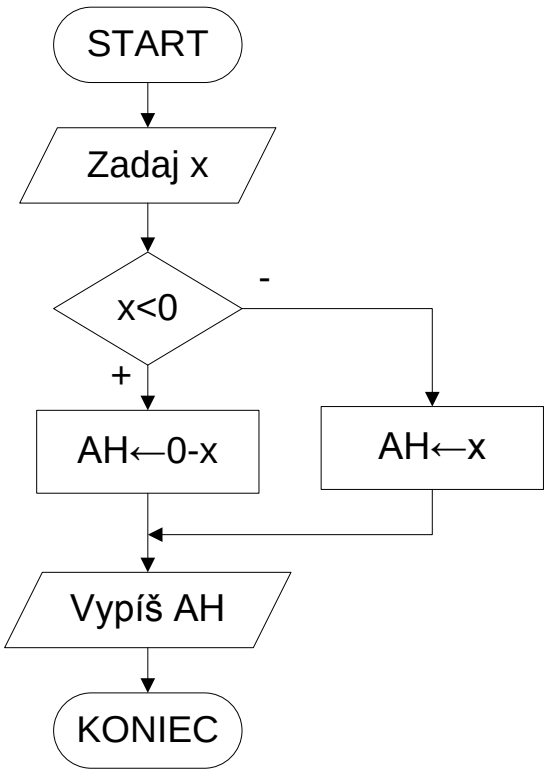
- vetvenie umožňuje pokračovať v programe rôznymi cestami
- na základe splnenia, resp. nesplnenia podmienky sa vyberajú ďalšie kroky programu
- umožňuje ošetriť program aby vypísal chybovú hlášku pri nesprávnom zadaní vstupu
- poznáme niekoľko typov vetvenia:
  preskok – pri splnení podmienky sa vykoná daný program, pri nesplnení sa program preskočí
  dvojcestné/trojcestné vetvenie – vykoná sa jedna z dvoch/troch alternatív podľa podmienky
  prepínač – má nekonečne veľa alternatív ako môže program pokračovať

1. Vytvorte algoritmus pre zistenie absolútnej hodnoty zadaného čísla.

Vstupná premenná: x
Výstupná premenná: AH

Analýza riešenia:
Absolútna hodnota čísla predstavuje vzdialenosť obrazu čísla na číselnej osi od čísla 0. Absolútna hodnota je vždy kladné číslo. Ak je číslo zadané na vstupe kladné, potom abs. hodnota zadaného čísla je jeho hodnota, teda AH=x. Pri zápornom čísle ide o číslo opačné, teda absolútnu hodnotu môžeme zistiť viacerými spôsobmi, napr. odčítaním čísla x od čísla 0 alebo vynásobením čísla x číslom -1.

Slovný popis algoritmu:
1. krok: zadaj číslo x
2. krok: zistiť či je $x < 0$
3. krok: ak číslo $x < 0$ – prejsť na 4. krok
4. krok: AH ← 0 – x
5. krok: ak číslo $x \geq 0$ – prejsť na 6. krok
6. krok: AH ← x
7. krok: vytlačiť AH



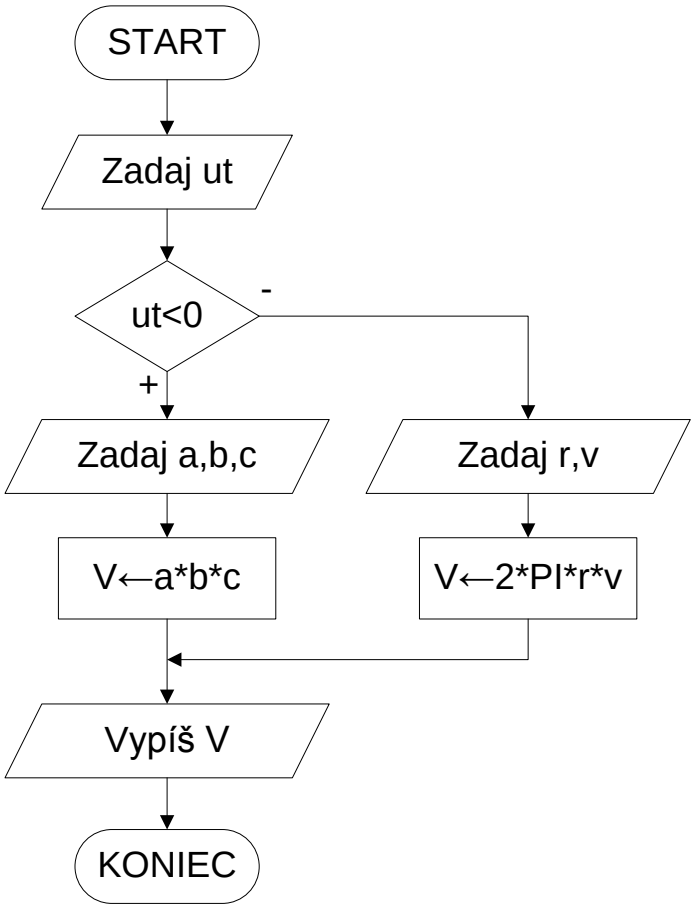

```
int main(void)
{
  float x;
  float AH;

  printf("Absolutna hodnota cisla \n\n");

  printf("Zadaj cislo x: ");
  scanf("%f",&x);

  if(x<0){
            AH=0-x;
            }
               else{
                     AH=x;
                   }

 printf("\nAbs. hodnota cisla %0.2f je %0.2f. \n\n",x,AH);
 system("PAUSE");
 return 0;
}
```

2. Vytvorte algoritmus pre výpočet objemu kvádra, prípadne valca pričom užívateľ sám určí či chce počítať objem kvádra alebo valca zadaním čísla do premennej útvar (ut). Pre zjednodušenie použijeme pre premennú „ut“ číselné hodnoty. Ak užívateľ zadá číslo menšie ako 0 spustí sa program na výpočet objemu kvádra, ak zadané číslo bude vyššie alebo rovné nule, spustí sa program na výpočet objemu valca.

   Vstupné premenné: ut, a, b, c, r, v
   Výstupná premenná: V

   Analýza riešenia:
   Pred písaním samotnej funkcie si definujeme hodnotu premennej pí príkazom #define PI 3.1415926. Program vyzve užívateľa k zadaniu útvaru, ktorého objem chce počítať. Použijeme vetvenie aby si užívateľ mohol vybrať požadovaný program. Pre zjednodušenie sme použili pre „ut“ typ premennej int. Ak má „ut“ zápornú hodnotu, počíta sa objem kvádra, v inom prípade počítame objem valca.

   Slovný popis algoritmu:
   1. krok: zadaj číslo ut
   2. krok: zistiť či je ut < 0
   3. krok: ak je ut < 0 – prejsť na krok 4



```
#include <stdio.h>
#define PI 3.1415926

int main(void)
{
  int ut;
  float a,b,c,r,v,V;

  printf("Vypocet objemu kvadra(-)/valca(+) \n\n");

  printf("Zadaj utvar: ");
  scanf("%d",&ut);

  if(ut<0){
              printf("Zadaj sirku a v cm: ");
              scanf("%f",&a);
              printf("Zadaj dlzku b v cm: ");
              scanf("%f",&b);
              printf("Zadaj vysku c v cm: ");
              scanf("%f",&c);

              V=a*b*c;
              printf("\nObjem kvadra je %0.2f cm3. \n\n",V);
            }
      else{
              printf("Zadaj polomer r v cm: ");
              scanf("%f",&r);
              printf("Zadaj vysku v v cm: ");
              scanf("%f",&v);

              V=2*PI*r*v;

              printf("\nObjem valca je %0.2f cm3.\n\n",V);
            }

 system("PAUSE");
 return 0;
}
```

3. Vytvorte algoritmus, ktorý vypočíta neznámu stranu obdĺžnika. Užívateľ na vstupe zadá, ktorý parameter obdĺžnika pozná (obsah alebo obvod) a tiež, ktorá strana (a alebo b) je známa. Následne bude vyzvaný k zadaniu hodnoty známeho parametra, potom k zadaniu dĺžky známej strany. Pre uľahčenie použijeme na vstupe číselné hodnoty nasledovným spôsobom:
   Parameter:
      1 – obsah
      2 – obvod
   Strana:
      1 – strana a
      2 – strana b

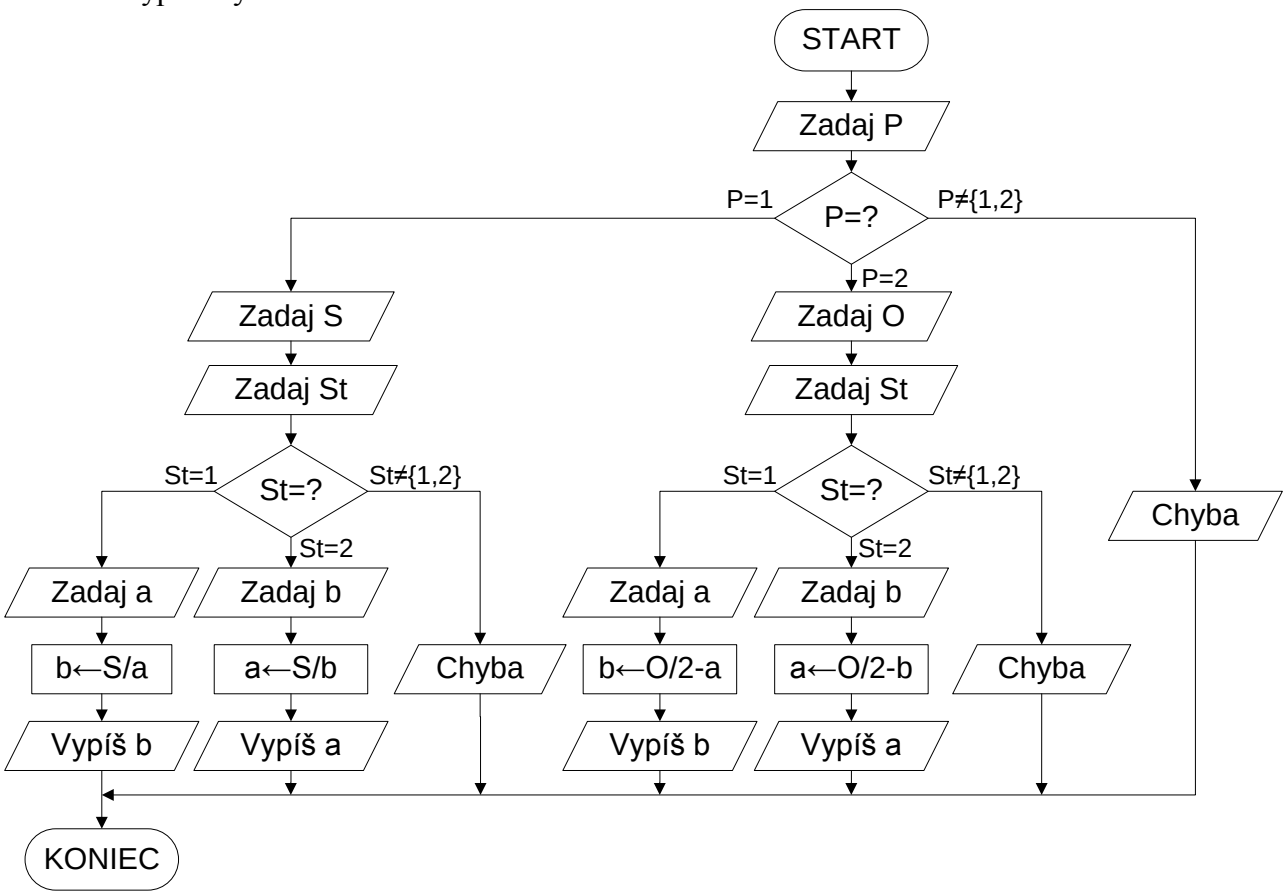
   Použitím trojcestného vetvenia ošetrite program aby pri zadaní čísla iného ako je 1 alebo 2 vypísal chybovú hlášku a následne sa ukončí.

   Vstupné premenné: P, St, S/O, a/b
   Výstupné premenné: a/b

   Analýza riešenia:
   Program začne vyzvaním užívateľa k zadaniu číselnej hodnoty známeho parametra. Po zadaní nasleduje trojcestné vetvenie, ktoré vyzve užívateľa k zadaniu hodnoty daného parametra a následne k zadaniu čísla 1 alebo 2 podľa toho, ktorá strana je známa. Nasleduje ďalšie vetvenie, ktoré rozhodne o ďalšom postupe – buď vyzve užívateľa k zadaniu dĺžky strany a, ak je hodnota premennej St=1 alebo b, ak má premenná St hodnotu 2 alebo vypíše chybové hlásenie pri zadaní akéhokoľvek iného čísla. Po zadaní dĺžky strany zbehne určený program a program vypíše pôvodne neznámu stranu obĺžnika.

   Slovný popis algoritmu:
   1. krok: zadaj známy parameter P
   2. krok: ak P=1 pokračuj krokom 3, inak prejdi na krok 14
   3. krok: zadaj obsah S
   4. krok: zadaj známu stranu St
   5. krok: ak St=1 pokračuj krokom 6, inak prejdi na krok 9
   6. krok: zadaj dĺžku strany a
   7. krok: výpočet strany b=S/a
   8. krok: vypíš dĺžku strany b
   9. krok: ak St=2 pokračuj krokom 10, inak prejdi na krok 13
   10. krok: zadaj dĺžku strany b
   11. krok: výpočet strany a=S/b
   12. krok: vypíš dĺžku strany a
   13. krok: vypíš chybové hlásenie
   14. krok: ak P=2 pokračuj krokom 15, inak prejdi na krok 26
   15. krok: zadaj obvod O
   16. krok: zadaj známu stranu St
   17. krok: ak St=1 pokračuj krokom 18, inak prejdi na krok 21
   18. krok: zadaj dĺžku strany a
   19. krok: výpočet strany b=O/2-a
   20. krok: vypíš dĺžku strany b
   21. krok: ak St=2 pokračuj krokom 22, inak prejdi na krok 25
   22. krok: zadaj dĺžku strany b
   23. krok: výpočet strany a=O/2-b
   24. krok: vypíš dĺžku strany a

25. krok: vypíš chybové hlásenie
26. krok: vypíš chybové hlásenie



```
int main(void)
{
 float S,O,a,b,P,St;

 printf("Vypocet chybajucej strany obdlznika x \n\n");

 printf("Zadaj znamy parameter (1-obsah/2-obvod): ");
 scanf("%f",&P);

 if(P==1){
        printf("Zadaj obsah S: ");
        scanf("%f",&S);
        printf("Zadaj znamu stranu (1-strana a/2-strana b): ");
        scanf("%f",&St);

        if(St==1){
                printf("Zadaj dlzku strany a v cm: ");
                scanf("%f",&a);
                b=S/a;
                printf("\nDlzka strany b je %0.2f cm.\n\n",b);
                }
                else if(St==2){
                        printf("Zadaj dlzku strany b v cm: ");
                        scanf("%f",&b);
                        a=S/b;
                        printf("\nDlzka strany a je %0.2f cm.\n\n",a);
                        }
                        else printf("\nZadanie nebolo spravne!\n\n");
        }

        else if(P==2){
                   printf("Zadaj obvod O: ");
                   scanf("%f",&O);
                   printf("Zadaj znamu stranu (1-strana a/2-strana b): ");
                   scanf("%f",&St);

                   if(St==1){
                          printf("Zadaj dlzku strany a v cm: ");
                          scanf("%f",&a);
                          b=O/2-a;
                          printf("\nDlzka strany b je %0.2f cm.\n\n",b);
                          }
                          else if(St==2){
                                  printf("Zadaj dlzku strany b v cm: ");
                                  scanf("%f",&b);
                                  a=O/2-b;
                                  printf("\nDlzka strany a je %0.2f cm.\n\n",a);
                                  }
                                  else printf("\nZadanie nebolo spravne!\n\n");
                                  }

        else printf("\nZadanie nebolo spravne!\n\n");

 system("PAUSE");
 return 0;
}
```

3. Algoritmy s opakovaním

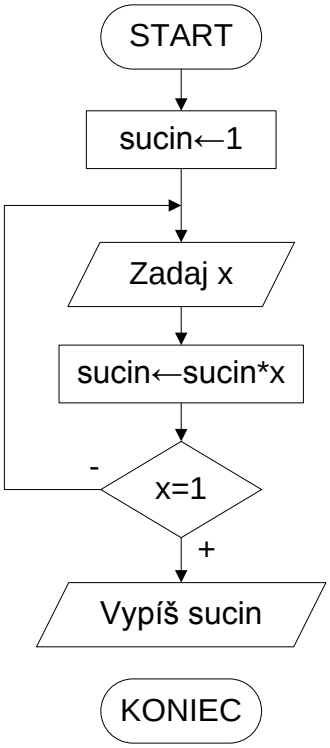
- použitím cyklu si môžeme ušetriť písanie dlhého zdrojového kódu pri programe, v ktorom na vstupe zadávame viac hodnôt do vstupnej premennej
- umožňujú riešenie aj zložitejších úloh opakovaním vybraných krokov
- cyklus môžeme ukončiť rôznymi spôsobmi, napr. zadaním čísla, ktoré uvedieme v podmienke alebo dosiahnutím zadaného počtu vstupov
- poznáme niekoľko typov cyklov:
  - **cyklus s podmienkou na začiatku** – po splnení podmienky sa vykoná telo cyklu, pri nesplnení podmienky cyklus končí, pričom sa telo nemusí vykonať ani raz
  - **cyklus s podmienkou na konci** – telo cyklu sa vykoná minimálne raz, po vykonaní cyklu podmienka rozhodne či sa cyklus zopakuje alebo nie
  - **cyklus s daným počtom opakovaní** – na začiatku sa určí podmienka dokedy má cyklus bežať zadaním tzv. riadiacej premennej i

1. Vytvorte algoritmus, ktorý vypočíta súčin zadaných čísel. Zadávanie čísel bude ukončené číslom 1, aby výsledok nebol skreslený. Použite cyklus WHILE s podmienkou na konci.

   Vstupné premenné: x
   Výstupná premenná: sucin

   Analýza riešenia:
   V prvom rade nastavíme počiatočnú hodnotu premennej sucin na 1. Program vyzve užívateľa k zadaniu čísla a po zadaní vypočíta nový súčin po vynásobení aktuálnej hodnoty súčinu zadaným číslom. Program opätovne vyzýva užívateľa k zadaniu ďalšieho čísla, až kým zadané číslo nie je rovné 1.

   Slovný popis algoritmu:
   1. krok: sucin←1
   2. krok: zadaj x
   3. krok: sucin=sucin*x
   4. krok: ak x = 1 pokračuj krokom 5 inak opakuj krok 2
   5. krok: vypíš sucin



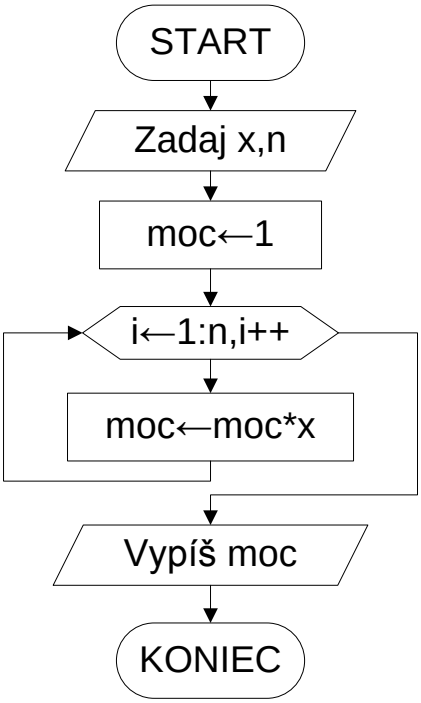

```
int main(void)
{
  int x,sucin=1;

   printf("Sucin zadanych cisel \n\n");

  printf("Zadajte cisla x (ukoncite cislom 1): \n");
  while(1){
              scanf("%d",&x);
              sucin=sucin*x;
               if(x==1)
              break;
            }

 printf("Sucinom zadanych cisel je cislo: %d \n\n",sucin);

 system("PAUSE");
 return 0;
}
```

2. Vytvorte algoritmus na výpočet n-tej mocniny zadaného čísla x.

Vstupné premenné: x, n
Výstupné premenné: moc

Analýza riešenia:
Užívateľ na vstupe zadá dvojicu čísel, prvé číslo, ktoré chce umocniť (x) a následne mocniteľa (n). Hodnotu výstupnej premennej nastavíme na začiatku na 1. Cyklus FOR beží od čísla 1 po n a aktuálnu hodnotu premennej moc násobí číslom x, kým nie je cyklus ukončený. Nakoniec vypíšeme premennú moc na monitor.

Slovný popis algoritmu:
1. krok: zadať vstupné premenné: x, n
2. krok: nastaviť hodnotu výstupnej premennej: moc=1
3. krok: ak je i < n pokračovať krokom 4, inak prejsť na krok 6
4. krok: pom=pom*x
5. krok: i++, vráť sa na krok 3
6. krok: vypísať moc




```
int main(void)
{
 int x,i,n,moc=1;

 printf("Vypocet n-tej mocniny cisla x \n\n");

 printf("Zadaj cislo x: ");
 scanf("%d",&x);

 printf("Zadaj mocnitela n: ");
 scanf("%d",&n);

 for(i=1;i<=n;i++){
            moc=moc*x;
            }

 printf("\nCislo %d umocnene na %d je %d. \n\n",x,n,moc);

 system("PAUSE");
 return 0;
}
```

4. Algoritmy s opakovaním

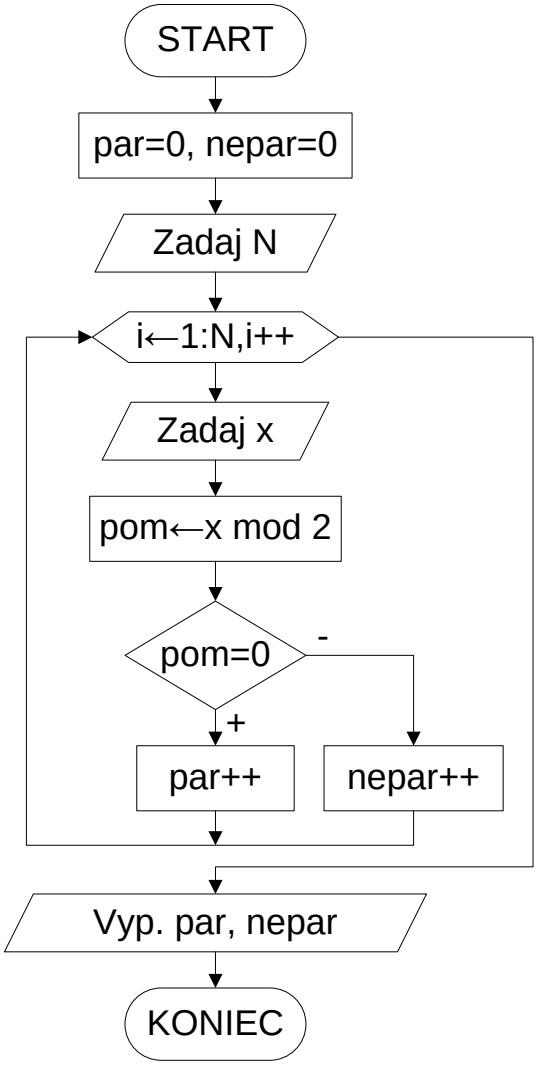
- použitím cyklu si môžeme ušetriť písanie dlhého zdrojového kódu pri programe, v ktorom na vstupe zadávame viac hodnôt do vstupnej premennej
- umožňujú riešenie aj zložitejších úloh opakovaním vybraných krokov
- cyklus môžeme ukončiť rôznymi spôsobmi, napr. zadaním čísla, ktoré uvedieme v podmienke alebo dosiahnutím zadaného počtu vstupov
- poznáme niekoľko typov cyklov:
  - **cyklus s podmienkou na začiatku** – po splnení podmienky sa vykoná telo cyklu, pri nesplnení podmienky cyklus končí, pričom sa telo nemusí vykonať ani raz
  - **cyklus s podmienkou na konci** – telo cyklu sa vykoná minimálne raz, po vykonaní cyklu podmienka rozhodne či sa cyklus zopakuje alebo nie
  - **cyklus s daným počtom opakovaní** – na začiatku sa určí podmienka dokedy má cyklus bežať zadaním tzv. riadiacej premennej i

1. Vytvorte algoritmus, ktorý zo zadaných N čísel vypíše koľko z nich je párnych a koľko nepárnych.

   Vstupné premenné: N, x
   Pomocné premenné: pom
   Výstupné premenné: par, nepar

   Analýza riešenia:
   Párne číslo je číslo, ktoré po delení číslom 2 dáva zvyšok 0. Použijeme cyklus FOR, kde pomocnej premennej pom priradíme číslo rovné zvyšku po delení čísla x číslom 2. Ak je hodnota pom rovná 0, zvýšime počet párnych čísel o 1, inak zvýšime o 1 počet nepárnych čísel.

   Slovný popis algoritmu:
   1. krok: nastaviť hodnoty par=0, nepar=0
   2. krok: zadať číslo N
   3. krok: kým je i < N opakuj kroky 4 až 6, inak prejdi na krok 9
   4. krok: zadaj číslo x
   5. krok: premennej pom priraď x mod 2
   6. krok: ak pom=0 pokračuj krokom 7, inak 8
   7. krok: zvýšiť par o 1
   8. krok: zvýšiť nepar o 1
   9. krok: vypíš par, nepar




```
int main(void)
{
  int N,x,par=0,nepar=0,i,pom;

   printf("Zistenie pocut parnych a neparnych cisel z
       N zadanych cisel\n\n");

  printf("Zadaj pocet cisel N: ");
  scanf("%d",&N);

  for(i=1;i<=N;i++){
              printf("Zadaj x: ");
              scanf("%d",&x);
              pom=x%2;
              if(pom==0){
                              par++;
                              }
                              else nepar++;
              }

   printf("\nZo zadanych %d cisel je pocet parnych
       cisel: %d a neparnych: %d.\n\n",N,par,nepar);

 system("PAUSE");
 return 0;
}
```

5. Algoritmy pre výpočet hodnôt funkcie

- pre korektný výpočet sa používa rozvoj do radov
- nasledujúci člen je vyjadrený dosadením predchádzajúceho člena do rekurentného vzorca
- ukončenie rozvoja je dané presnosťou E

1. Zostavte algoritmus pre výpočet F(x) podľa vzťahu:

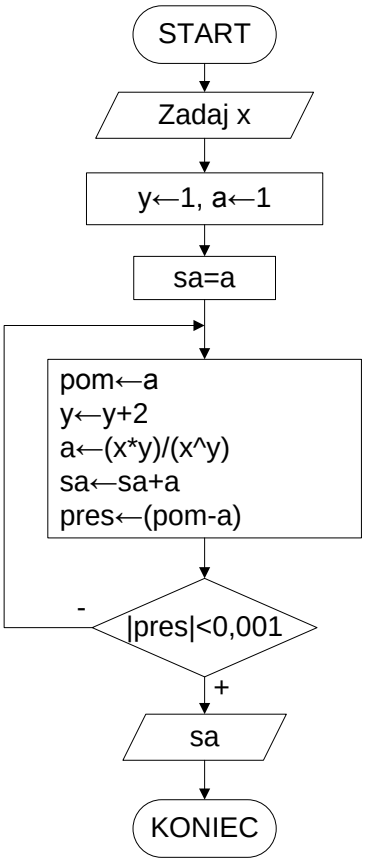
$$F(x) = 1 + \frac{x \cdot 3}{x^3} + \frac{x \cdot 5}{x^5} + \frac{x \cdot 7}{x^7} + \cdots + \frac{x \cdot n}{x^n}$$

Uvažujte taký počet členov rozvoja radu, aby rozdiel medzi poslednými dvomi členmi bol menej ako 0,001.

Vstupné premenné: x
Pomocné premenné: a, y, pom, pres
Výstupná premenná: sa

Analýza riešenia:
Pre zvyšovanie násobiteľov a mocniteľov čísla x využijeme pomocnú premennú y, ktorej hodnotu budeme zvyšovať o 2. Pri hodnote premennej y=1 je funkčná hodnota rovná 1, teda hodnotu prvého člena radu a nastavíme na začiatku na 1. Následne v cykle zvyšujeme hodnotu y o 2 a počítame hodnotu ďalších členov radu. Výpočet ukončíme vtedy, ak rozdiel dvoch po sebe nasledujúcich členov je menej ako 0,001. Na zistenie rozdielu nám slúži pomocná premenná pres, ktorá je rozdielom premennej pom, do ktorej sme uložili hodnotu predchádzajúceho člena a premennej a, čo je vlastne aktuálne vypočítaný člen radu.

Slovný popis algoritmu:
1. krok: načítať hodnotu x
2. krok: nastaviť hodnoty pomocných premenných: y=1, a=1
3. krok: výstupnej premennej sa priradiť hodnotu prvého člena radu: sa=a
4. krok: priradiť nasledujúcim premenným dané hodnoty:
   pom = a
   y = y+2
   $a = \frac{x \cdot y}{x^y}$
   sa = sa + a
   pres = pom – a
5. krok: ak |pres| < 0,001 pokračuj na krok 6, inak opakuj krok 4
6. krok: vypísať hodnotu výstupnej premennej sa



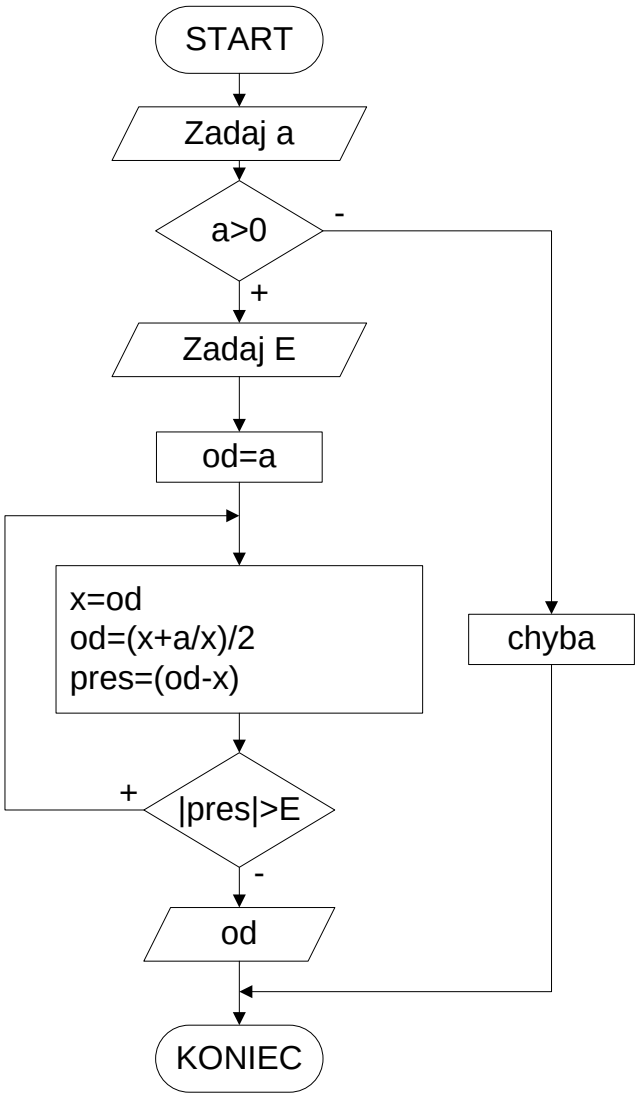

```
int main(void)
{
  printf("Vypocet hodnoty funkcie\n\n");
  float y=1,sa,a=1,x,pom,pres;
  printf("Zadaj x: ");
  scanf("%f",&x);

  sa=a;
  while(1){
              pom=a;
              y=y+2;
              a=(x*y)/pow(x,y);
              sa=sa+a;
              pres=pom-a;
              if(fabs(pres)<0.001)
              break;
            }
 printf("Hodnota danej funkcie pre x=%0.2f je %0.5f.\n\n",x,sa);

 system("PAUSE");
 return 0;
}
```

2. Navrhnite algoritmus pre výpočet druhej odmocniny $X = \sqrt{a}$ pričom a > 0. Pre výpočet použite Newtonov iteračný vzorec: $x_{i+1} = \frac{1}{2}(x_i + a/x_i)$, kde i = 0, 1, 2 . . . n. V prípade zadania čísla a ≤ 0, program vypíše chybu.

Vstupné premenné: a, E
Pomocné premenné: x, pres
Výstupné premenné: od

Analýza riešenia:
Prvou vstupnou premennou je číslo a, z ktorého sa bude počítať odmocnina. Použijeme podmienku či je zadané číslo väčšie ako 0, pretože číslo umocnené na druhú nemôže mať záporný výsledok. Pri nesplnení podmienky program vypíše chybu a ukončí sa, ak je podmienka splnená, používateľ bude vyzvaný k zadaniu presnosti výpočtu. Premennej od priradíme najprv hodnotu premennej a a následne spustíme cyklus, v ktorom je hodnota premennej od priradená premennej x, čo nám slúži na výpočet presnosti v premennej pres. Cyklus sa opakuje dovtedy, kým rozdiel dvoch po sebe nasledujúcich členov nie je menší ako zadaná presnosť.

Slovný popis algoritmu:
1. krok: zadaj číslo a
2. krok: ak je a > 0 pokračuj krokom 3, inak vypíš chybu a ukonči program
3. krok: zadaj presnosť E
4. krok: výstupnej premennej od priraď číslo a
5. krok: nasledujúcim premenným priraď tieto hodnoty:
   x = od
   $od = \frac{1}{2}(x + \frac{a}{x})$
   pres = od – x
6. krok: ak je |pres| > E, opakuj krok 5, inak prejdi na krok 7
7. krok: vypíš hodnotu druhej odmocniny od




```
int main(void)
{
 printf("Vypocet 2. domocniny zo zadaneho
cisla\n\n");
 float a,E,od,pres,x;

 printf("Zadaj a: ");
 scanf("%f",&a);

if(a>0)
{
 printf("Zadaj E: ");
 scanf("%f",&E);
 od=a;
 do
  {
   x=od;
   od=(x+a/x)/2;
   pres=(od-x);
  }
  while(fabs(pres)>E);

  printf("\nDruha odmocnina z cisla %0.2f je
%0.2f.\n\n",a,od);

}
else printf("Chybne cislo!\n\n");

 system("PAUSE");
 return 0;
}
```

6. Algoritmy pre výpočet hodnôt funkcie

- pre korektný výpočet sa používa rozvoj do radov
- nasledujúci člen je vyjadrený dosadením predchádzajúceho člena do rekurentného vzorca
- ukončenie rozvoja je dané presnosťou E

1. Vytvorte algoritmus pre výpočet goniometrickej funkcie kosínus s argumentom x. Pre výpočet použijeme nasledujúci vzťah:

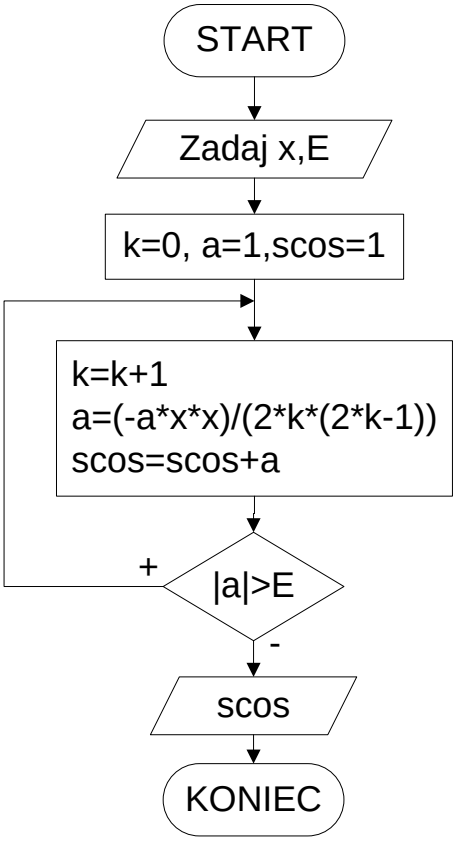
$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$$

Vstupné premenné: x, E
Pomocné premenné: k, b
Výstupné premenné: scos

Analýza riešenia:
Na začiatku používateľ zadá hodnotu čísla, ktorého kosínus chce zistiť (x). Následne zadá presnosť, s akou chce výsledok. Hodnotu premennej k (krok) nastavíme na 0, čo pri prvom behu cyklu zabezpečí hodnotu kroku 1. Následne z predchádzajúceho člena radu určíme ďalší člen a pričítame ho do premennej scos. Následný súčet jednotlivých členov scos vytlačíme, v prípade, že je hodnota aktuálneho člena radu nižšia ako požadovaná presnosť.

Slovný popis algoritmu:
1. krok: zadaj číslo x a presnosť E
2. krok: nastav hodnoty pomocných premenných k=0, a=1 a hodnotu výst. prem. scos=1
3. krok: nasledujúcim premenným priraď dané hodnoty:
   k=k+1
   a=(-a*x*x)/(2*k*(2*k-1))
   scos=scos+a
4. krok: ak |a| > E opakuj krok 3, inak prejdi na krok 5
5. krok: vypíš scos

```
int main()
{
 float x,e,a,scos;
 int k;
 printf("\nZadaj X:");
 scanf("%f",&x);
 printf("\nZadaj E:");
 scanf("%f",&e);

 k=0;
 a=1;
 scos=1;

 do
 {
 k++;
 a=(-a*x*x)/(2*k*(2*k-1));
 scos=scos+a;
 }
 while(fabs(a)>e);

 printf("\nHodnota funkcie cos %0.2f je :  %f \n
\n",x,scos);

 system("PAUSE");
 return 0;
}
```


START
Zadaj x,E
k=0, a=1,scos=1
k=k+1
a=(-a*x*x)/(2*k*(2*k-1))
scos=scos+a
|a|>E
+
-
scos
KONIEC

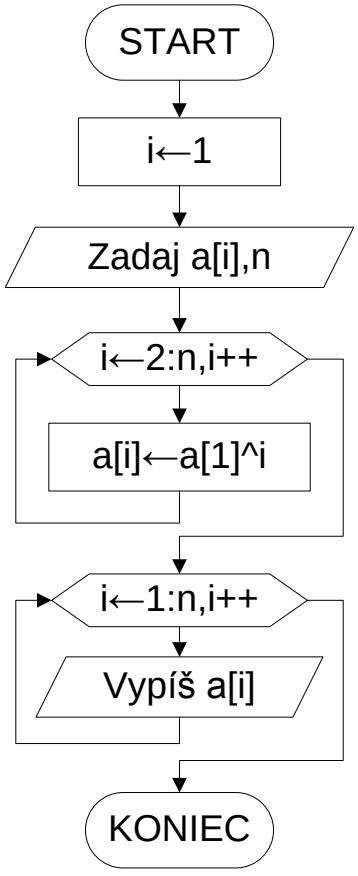
7. Algoritmy s indexovanou premennou

1. Vytvorte algoritmus, ktorý vygeneruje čísla v poli nasledujúcim spôsobom:
   Užívateľ zadá na vstupe číslo *a* a počet členov poľa *n*. Program následne vypočíta 1., 2., 3., ... , n-tú mocninu čísla a. Výsledky uloží do poľa, ktoré nakoniec vytlačí.

   Vstupné premenné: a[1], n
   Výstupná premenná: a[i]

   Analýza riešenia:
   Na začiatku nastavíme premennej index i hodnotu 1. Zadané číslo program potom uloží do člena poľa s indexom i, teda do a[1]. Následne cyklom FOR zabezpečíme výpočet ďalších členov poľa. Index i nastavíme na začiatku cyklu na 2, keďže hodnotu a[1] už máme známu. Index postupne zvyšujeme o 1 pričom vždy do premennej a[i] vložíme zadané číslo a[1] umocnené na i. Nakoniec použijeme ďalší cyklus FOR na vypísanie celého poľa, pričom index na začiatku nastavíme tento krát na 1, aby bolo vypísané celé pole.

   Slovný popis algoritmu:
   1. krok: nastav hodnotu i=1
   2. krok: zadaj číslo a[1] a počet členov radu n
   3. krok: nastav hodnotu i=2 a ak i≤n pokračuj krokom 4, inak prejdi na krok 5
   4. krok: a[i]←a[1]^i, i++, vráť sa na krok 3
   5. krok: nastav hodnotu i=1 a ak i≤n pokračuj krokom 6, inak ukonči program
   6. krok: vypíš a[i], i++, vráť sa na krok 5




```
int main()
{
 int i=1,n,a[i];

 printf("Zadaj pocet prvkov pola n: ");
 scanf("%d",&n);

 printf("Zadaj hodnotu cisla a: ");
 scanf("%d",&a[i]);

 printf("\n");

 for(i=2;i<=n;i++){
                    a[i]=pow(a[1],i);
                   }

 for(i=1;i<=n;i++){
                    printf("a[%d]=%d\n",i,a[i]);
                   }
 printf("\n");

 system("PAUSE");
 return 0;
}
```

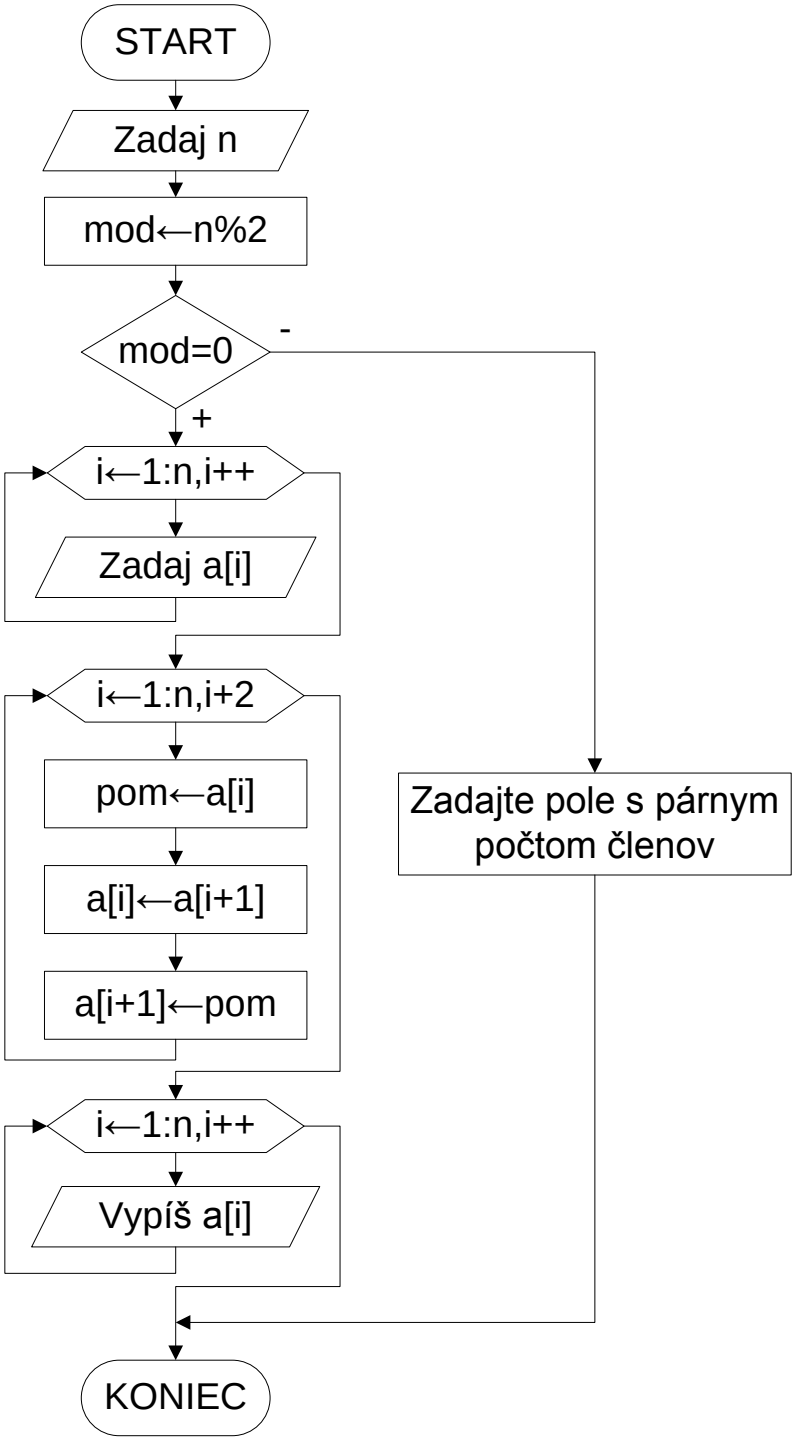
2. Vytvorte algoritmus, ktorý v zadanom poli vymení každé 2 susedné prvky (1. s 2., 3. so 4., 5. so 6., ... , n-1. s n.). Na vstupe užívateľ zadá počet prvkov poľa n a jednotlivé prvky poľa a[i]. Program ošetrite aby vypísal chybu pri zadaní poľa s nepárnym počtom prvkov.

Vstupné premenné: n, a[i]
Pomocné premenné: pom, mod
Výstupné premenné: a[i]

Analýza riešenia:
Užívateľ najprv zadá počet prvkov poľa n. Následne program overí, či je počet členov párny overením zvyšku po delení počtu členov číslom 2. V prípade, že zvyšok po delení nie je nula, program vypíše chybovú hlášku. V opačnom prípade vyzve program užívateľa k zadaniu n čísel. Po zadaní čísel nasleduje cyklus FOR, v ktorom je index i zvyšovaný o 2 a vymieňa hodnoty premenných a[i] a a[i+1] pomocou premennej pom. Nakoniec program vypíše pôvodné aj zmenené pole.

Slovný popis algoritmu:
1. krok: zadaj n
2. krok: premennej mod priraď hodnotu mod=n%2
3. krok: ak mod=0 pokračuj krokom 4, inak vypíš chybu a ukonči program
4. krok: nastav i=1, ak je i≤n pokračuj krokom 5, inak prejdi na krok 6
5. krok: zadaj a[i], i++, vráť sa na krok 4
6. krok: nastav i=1, ak je i≤n pokračuj krokom 7, inak prejdi na krok 10
7. krok: premennej pom priraď hodnotu a[i]
8. krok: premennej a[i] priraď hodnotu a[i+1]
9. krok: premennej a[i+1] priraď hodnotu pom, i=i+2, vráť sa na krok 6
10. krok: nastav i=1, ak je i≤n pokračuj krokom 11, inak ukonči program
11. krok: vypíš a[i], i++, vráť sa na krok 10

```
int main()
{
int i=1,n,a[i],pom,mod;
printf("Zadaj pocet prvkov pola n: ");
scanf("%d",&n);
printf("\n");
mod=n%2;

if(mod==0){
for(i=1;i<=n;i++){
printf("Zadaj a[%d]: ",i);
scanf("%d",&a[i]);
}
printf("\na=[");
for(i=1;i<=n;i++){
printf(" %d ",a[i]);
}
printf("]");
for(i=1;i<=n;i=i+2){
pom=a[i];
a[i]=a[i+1];
a[i+1]=pom;
}
printf("\na=[");
for(i=1;i<=n;i++){
printf(" %d ",a[i]);
}
printf("]\n\n");
}
else printf("Zadajte pole s parnym poctom clenov!\n\n");
system("PAUSE");
return 0;
}
```

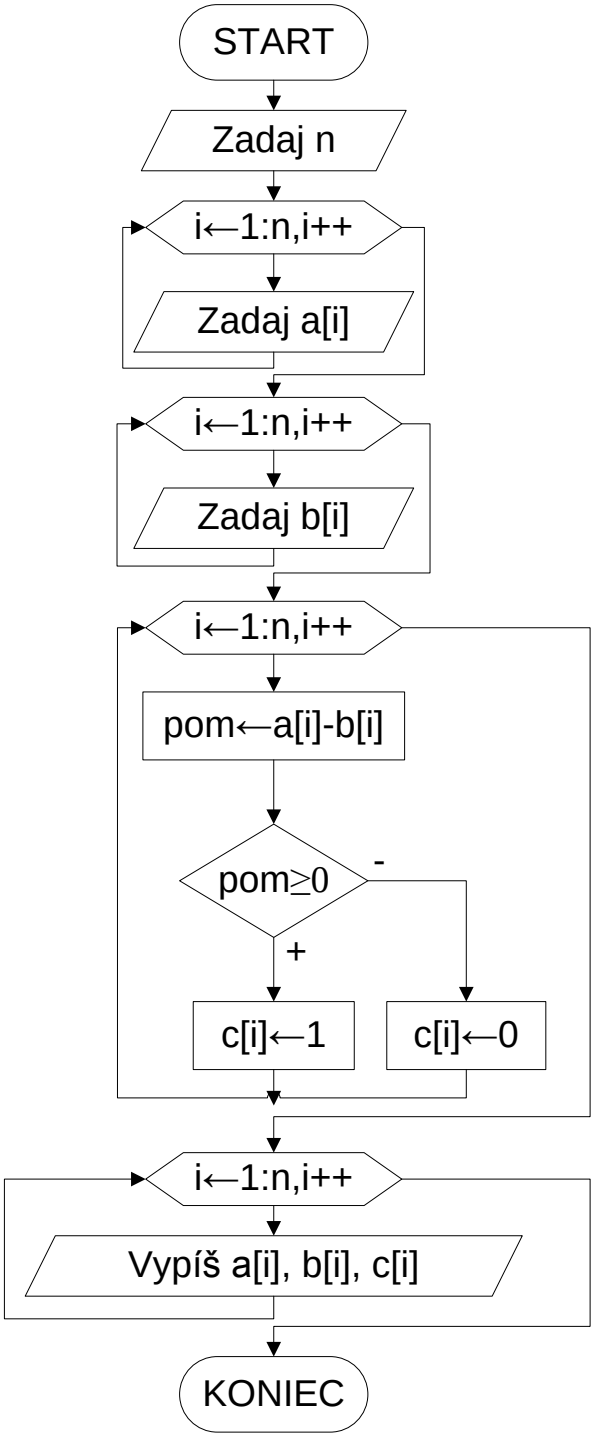

START
Zadaj n
mod←n%2
mod=0
-
+
i←1:n,i++
Zadaj a[i]
i←1:n,i+2
pom←a[i]
a[i]←a[i+1]
a[i+1]←pom
i←1:n,i++
Vypíš a[i]
Zadajte pole s párnym počtom členov
KONIEC

3. Vytvorte algoritmus, ktorý porovná 2 polia zadané užívateľom na vstupe. Polia budú mať rovnakú dĺžku n, ktorú určí používateľ. Program odčíta prvok poľa B od prvku poľa A na rovnakom mieste a ak je výsledok väčší alebo rovný 0, do zodpovedajúceho prvku v poli C vloží číslo 1, v opačnom prípade do C vloží číslo 0.

   Vstupné premenné: n, a[i], b[i]
   Pomocné premenné: pom
   Výstupné premenné: c[i]

   Analýza riešenia:
   Užívateľ zadá dĺžku poľa n pre oba polia, A aj B. Cykly FOR pri zadávaní čísel bežia až kým hodnota indexu nedosiahne hodnotu premennej n. Nasleduje ďalší cyklus FOR, ktorý zabezpečuje počítanie rozdielu polí A a B, rozdiel uloží do premennej pom. Po vypočítaní rozdielu program zistí, či je premenná pom väčšia alebo rovná 0, ak áno, do premennej c[i] vloží číslo 1, ak nie, do premennej c[i] vloží 0. Nakoniec vypíšeme vedľa seba čísla a[i], b[i] a c[i] pre ľahšie overenie funkčnosti programu.

   Slovný popis algoritmu:
   1. krok: zadaj n
   2. krok: nastav i=1, ak i≤n prejdi na krok 3, inak preskoč na krok 4
   3. krok: zadaj a[i], i++, vráť sa na krok 2
   4. krok: nastav i=1, ak i≤n prejdi na krok 5, inak preskoč na krok 6
   5. krok: zadaj b[i], i++, vráť sa na krok 4
   6. krok: nastav i=1, ak i≤n prejdi na krok 7, inak preskoč na krok 11
   7. krok: premennej pom priraď hodnotu pom=a[i]-b[i]
   8. krok: ak pom≥0, pokračuj krokom 9, inak preskoč na krok 10
   9. krok: premennej c[i] priraď hodnotu 1, i++, vráť sa na krok 6
   10. krok: premennej c[i] priraď hodnotu 0, i++, vráť sa na krok 6
   11. krok: nastav i=1, ak i≤n prejdi na krok 12, inak ukonči program
   12. krok: vypíš a[i], b[i] a c[i], i++, vráť sa na krok 11




```
int main()
{
 int i=1,n,a[i],b[i],c[i],pom;

 printf("Zadaj pocet prvkov pola n: ");
 scanf("%d",&n);
 printf("\n");

 for(i=1;i<=n;i++){
            printf("Zadaj a[%d]: ",i);
            scanf("%d",&a[i]);
            }

 for(i=1;i<=n;i++){
            printf("Zadaj b[%d]: ",i);
            scanf("%d",&b[i]);
            }

 for(i=1;i<=n;i++){
            pom=a[i]-b[i];
            if(pom>=0){
                  c[i]=1;
                  }
            else c[i]=0;
            }

 printf("\n");

 for(i=1;i<=n;i++){
            printf("a[%d]=%d, b[%d]=%d,
c[%d]=%d \n",i,a[i],i,b[i],i,c[i]);
            }

 printf("\n");

 system("PAUSE");
 return 0;
}
```

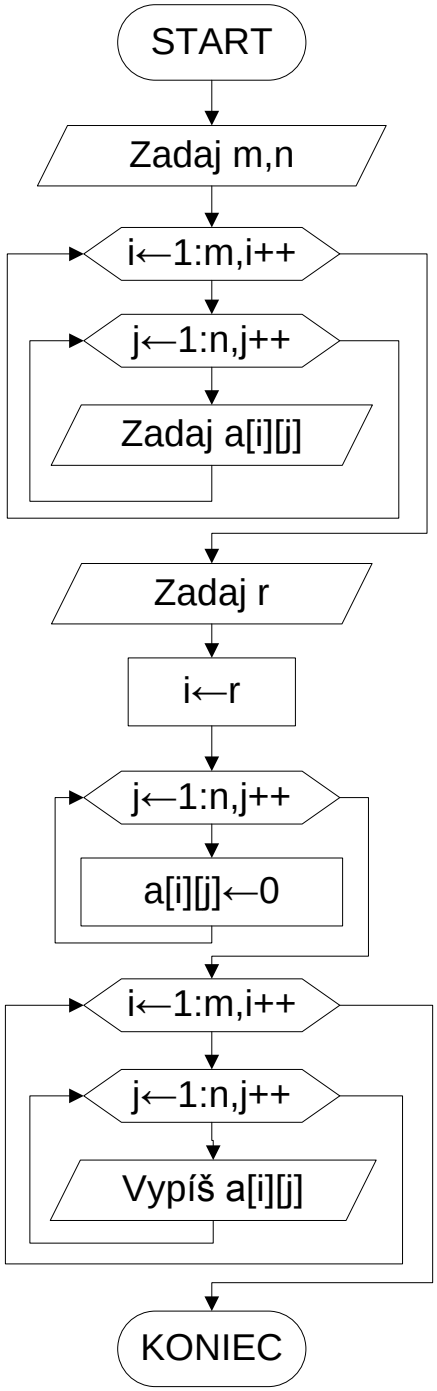
9. Algoritmy s indexovanou premennou – dvojrozmerné pole

1. Vytvorte algoritmus, ktorý načíta prvky do matice a následne prvky v jednom z riadkov zadanom používateľom zmení hodnotu všetkých prvkov na 0.

   Vstupné premenné: m, n, a[i][j], r
   Pomocné premenné: i, j
   Výstupná premenná: a[i][j]

   Analýza riešenia:
   Po zadaní počtu riadkov a stĺpcov použitím dvoch cyklov FOR s indexovaním i pre riadky a j pre stĺpce zadáme prvky matice. Následne zadáme číslo riadku (od 1 do m), v ktorom chceme všetky prvky zmeniť na nuly. Zadané číslo riadku načítame do premennej r a pre zmenu prvkov riadku nepoužijeme 2 cykly FOR ako pri načítavaní, pretože nemeníme hodnoty prvkov vo všetkých riadkoch. Preto premennej i priradíme hodnotu premennej r a cyklus FOR použijeme len pre zmenu indexov stĺpcov, v ktorých má byť zmena vykonaná, teda budeme meniť hodnoty prvkov: a[r][1], a[r][2],..., a[r][n]. Následne vypíšeme všetky prvky matice po zmene zadaného riadku.

   Slovný popis algoritmu:
   1. krok: zadaj počet riadkov m, počet stĺpcov n
   2. krok: ak i=1 až m, pokračuj krokom 3, inak prejdi na krok 5
   3. krok: ak j=1 až n, pokračuj krokom 4, inak sa vráť na krok 2
   4. krok: zadaj a[i][j], vráť sa na krok 3
   5. krok: zadaj číslo riadku r, ktorý chceš meniť
   6. krok: premennej i priraď hodnotu r
   7. krok: ak j=1 až n, pokračuj krokom 8, inak prejdi na krok 9
   8. krok: premennej a[i][j] priraď hodnotu 0, vráť sa na krok 7
   9. krok: ak i=1 až m, pokračuj krokom 10, ukonči program
   10. krok: ak j=1 až n, pokračuj krokom 11, inak sa vráť na krok 9
   11. krok: vypíš a[i][j], vráť sa na krok 10



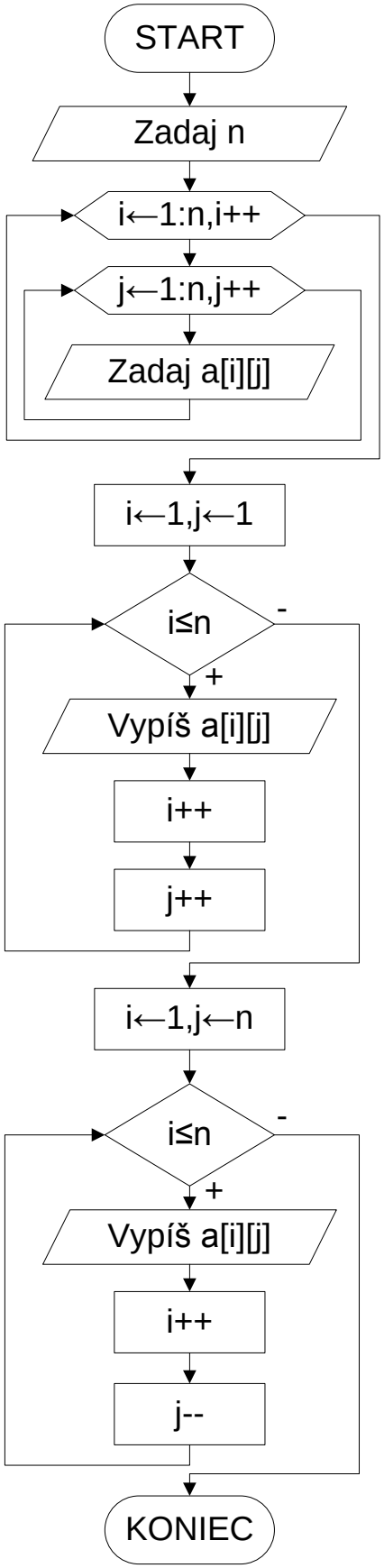

```
int main()
{
 int m,n,r,i,j,a[i][j];

 printf("Zmena prvkov zadaneho riadku na nuly\n\n");

 printf("Zadaj pocet riadkov m: ");
 scanf("%d",&m);

 printf("Zadaj pocet stlpcov n: ");
 scanf("%d",&n);

 int a[m][n];

 printf("\n");

 for(i=1;i<=m;i++){
        for(j=1;j<=n;j++){
                printf("Zadaj cislo a[%d][%d]: ",i,j);
                scanf("%d",&a[i][j]);
                }
        }

 printf("\nZadaj riadok, ktory chces zmenit na 0: ");
 scanf("%d",&r);
 i=r;
 for(j=1;j<=n;j++){
        a[i][j]=0;
        }

 for(i=1;i<=m;i++){
        printf("\n");
        for(j=1;j<=n;j++){
                printf("%d  ",a[i][j]);
                }
        }
 printf("\n\n");
 system("PAUSE");
 return 0;
}
```

2. Vytvorte algoritmus, ktorý vypíše hlavnú a vedľajšiu diagonálu zadanej štvorcovej matice, pričom používateľ na vstupe nezadá počet riadkov aj stĺpcov, ale len jedno číslo, ktoré bude symbolizovať počet riadkov, ako aj stĺpcov.

Vstupné premenné: n, a[i][j]
Pomocné premenné: i, j
Výstupná premenná: a[i][j]

Analýza riešenia:
Zadáme jednotlivé prvky matice. Hlavná diagonála začína vždy prvkom a[1][1], pokračuje a[2][2], ..., a končí prvkom a[n][n]. To znamená, že hodnota indexu j je rovnaká ako hodnota indexu i. Preto najprv nastavíme hodnotu oboch premenných na 1 a cyklom WHILE overíme, či je hodnota ľubovoľného z nich menšia alebo rovná hodnote n. Ak podmienka v cykle WHILE je splnená, vypíšeme prvok a[i][j] čo je v prvom prípade a[1][1], následne zvýšime hodnotu oboch indexových premenných o 1, čím a opakujeme cyklus až kým program nevypíše hodnotu prvku a[n][n]. Pri vypisovaní vedľajšej diagonály je postup takmer rovnaký až nato, že prvý prvok je a[1][n] a posledný a[n][1]. Čo znamená, že zatiaľ čo hodnotu premennej i pri každom behu cyklu zvyšujeme o 1, premennú j naopak o 1 znižujeme, pričom sme pred zbehnutím cyklu nastavili hodnotu hodnoty premenných i a j nasledovne: i=1, j=n.

Slovný popis algoritmu:
1. krok: zadaj počet riadkov a stĺpcov n
2. krok: ak i=1 až n, pokračuj krokom 3, inak prejdi na krok 5
3. krok: ak j=1 až n, pokračuj krokom 4, inak sa vráť na krok 2
4. krok: zadaj a[i][j], vráť sa na krok 3
5. krok: nastav i=1, j=1
6. krok: kým i≤n, opakuj kroky 7-9, inak prejdi na krok 10
7. krok: vypíš a[i][j]
8. krok: i zvýš o 1
9. krok: j zvýš o 1, vráť sa na krok 6
10. krok: nastav i=1, j=n
11. krok: kým i≤n, opakuj kroky 12-14, inak ukonči program
12. krok: vypíš a[i][j]
13. krok: i zvýš o 1
14. krok: j zníž o 1, vráť sa na krok 11




```
int main()
{
 int n,i,j,a[i][j];

 printf("Zadaj pocet riadkov a stlpcov n: ");
 scanf("%d",&n);

 printf("\n");

 for(i=1;i<=n;i++){
        for(j=1;j<=n;j++){
                printf("Zadaj cislo a[%d][%d]: ",i,j);
                scanf("%d",&a[i][j]);
                }
        }

 i=1;
 j=1;
 printf("\nHlavna diagonala obsahuje prvky: ");
 while(i<=n){
        printf("%d ",a[i][j]);
        i++;
        j++;
        }

 i=1;
 j=n;
 printf("\nVedlajsia diagonala obsahuje prvky: ");
 while(i<=n){
        printf("%d ",a[i][j]);
        i++;
        j--;
        }

 printf("\n\n");
 system("PAUSE");
 return 0;
}
```

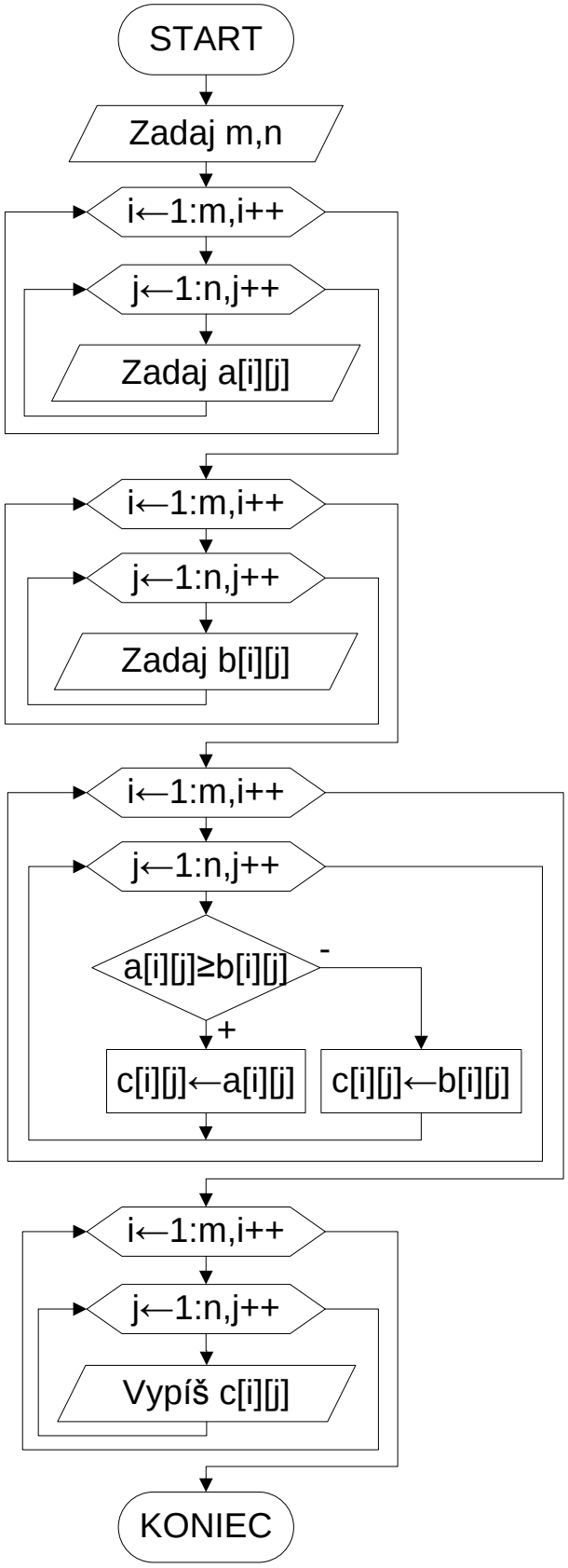
10. Algoritmy s indexovanou premennou – dvojrozmerné pole

1 Vytvorte algoritmus, ktorý zo zadaných 2 matíc vytvorí tretiu a to tak, že v dvoch zadaných maticiach porovná prvky na rovnakých miestach a do tretej matice zapíše väčší z nich. Užívateľ na vstupe zadá počet riadkov m a stĺpcov n, ktoré budú rovnaké pre obidve zadávané matice.

Vstupné premenné: m, n, a[i][j], b[i][j]
Pomocné premenné: i, j
Výstupná premenná: c[i][j]

Analýza riešenia:
Po zadaní matíc A a B, nasledujú 2 cykly FOR – jeden pre index riadkov i a druhý pre index stĺpcov j. Tie zabezpečujú, aby bola porovnaná veľkosť všetkých dvojíc zodpovedajúcich prvkov z matice A a B, teda či platí že a[i][j]≥b[i][j]. Ak áno, do premennej c[i][j] vložíme hodnotu premennej a[i][j]. V opačnom prípade do premennej c[i][j] vložíme hodnotu premennej b[i][j]. Nakoniec vypíšeme všetky prvky matice C.

Slovný popis algoritmu:
1. krok: zadaj počet riadkov m a počet stĺpcov n
2. krok: ak i=1 až m, pokračuj krokom 3, inak prejdi na krok 5
3. krok: ak j=1 až n, pokračuj krokom 4, inak sa vráť na krok 2
4. krok: zadaj a[i][j], vráť sa na krok 3
5. krok: ak i=1 až m, pokračuj krokom 6, inak prejdi na krok 8
6. krok: ak j=1 až n, pokračuj krokom 7, inak sa vráť na krok 5
7. krok: zadaj b[i][j], vráť sa na krok 6
8. krok: ak i=1 až m, pokračuj krokom 9, inak prejdi na krok 13
9. krok: ak j=1 až n, pokračuj krokom 10, inak sa vráť na krok 8
10. krok: ak platí: a[i][j]≥b[i][j], prejdi na krok 11, inak prejdi na krok 12
11. krok: priraď c[i][j]=a[i][j], vráť sa na krok 9
12. krok: priraď c[i][j]=b[i][j], vráť sa na krok 9
13. krok: ak i=1 až m, pokračuj krokom 14, inak ukonči program
14. krok: ak j=1 až n, pokračuj krokom 15, inak sa vráť na krok 13
15. krok: vypíš c[i][j], vráť sa na krok 14




```
int main()
{
 int m,n,i,j,a[i][j],b[i][j],c[i][j];

 printf("Vypisanie matice s vacsimi prvkami z dvoch
matic\n\n");

 printf("Zadaj pocet riadkov m: ");
 scanf("%d",&m);

 printf("Zadaj pocet stlpcov n: ");
 scanf("%d",&n);

 printf("\n");

 for(i=1;i<=m;i++){
        for(j=1;j<=n;j++){
                printf("Zadaj cislo a[%d][%d]: ",i,j);
                scanf("%d",&a[i][j]);
                }
        }

 for(i=1;i<=m;i++){
        for(j=1;j<=n;j++){
                printf("Zadaj cislo b[%d][%d]: ",i,j);
                scanf("%d",&b[i][j]);
                }
        }

 for(i=1;i<=m;i++){
        for(j=1;j<=n;j++){
                if(a[i][j]>=b[i][j]){
                        c[i][j]=a[i][j];
                        }
                else c[i][j]=b[i][j];
                }
        }

 for(i=1;i<=m;i++){
        printf("\n");
        for(j=1;j<=n;j++){
                printf("%d ",c[i][j]);
                }
        }

 printf("\n\n");
 system("PAUSE");
 return 0;
}
```