Digitální učební materiál

Číslo projektu:	CZ.1.07/1.5.00/34.0548
Název školy:	Gymnázium, Trutnov, Jiráskovo náměstí 325
Název materiálu:	VY_32_INOVACE_141_IVT
Autor:	Ing. Pavel Bezděk
Tematický okruh:	Algoritmy
Datum tvorby:	červenec 2013
Ročník:	4. ročník a oktáva
Anotace:	Algoritmus I. – Definice algoritmu, jeho vlastnosti a vývojové diagramy
Metodický pokyn:	

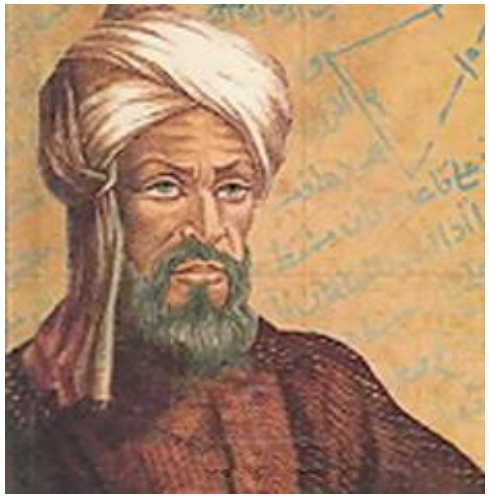
esf evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Autor	**Ing. Pavel Bezděk**		
Vytvořeno dne	**1. 7. 2013**		
Odpilotováno dne	**11. 11. 2013**	**ve třídě**	**8.Y**
Vzdělávací oblast	**Informatika a informační a komunikační technologie**		
Vzdělávací obor	**Informatika a výpočetní technika**		
Tematický okruh	**Algoritmus**		
Téma	**Algoritmus I. – Definice algoritmu, jeho vlastnosti a vývojové diagramy**		
Klíčová slova	**Algoritmus, determinovanost, konečnost**		

Algoritmus

a jeho vlastnosti

Historie algoritmu



Slovo algoritmus pochází ze jména významného perského matematika žijícího v první polovině 9. století (cca 780–840 n. l.), kterým byl Abū ʻAbd Allāh Muhammad ibn Mūsā al-Khwārizmī (أبو عبد الله محمد بن موسى الخوارزمي) (doslova „Otec Abdulláha, Mohameda, syn Mojžíšův, pocházející z města Khwārizm.“ Region Khwārizm se nachází jižně od Aralského moře.

Tento učenec ve svém díle prakticky vytvořil systém arabských číslic a základy algebry (konkrétně metody řešení lineárních a kvadratických rovnic), jejíž název pochází přímo z titulu tohoto díla (Kitūb al-jabr waāl-muqūbala). Jeho jméno bylo do latiny převedeno jako algorismus, a původně znamenalo: „provádění aritmetiky pomocí arabských číslic.“

Postupem času se kvůli neznalosti původu slova jeho podoba měnila, záměnou arabského kořene s kořenem řeckého slova αριθμός (arithmos) se z algorismu stal algoritmus. (Později bylo v některých jazycích včetně češtiny th změněno na t, v katalánštině se vrátilo s.). Toto slovo se používalo jako označení různých matematických postupů.

Slovo algoritmus v dnešním významu se používá až zhruba od 20. století.

Definice algoritmu

Algoritmus bývá často definován jako: „přesný návod či postup, kterým lze vyřešit daný typ úlohy.“ Toto není zcela přesné.

Přesné znění definice algoritmu zní: „Algoritmus je procedura proveditelná Turingovým strojem.“

Turingův stroj (z roku 1936) je teoretický model počítače popsaný matematikem Alanem Turingem. Skládá se z procesorové jednotky, tvořené konečným automatem a programu ve tvaru pravidel přechodové funkce a pravostranně nekonečné pásky pro zápis mezivýsledků a vstupů dat. Přesnější definice algoritmu je pro nás již moc složitá, protože bychom pro její pochopení potřebovali mít znalosti z teoretické informatiky. Pokusíme se vytvořit ne zcela přesnou, ale srozumitelnější definici.

Algoritmus je jasný, přesný a jednoznačný postup, s konečným množstvím kroků, kterým lze vyřešit daný typ úlohy.

Přičemž algoritmus musí mít tyto základní vlastnosti: „Konečnost, korektnost, obecnost, determinovanost, resultativnost, efektivnost a elementárnost.“

Vlastnosti algoritmu

- ***Konečnost* (finitnost)**

Každý algoritmus musí skončit v *konečném* počtu kroků. Tento počet kroků může být libovolně velký (podle rozsahu a hodnot vstupních údajů), ale pro každý jednotlivý vstup musí být konečný.

Některé metody jsou teoreticky konečné a algoritmicky správné, mohou však trvat tak dlouho, že jsou prakticky nepoužitelné. Algoritmus se pak musí převést na jiný typ algoritmu, který dodá výsledek, který sice nebude zcela přesný, ale přesnost pro naše potřeby bude vyhovovat. Výsledek však dodá v podstatně kratším čase. Když bychom potřebovali ještě přesnější výsledek, doba výpočtu se sice prodlouží, ale stále ještě bude v reálném čase.

Algoritmus nazýváme konečný, když skončí v konečném čase pro libovolné vstupní údaje.

- ***Korektnost***

Algoritmus skončí pro libovolná (korektní) data správným výsledkem v konečném množství kroků.

- ***Obecnost (hromadnost, masovost)***

Algoritmus neřeší jeden konkrétní problém (např. „součin 5×11“), ale obecnou třídu obdobných problémů (např. „jak spočítat součin dvou celých čísel“), má širokou množinu možných vstupů.

- ***Vstup (univerzálnost)***

Vstupy mají definované množiny hodnot, kterých mohou nabývat. Algoritmus pracuje s nějakými vstupy, veličinami, které mu jsou předány před započetím jeho provádění nebo v průběhu činnosti.

Jednoduše řečeno, jsou-li vstupem programu reálná čísla, nemohu dát na vstup např. řetězec znaků „AcBaCb“.

- ***Výstup (rezultativnost)***

Algoritmus má alespoň jeden *výstup*, veličinu, která je v požadovaném vztahu k zadaným vstupům, a tím tvoří odpověď na problém, který algoritmus řeší (algoritmus vede od zpracování hodnot k výstupu).

Algoritmus dává pro stejné vstupní údaje vždy stejné výsledky (když skončí). Po konečném množství kroků dostáváme výsledek.

- ***Elementárnost***

Algoritmus se skládá z konečného počtu jednoduchých (elementárních) kroků.

Každý postup může být zapsaný vícerými způsoby. Při jeho navrhování je třeba dbát na to, aby jednotlivé instrukce byly srozumitelné, jednoduché a jednoznačné. Algoritmus je tedy složený z jednoduchých kroků, které jsou srozumitelné pro vykonavatele (počítač, člověk).

- ***Determinovanost***

Každý krok algoritmu musí být *jednoznačně* a *přesně definován*; v každé situaci musí být naprosto zřejmé, co a jak se má provést, jak má provádění algoritmu pokračovat, takže pro stejné vstupy dostaneme pokaždé stejné výsledky.

V každém momentu vykonávání algoritmu je tedy jednoznačně určené, co se má provést a jaká činnost má následovat nebo zda již postup skončil.

Protože běžný jazyk obvykle neposkytuje naprostou přesnost a jednoznačnost vyjadřování, byly *pro zápis algoritmů navrženy programovací jazyky,* ve kterých má každý příkaz jasně definovaný význam.

Vyjádření výpočetní metody v programovacím jazyce se nazývá program.

Nesmíme zapomenout na žádnou možnost, která může nastat při běhu programu, ve výčtu jednotlivých kroků.

- ***Efektivnost***

Algoritmus se uskutečňuje v co nejkratším čase a s využitím co nejmenšího počtu prostředků (paměti). Efektivnost je proto velmi důležitá hlavně při zpracování velkého množství údajů.

Algoritmy musí splňovat různá kritéria, měřená např. počtem kroků potřebných pro běh algoritmu, jednoduchost, efektivitu či eleganci.

Problematikou efektivity algoritmů, tzn. metodami, jak z několika známých algoritmů řešících konkrétní problém vybrat ten nejlepší, se zabývají odvětví informatiky nazývané *algoritmická analýza* (rozložení na kroky) a *teorie složitosti*.

Programovací jazyky

Algoritmus musíme nějak převést do počítače. K tomu používáme tzv. **programovací jazyky**, např. Java, C++, Pascal apod. Algoritmus přepíšeme do programovacího jazyka, je to tedy text napsaný v programovacím jazyce. Když programovací jazyk umíme, textu rozumíme. Ale aby mu rozuměl i počítač, převede tento text **kompilátor** nebo **interpret** do **strojového jazyka**, kterému již rozumí počítač a je schopen provádět jeho instrukce.

Kompilace je **přeložení** tzv. kompilátorem **celého textu najednou do strojového kódu a uložení do spustitelného exe souboru** (přípona souboru je exe). Spuštěním exe souboru spustíme program. Máme exe soubor, nemusíme, chceme-li jej znovu spustit, překládat kompilátorem, stačí jen pustit již přeložený exe soubor.

Interpretace je odlišná, interpret **čte text a přímo provádí jeho příkazy.** Ale při každém spuštění programu musíme znovu provést interpretaci. Při kompilaci přeložíme jen jednou a při dalším spuštění programu, spouštíme program exe souborem. Některé programovací jazyky mají kompilátor (Java, Pascal), jiné interpret (Perl, Python, unixový shell, shell: interpret příkazů) a některé obojí (Python).

Poznámka:

Interpretované jazyky byly zpočátku kompilovány po jednotlivých řádcích; pokud byl určitý řádek uvnitř cyklu, byl kompilován při každém průchodu cyklem. V současnosti většina interpretovaných jazyků používá mezikód a tak kombinuje kompilační a interpretační přístup. Kompilátor pak vytváří určitou formu mezikódu, například bytový kód (bytecode) nebo zřetězený kód, a ten je pak interpretován interpretem mezikódu. Mezikód může být překládán jen jednou (např. u Javy), před každým spuštěním (např. u Perlu nebo Ruby) nebo pokaždé, když se zjistí, že byl změněn zdrojový kód (např. u Pythonu).

Vývojové diagramy

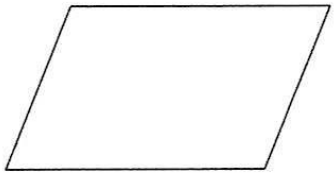
Chceme-li, aby text programu byl lépe srozumitelný pro každého, vypracováváme ještě před přepisem algoritmu do programovacího jazyka vývojový diagram.

Nám to pomůže snadněji převést algoritmus do programovacího jazyka a každý kdo nepsal daný program, ho snadněji pochopí z vývojového diagramu, a to i bez znalosti daného programovací jazyka, pokud rozumí symbolice značek používaných ve vývojových diagramech.

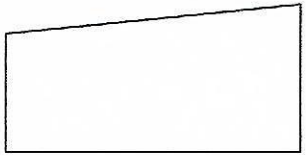
Vývojové diagramy znázorňují průběh či stavbu programu, proto se používají jako část dokumentace programu (projektu). Většina projektů začíná tvorbou takového vývojového diagramu.

Vývojový diagram je grafické znázornění algoritmu.

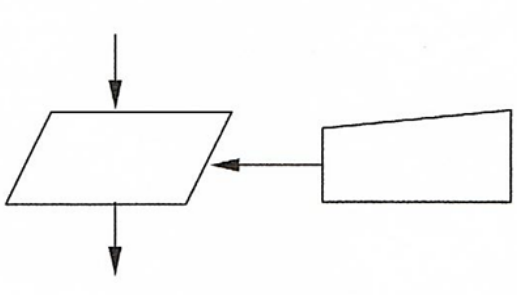
Symboly používané ve vývojovém diagramu



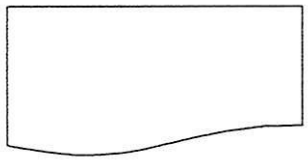
Data vstupně - výstupní operace s daty



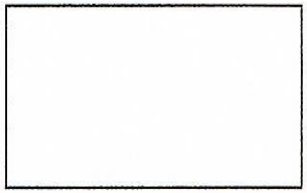
Ruční vstup (klávesnice, apod.)



Ruční vstup

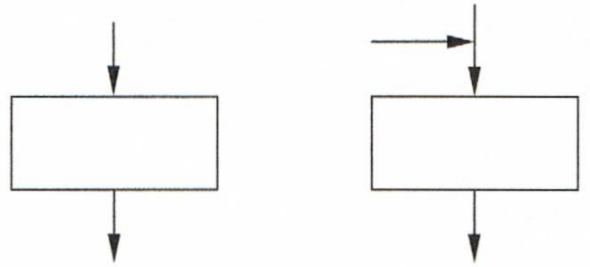


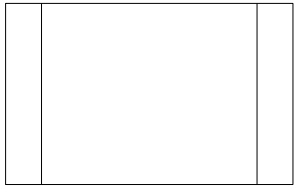
Dokument (**tištěný výstup**)



Zpracování

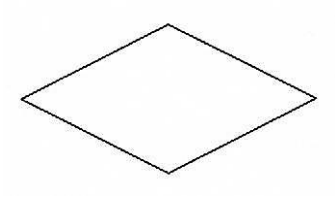
Zpracování nebo provedení definované operace.





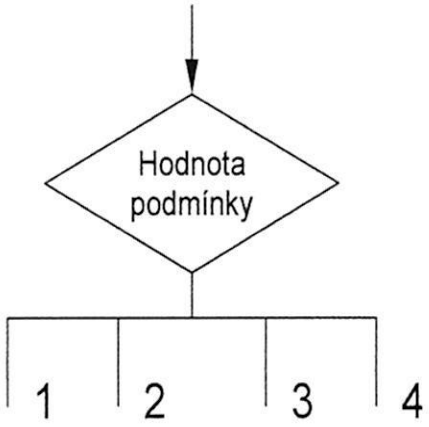
Předdefinované zpracování (podprogram)

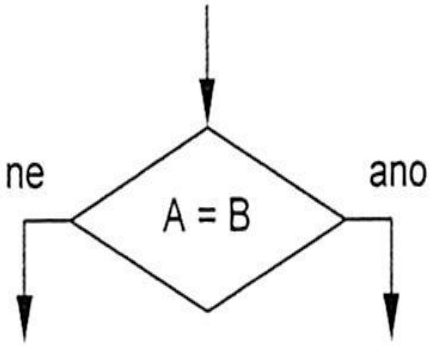
např. procedura, funkce

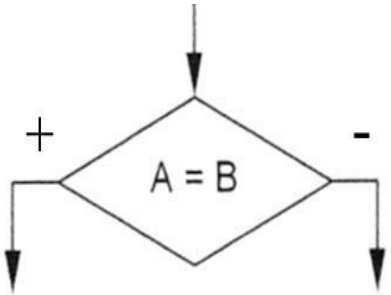


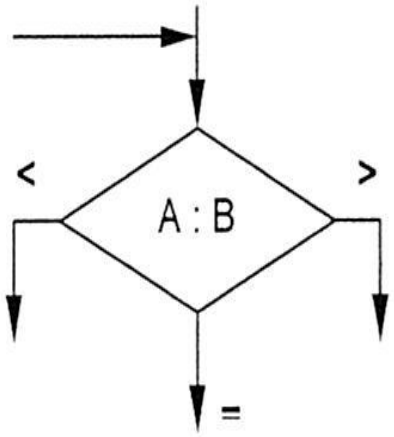
Rozhodování Alternativní výstupy, z nichž pouze jeden je aktivován po vyhodnocení podmínek definovaných uvnitř symbolu.

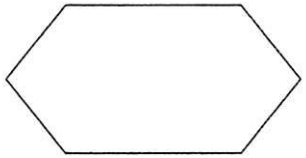
Použití „Rozhodování“







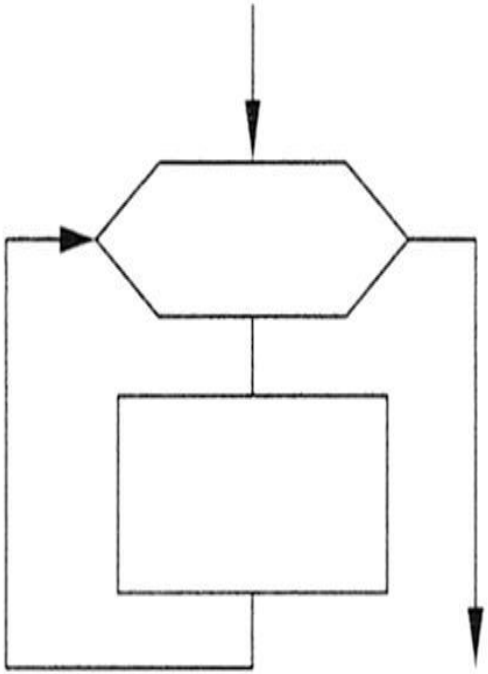




Příprava

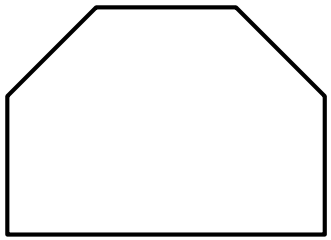
Změna činnosti,
která mění vlastní postup
následné činnosti

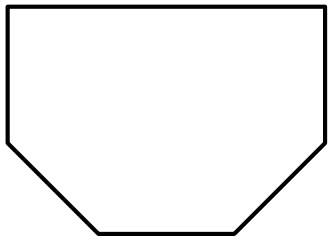
např. For cyklus

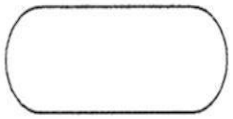


Mez cyklu např. **While cyklus** a **Repeat until cyklus**

Začátek a konec cyklu.





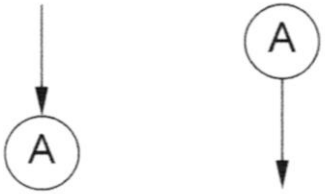


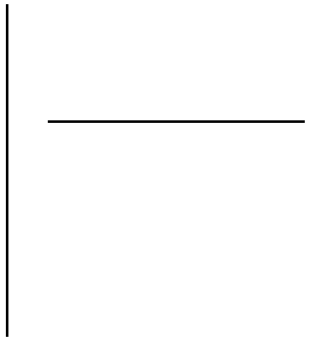
Mezní značka

Začátek nebo konec programu.

Spojka

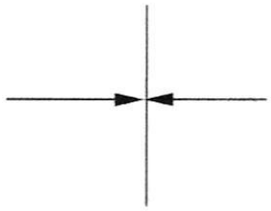
Výstup do jiné části téhož vývojového diagramu nebo vstup z ní na jinou část.





Spojnice

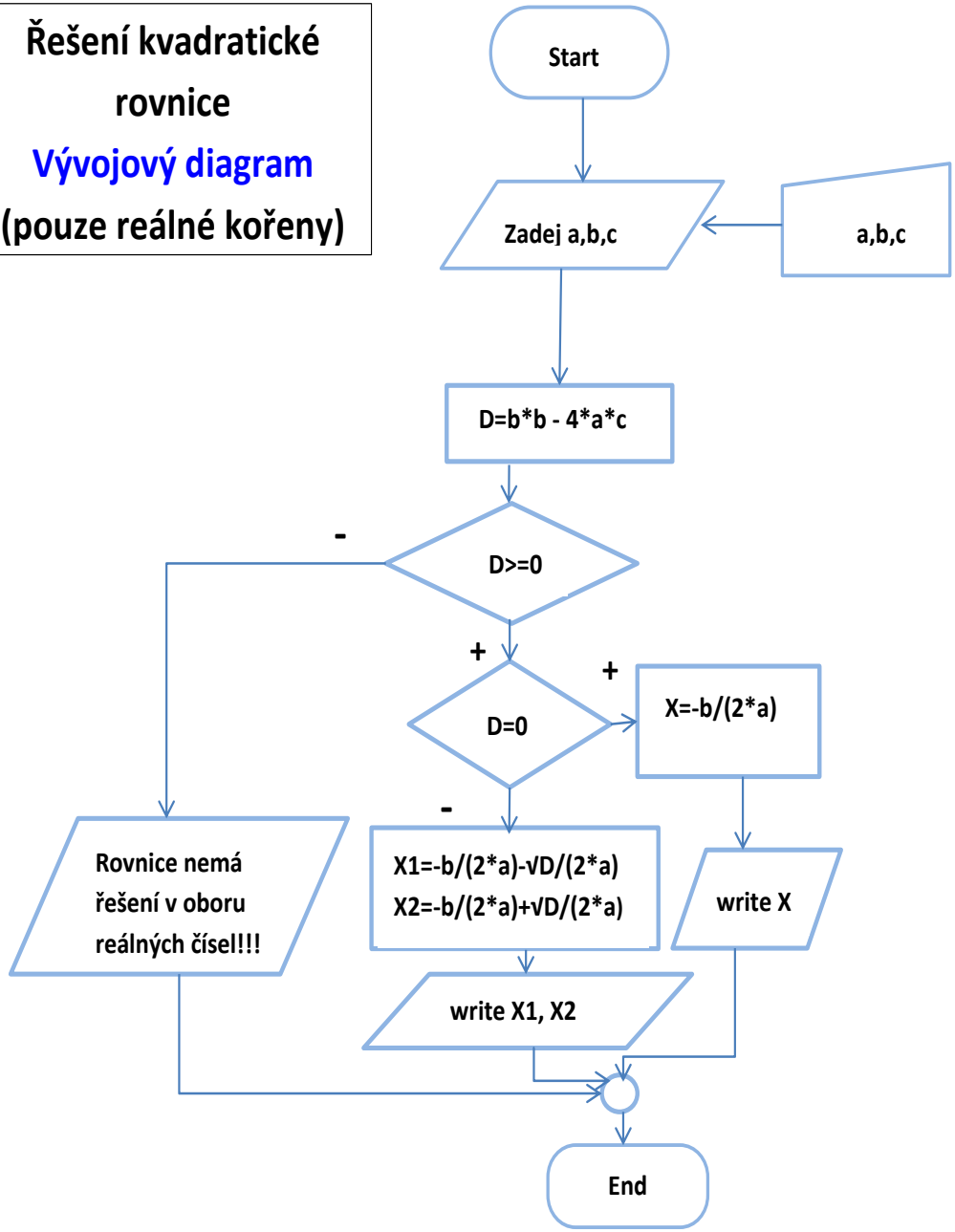
Svislé nebo vodorovné čáry.
Spojení jednotlivých symbolů ve vývojovém diagramu.
Standardní směr toku informací je shora dolů a zleva doprava.
Pro zvýšení jasnosti by měl být u připojovaných spojnic uveden směr toku plnou nebo otevřenou šipkou.
Změny směru v bodech křížením nejsou možné.



Řešení kvadratické rovnice

Vývojový diagram

(pouze reálné kořeny)



Program kvadratická rovnice
Pascal
(pouze reálné kořeny)

```
Program KvadratickaRovnice2;
uses Crt;
var   a,b,c,x1,x2,x,d: real;
 begin
 writeln; writeln; writeln; writeln;
 writeln('Vypocet korenu kvadraticke rovnice:');    writeln('a*x^2 + b*x + c = 0');
 writeln('Po zadani hodnoty a nebo b nebo c, stiskni enter!');    writeln;
 write('Zadej a: ');   readln(a);   write('Zadej b: ');   readln(b);   write('Zadej c: ');   readln(c);
 d:=b*b-4*a*c;
 if d>=0 then begin
              writeln('Rovnice ma reseni v oboru realnych cisel.');   writeln;
              if d>0 then
                        begin
                   x1:=(-b-sqrt(d))/(2*a);    x2:=(-b+sqrt(d))/(2*a);
                   writeln('x1= ',x1:7:3,'    x2= ',x2:7:3);
                       end            {Počet cislic zobrazovanych před (:7) a za (:3) desetinou carkou}
                    else   begin
                  x:=-b/(2*a);   writeln('Rovnice ma 1 dvojnasobny koren:');   writeln ('x = ',x:7:3);
                             end;
                  end
            else   writeln('Rovnice nema reseni v oboru realnych cisel.');
 writeln;  writeln('Po precteni vysledku stiskni enter!');
 repeat until keypressed;
 end.
```

Test

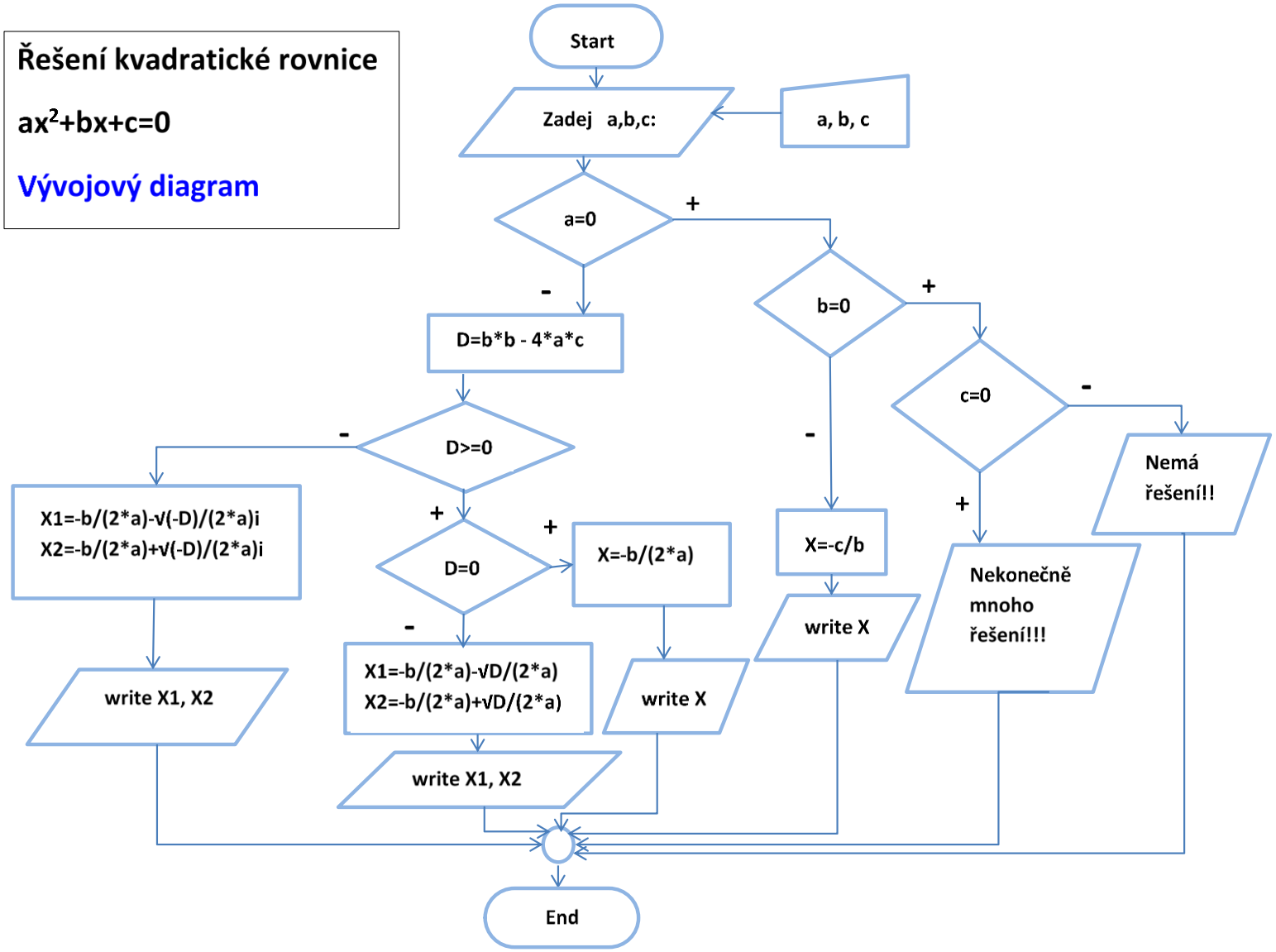
Nakreslete vývojový diagram a pak vypracujte i program pro řešení kvadratické rovnice v oboru reálných i komplexních čísel.

Doplňte i případ, kdy rovnice není kvadratická, ale je pouze lineární.

Řešení kvadratické rovnice

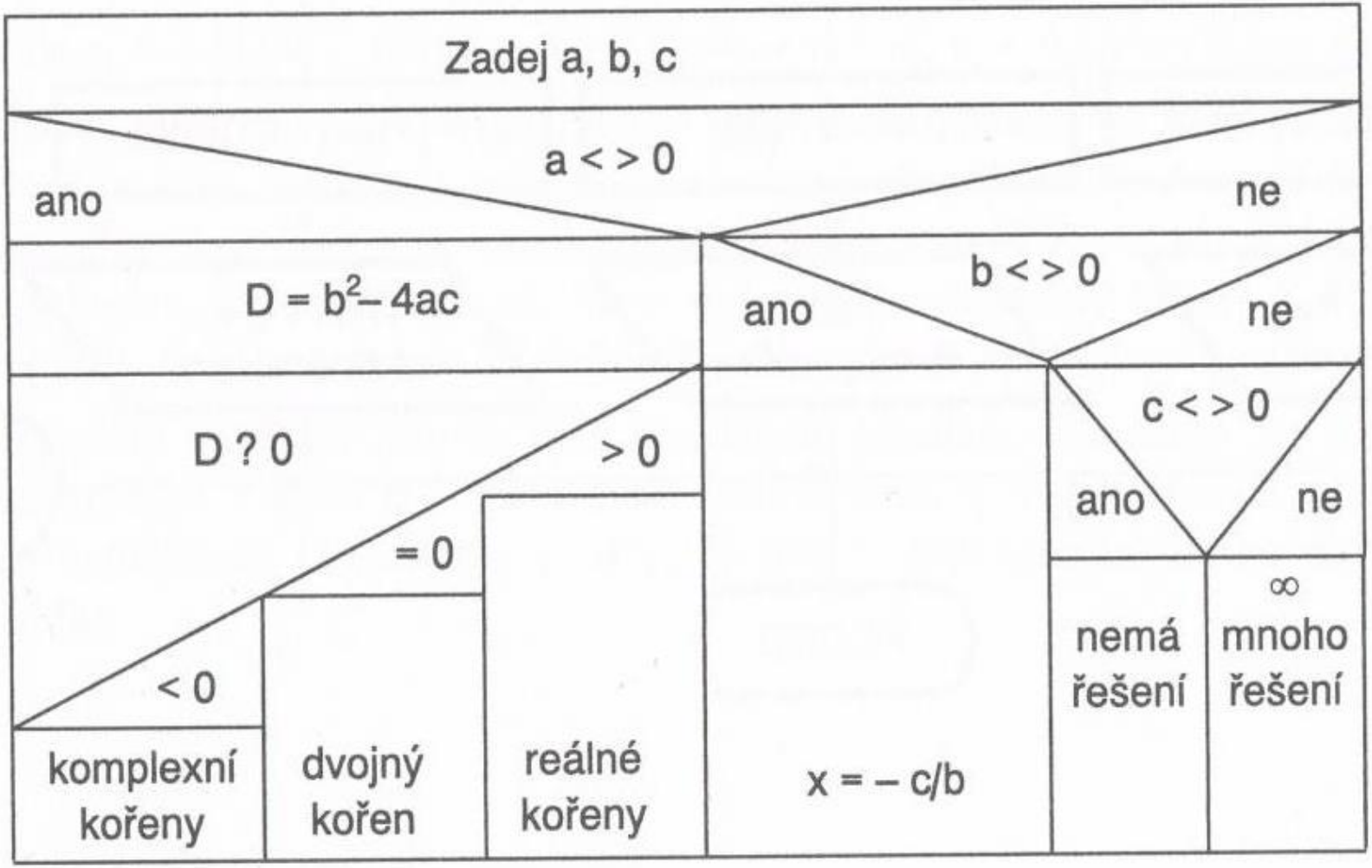
$ax^2+bx+c=0$

Vývojový diagram



Strukturogram

pro kvadratickou rovnici



Program kvadratická rovnice
Pascal

```
Program KvadratickaRovnice;
uses Crt;
var   a,b,c,x1,x2,x,x1r,x2r,x1i,x2i,d: real;
 begin
 writeln; writeln; writeln; writeln;  writeln('Vypocet korenu kvadraticke rovnice:');
 writeln('a*x^2 + b*x + c = 0');
 writeln('Po za dani hodnoty a nebo b nebo c, stiskni enter!');  writeln;
 write('Zadej a: '); readln(a); write('Zadej b: '); readln(b); write('Zadej c: '); readln(c);
 d:=b*b-4*a*c;
 if a<>0 then
          begin
          writeln(' Rovnice je kvadraticka:');
          if d>=0 then
                    begin
                    writeln('Rovnice ma reseni v oboru realnych cisel.'); writeln;
                    if d>0 then
                         begin
                         x1:=(-b-sqrt(d))/(2*a);  x2:=(-b+sqrt(d))/(2*a);
                         writeln('x1= ',x1:7:3,'    x2= ',x2:7:3);
                         end
                           else  begin x:=-b/(2*a);
                              writeln('Rovnice ma 1 dvojnasobny koren:');   writeln ('x = ',x:7:3);
                                 end;
                  end
```

```
                else begin
                writeln('Rovnice ma reseni v oboru komplexnich cisel:');
                x1r:=(-b)/(2*a);              x2r:=(-b)/(2*a);
                x1i:=(-sqrt(-d))/(2*a);     x2i:=(+sqrt(-d))/(2*a);
                writeln('x1= ',x1r:7:3,', ',x1i:7:3,'i');  writeln('x2= ',x2r:7:3,', ',x2i:7:3,'i');
                      end;
        end
        else
         begin writeln('Rovnice je linearni:');
                if b<>0 then begin x:=-c/b;  writeln('x = ',x:7:3);   end
                        else if c<>0 then writeln('Linearni rovnice nema reseni!')
                                     else writeln('Linearni rovnice ma nekonecne mnoho reseni!');
         end;
writeln;     writeln('Po precteni vysledku stiskni enter!');
repeat until keypressed;
end.
```

Program kvadratická rovnice
C++

```
//  KVADRATICKA ROVNICE
#include <iostream>                          //hlavickovy soubor vstupy a vystupy
#include <math.h>                            // hlavickovy soubor matematicky
using namespace std;
float a,b,c,D,x,x1,x2,x1r,x2r,x1i,x2i;       // deklarace globalnich promnenych typu float
void dvoj();                                 //  Prototyp funkce dvoj()
void real();                                 //  Prototyp funkce real()
void komp();                                 //  Prototyp funkce komp()

int main()                                   // hlavni cast programu
{                                            // pocatek hlavniho bloku
   cout<<("\n");
   cout<<(" KVADRATICKA ROVNICE ")<<endl<<endl;
   cout<<("zadejte kvadraticky soucinitel a= ");          //  tisk na obrazovku
   cin>>a;                                   //  zadani kvadratickeho koef.  "a“  z klavesnice
   cout<<("\n");
   cout<<("zadejte linearni soucinitel  b= ");            //  tisk na obrazovku
   cin>>b;                                   // zadani linearniho koef.  "b"  z klavesnice
   cout<<("\n");
   cout<<("zadejte absolutni soucinitel  c= ");           // tisk na obrazovku
   cin>>c;                                   // zadani absolutniho koef.  "c“  z klavesnice
    cout<<("\n");
```

```
if (a!=0)
   {   D=b*b-4*a*c;                                          // pocatek bloku if (a!=0)
      cout<<("Diskriminant D = ")<<D<<endl;
   if (D==0)
         { dvoj();
         }
     else if (D>0)
                  { real();
                  }
                    else komp();
   }                                                         //  konec bloku  if (a!=0)
                else
                {                                                 // zacatek bloku else  a=0
                      if (b!=0)
                                              {                          // pocatek bloku if (b!=0)
                                              x = -c/b;
                                              cout<<(" Reseni linearni rovnice je x = ")<<x;
                                              }                          //  konec bloku if (b!=0)
                                              else
                                                  if (c!=0)
                                                    cout<<("Linearni rovnice nema reseni!!!");
                                                  else
                                                    cout<<("Linearni rovnice ma nekonecne mnoho reseni!!!");
                }                                                       // konec bloku   else  a=0
return 0;
}                                        // konec hlavniho bloku
```

```
void dvoj()                                          // definice funkce dvoj() - hlavicka funkce dvoj()
 {                                                     // Pocatek bloku tela funkce dvoj()
 x=-b/(2*a);
 cout<<("Dvojnasobny koren x = ")<<x <<endl;
 }                                                   // Konec bloku tela   funkce dvoj()

void real()                                         // definice funkce real() - hlavicka funkce real()
    {                                                  // Pocatek bloku tela funkce real()
    x1=(-b+sqrt(D))/(2*a);
    cout<<(" Koren x1 = ")<< x1<<endl;
    x2=(-b-sqrt(D))/(2*a);
    cout<<(" Koren x2 = ") <<x2;
    }                                                  // Konec bloku tela   funkce real()

void komp()                                         // definice funkce komp() - hlavicka funkce komp()
        {                                              // Pocatek bloku tela funkce komp()
        x1r=-b/(2*a);
        x1i=sqrt(-D)/(2*a);
        cout<<("Komplexni koren x1 = ")<<"( "<<x1r<< ",  "<< x1i<<" i )" <<endl;
                                          cout<<("Komplexni koren x2 = ")<<"( "<<x1r<< ",  "<<-x1i<<" i )" ;
                                          cout<<("\nDruhy komplexni koren je k vypoctenemu
sdruzeny.")<<endl;
         }                                          // Konec bloku tela funkce komp()
```

Použité zdroje

Algoritmus:

Algoritmus. In: *Wikipedia: Otevřená encyklopedie* [online]. [cit. 2013-07-01]. Dostupné podle licence Creative Commons z www: http://cs.wikipedia.org/wiki/Algoritmus

Vývojové diagramy:

ČSN ISO 5807. *Zpracování informací: Dokumentační symboly a konvence pro vývojové diagramy toku dat, programu a systému, síťové diagramy programu a diagramy zdrojů systému*. Český normalizační institut, 1995.