Algoritmy a jejich znázorňování.

Znázornění algoritmů. Formální jazky. Programovací jazyky a jejich dělení.

Tomáš Bayer | bayertom@natur.cuni.cz

Katedra aplikované geoinformatiky a kartografie. Přírodovědecká fakulta UK.

Obsah přednášky

1. Znázorňování algoritmů:

Algoritmus lze znázorňovat mnoha způsoby.

Nejčastěji jsou používáno:

- Grafické vyjádření algoritmu.
- Textové vyjádření algoritmu.

Grafické vyjádření algoritmu

Algoritmus je popsán formalizovanou soustavou grafických symbolů.

Používány vývojové diagramy nebo strukturogramy.

Výhody: přehlednost, názornost, znázornění struktury problému, poskytuje informace o postupu jeho řešení.

Nevýhody: náročnost konstrukce grafických symbolů a jejich vzájemných vztahů, obtížná možnost dodatečných úprav postupu řešení vedoucí často k “překreslení” celého postupu, technika není vhodná pro rozsáhlé a složité problémy,

2. Vývojové diagramy

Jeden z nejčastěji používaných prostředků pro znázorňování algoritmů.

Tvořeny značkami ve formě uzavřených rovinných obrazců, do kterých jsou vepisovány slovní či symbolickou formou jednotlivé operace.

Tvary a velikosti značek jsou dány normami.

Značky jsou spojeny přímými nebo lomenými spojnicemi čarami a znázorňují tak posloupnosti jednotlivých kroků. Čáry mohou být orientované zavedením šipek, neměly by se křížit.

Pokud již ke křížení dojde, měly by být čáry zvýrazněny tak, aby bylo jednoznačně patrné, odkud a kam směřují.

Vývojový diagram čteme ve směru shora dolů.

Výhody: názornost, přehlednost

Nevýhody: pracnost a složitost konstrukce, složité diagramy se

3. Komponenty vývojového diagramu

Start/konec algoritmu:

Počáteční a koncový krok algoritmu.

Vstup/Výstup:

Načtení dat potřebných pro běh programu, uložení dat.

Předzpracování dat:

Incializace proměnných.

Zpracování dat:

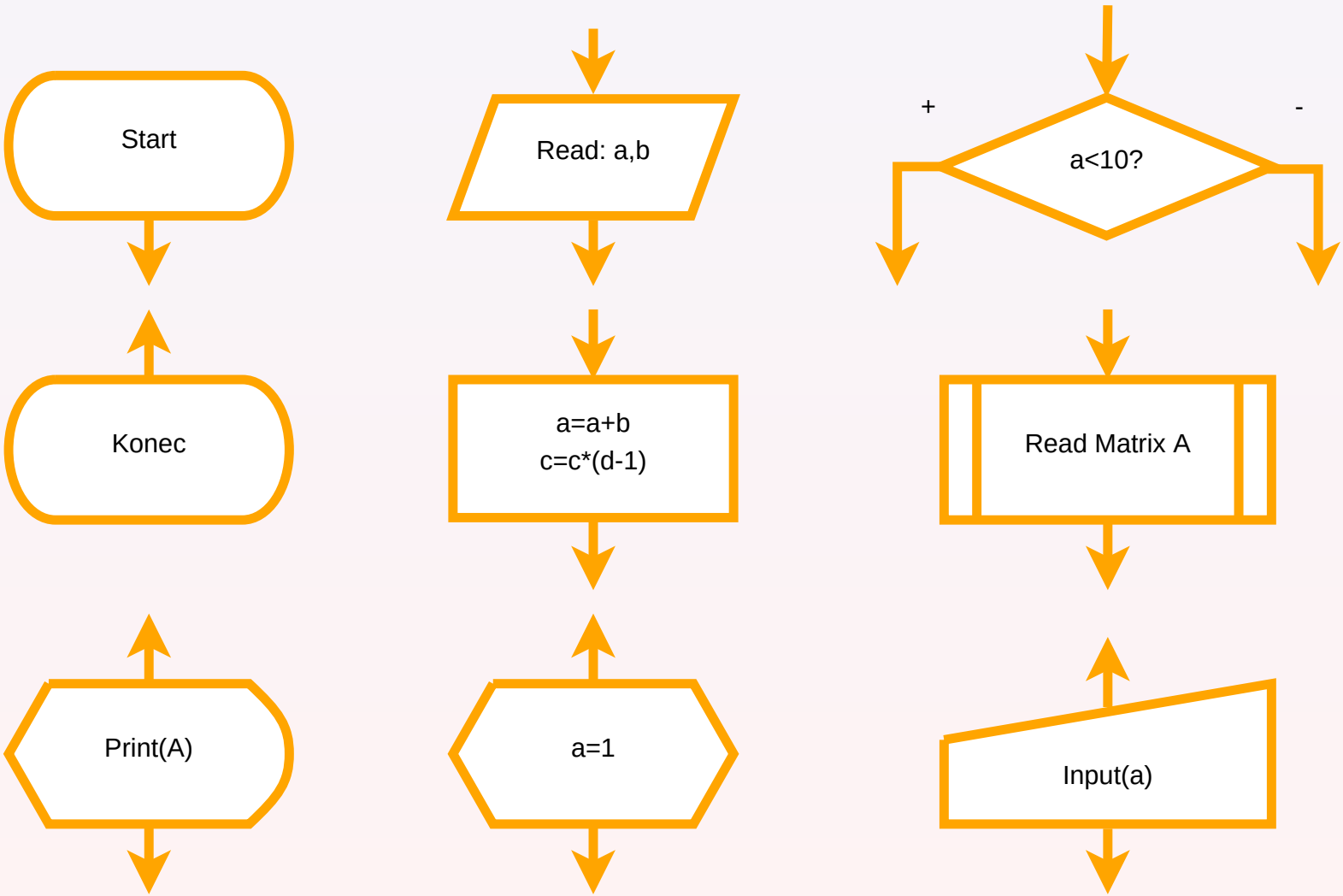
Dochází k transformaci dat. Jeden vstup a jeden výstup, nesmí dojít k rozvětvení programu.

Rozhodovací blok:

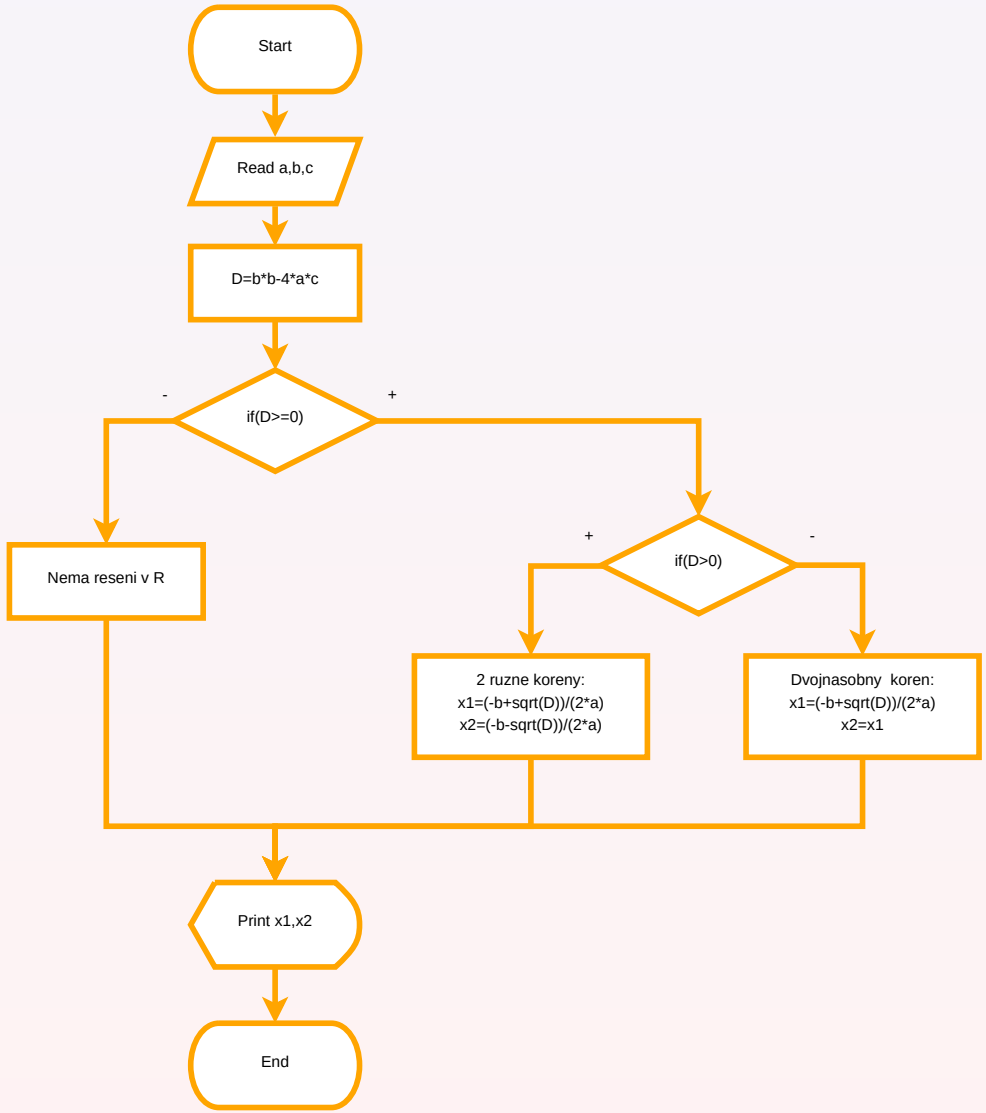
Větvení na základě vstupní podmínky. Je -li splněna, pokračuje se větví (+), v opačném případě větví (-).

Podprogram:

4. Ukázky vývojových diagramů



5. Vývojový diagram řešení kvadratické rovnice



6. Strukturogram

Úspornější znázornění algoritmu, kombinace grafického a textového popisu

Tvořen obdélníkovou tabulkou, do řádků zapisujeme postup kroků symbolickou či slovní formou v pořadí, v jakém budou prováděny.

Záhlaví tabulky obsahuje název algoritmu nebo dílčího kroku.

Výhody:

Přehlednější způsob znázornění.

Lze ho aplikovat i na složitější problémy.

Jednoznačný a snadný přepis do formálního jazyka.

Nevýhody:

Pracnost konstrukce, složité strukturogramy se nevejdou na jednu stránku.

Malé možnosti pozdějších úprav.

<table>
<tr><td colspan="4">Řešení kvadratické rovnice</td></tr>
<tr><td colspan="4">Čti a, b, c</td></tr>
<tr><td colspan="4">Spočti diskriminant D=b*b -4*a*c</td></tr>
<tr><td colspan="4">Je D>0?</td></tr>
<tr><td rowspan="3"></td><td>Ano</td><td colspan="2">Ne</td></tr>
<tr><td>x1=(-b+sqrt(D))/(2*a)</td><td colspan="2">Je D=0 ?</td></tr>
<tr><td>x2=(-b-sqrt(D))/(2*a)</td><td rowspan="2"></td><td>Ano</td><td>Ne</td></tr>
<tr><td>Tisk 2 kořeny:</td><td>x1=(-b+sqrt(D))/(2*a)</td><td>Tisk:</td></tr>
</table>

7. UML

= Unified Modeling Language

Jazyk pro vizuální návrh a popis datových struktur, programů či systémů.

Umožňuje vytvářet různé typy diagramů: diagram komponent programu, diagram objektů, diagram tříd...
Z těchto diagramů lze zpětně vygenerovat kostru zdrojového kódu.

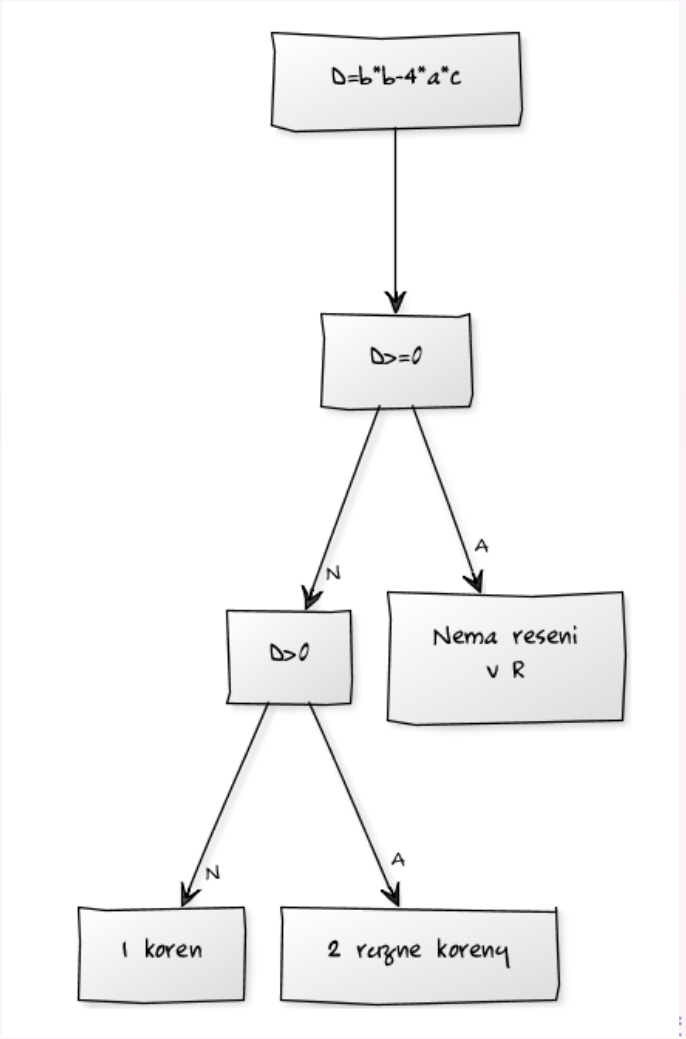
Výhody UML:

- Open-source.
- Možnost vyjádření grafického návrhu formalizovaným jazykem.
- Propojení vizuálního a textového popisu.
- Univerzálně použitelný zejména pro modelování objektově orientovaných systémů.

8. Ukázka UML

Zdrojový kód:

```
# Test UML Diagram

[D=b*b-4*a*c]->[D>=0]

[D>=0]-N>[D>0]

[D>=0]-A>[Nema reseni v R]

[D>0]-A>[2 ruzne koreny]

[D>0]-N>[1 koren]
```



9. Textové vyjádření algoritmu

Tento způsob vyjádření algoritmů se v současné době používá nejčastěji.

Zápis algoritmu prostřednictvím formalizovaného jazyka.

Využíván pesudokód nebo PDL (Program Description Language).

Výhody:

Přehlednost zápisu.

Jednoznačnost jednotlivých kroků.

Snadný přepis do programovacího jazyku (nejblíže ke skutečnému programovacímu jazyku).

Možnost pozdější modifikace postupu řešení.

10. Textové vyjádření řešení kvadratické rovnice

Algoritmus 3: Kvadratická rovnice (a,b,c)

```
1: Read (a,b,c)
2: D=b*b-4*a*c
2: If D > 0:
3:         x1=(-b+sqrt(D))/(2*a)
4:         x2=(-b- sqrt(D))/(2*a)
5: Else if D = 0
6:         x1=(-b+sqrt(D))/(2*a)
7:         x2=x1
8: else
9:         x1=∅, x2=x1
10: Print (x1,x2)
```

11. Jazyky a jejich vlastnosti (úvod)

Jazyky a jejich dělení:

- **Přirozené jazyky**
  Komunikační prostředek mezi lidmi.
  Skládání symbolů do vyšších celků není řízeno tak striktnějšími pravidly jako u formálních jazyků.
  Nevýhodou je významová nejednoznačnost některých slov (synonyma, homonyma), která omezuje jejich použití při popisu řešení problémů.
- **Formální jazyky**
  Vznikají umělou cestou.
  Skládání symbolů do vyšších celků je řízeno striktnějšími pravidly.
  Zamezení významové nejednoznačnosti.
  Zástupci: programovací jazyky, matematická symbolika, jazyk mapy...

12. Slovo, abeceda, jazyk

Pro popis algoritmů používány formální jazyky.

Formální abeceda $\sum$:

Konečná množina elementárních symbolů (písmen, znaků) přesného významu, které lze spojovat do složitějších útvarů.

Binární abeceda $\sum = \{0, 1\}$, hexadecimální abeceda $\sum = \{0, .., 9, A, ..., F\}$, textová abeceda: ASCII, UTF.

Formální slovo w nad abecedou $\sum$:

Konečný řetězec symbolů nad danou abecedou představovan libovolnou posloupností těchto symbolů.

Má jednoznačný význam.

Příklady slov: $w = 010101$, prázdné slovo ε.

Formální jazyk:

Množina všech formálních slov $\{w\}$, může být omezená či

13. Historie programovacích jazyků

John von Neumann (1903 - 1957):

Teoretické schéma počítače tvořené: procesor, řadič, operační paměť, vstupní a výstupní zařízení.

Základ architektury současných počítačů.

Alan Turing (1912 - 1954):

Zabýval se problematikou umělé inteligence

Může stroj řešit problémy.

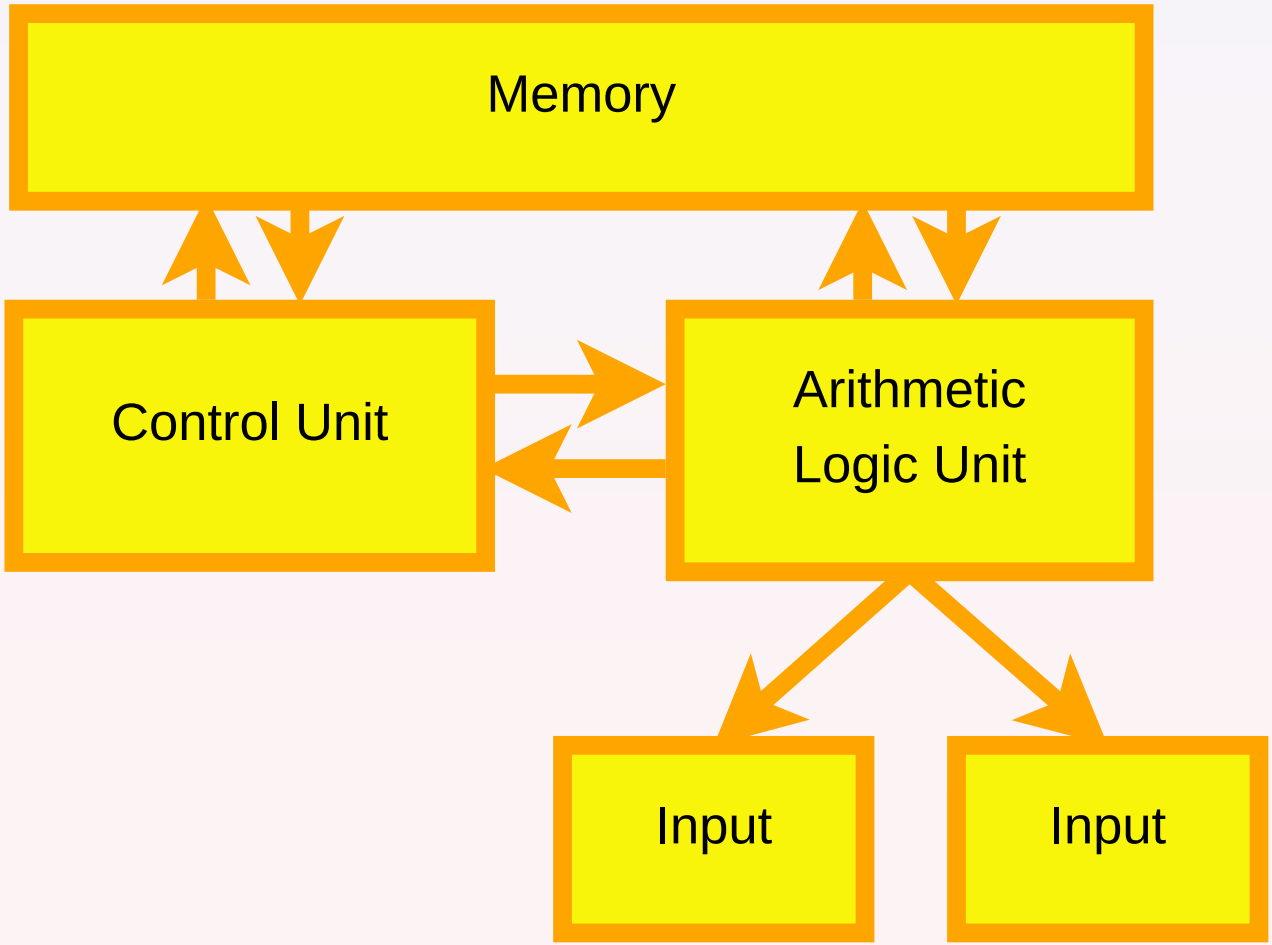
Vytvořil abstraktní model počítače, tzv. Turingův stroj (eliminaci závislosti na hardware).

Konrad Zuse (1910 - 1995):

První programovací jazyk (1946), Plankalkul.

Nikdy nebyl implementován.

14. von Neumannovo schéma počítače



15. Vývoj programovacích jazyků (1/4)

FORTRAN (FORmulaTRANslator, 1954-57)

Autorem IBM, používán pro vědecké výpočty a numerické aplikace.

Kompilovaný jazyk.

Využíván dodnes, řada vědeckých projektů napsána ve Fortranu.

FORTRAN 77, Fortran 90, Fortran 95, Fortran 2000, Fortran 2003 a Fortran 2008.

ALGOL (ALGOrithmic Language, 1958-59)

Jazyk pro popis algoritmů...

Moderní koncepce, implementace rekurze.

Podobný současným programovacím jazykům.

Ve své době velmi populární.

LISP (List Processing, 1958-59)

Používán v oblasti umělé inteligence.

16. Vývoj programovacích jazyků (2/4)

COBOL (Common Business Oriented Language, 1959)

Zápis programů v jazyce blízkém angličtině, přístupnost široké veřejnosti.

Vytváření rozsáhlých programů k obchodním účelům.

Snadný převod programů mezi počítači.

Mnoho verzí, poměrně dlouho vyvíjen.

BASIC (Beginners All-Purpose Symbolic InstructionCode, 1964)

Jednoduchý jazyk pro řešení jednoduchých úkolů výuku programování.

Nerozlišoval datové typy.

Používán dodnes, doplněn o možnost objektového progrmování (Visual Basic).

Pascal (NiklausWirth, 1971)

Navržen pro výuku programování, odvozen za Algolu.

17. Vývoj programovacích jazyků (3/4)

Simula (1967)

První objektově orientovaný programovací jazyk.

Odvozen z Algolu.

Používán pro vědecké výpočty a simulace, možnost paralelizace výpočtů.

Uplatnil se pouze v akademickém prostředí, v praxi příliš nepoužit.

Přinesl řadu moderních prvků, např. garbage collector (převzala Java).

Smalltalk (Xerox)

Interpretovaný objektově orientovaný jazyk.

Dodnes velmi často používán, současná implementace s evýtazně liší od původní.

Zavedl událost (posílání zpráv).Možnost vyjádření grafického návrhu formalizovaným jazykem.

- Propojení vizuálního a textového popisu.

18. Vývoj programovacích jazyků (4/4)

C++ (Bjarne Stroustrup, 1982-85)

Rozšířením jazyka C možnosti objektového a generického programování

Zřejmě nejrozšířenější programovací jazyk.

První standardizace až 1997, další 2003, 2006, 2007.

Java (Sun, 1994-1995)

Původně navržen jako jazyk pro inteligentní domácí spotřebiče.

Interpretovaný jazyk, nezávislost na hardware a operačním systému.

Snaha o zjednodušení a odstranění nebezpečných konstrukcí z C++ (automatická správa paměti, kompletně objektový).

C# (Microsoft, 2002)

Objektově orientovaný interpretovaný jazyk založený na C++ a Javě.

Základ platformy NET.

19. Překlad programu

Editor:

Zápis zdrojového kódu v textovém souboru.

Čisté ASCII bez formátování, nepoužívat textové editory.

Preprocesor:

Předzpracování zdrojového kódu, vynechání komentářů, správné includování souborů, zpracování maker.

Výsledkem opět textový soubor.

Kompilátor:

Překlad textového souboru vytvořeného preprocesorem do assembleru.

Následně překlad do relativního kódu (absolutní adresy proměnných a funkcí nemusí být ještě známy), tzv. Object code, *.obj.

Každý programový modul tvořen samostatným *.obj souborem.

20. Překlad programu

Linker:

Spojení jednotlivých *.obj souborů generovaných kompilátorem.

Relativní adresy nahrazeny absolutními.

Výsledkem přímospustitelný kód.

Debugger:

Slouží pro ladění programu.

Hledání chyb nastávající za běhu programu.

Nástroje break point, watch, step into/over, conditional break point, memory leak detection.

Po nalezení chyby opakujeme proces od kompilace znovu.

Nejčastější chyby:

Neinicializované proměnné, použití ukazatele s hodnotou NULL, chyby při práci s pamětí, zakrytí lokálních proměnných, překročení indexu, nevhodný typ návratové hodnoty.

21. Paradigmata programování

Paradigma:

Souhrn základních domněnek, předpokladů, představ dané skupiny vědců.

Souhrn způsobů formulace problémů, metodologických prostředků řešení, metodik zpracování apod.

Paradigma v programování ovlivňuje styl programování.

Definuje jakým způsobem vnímá programátor problém, a jak ho řeší.

Různá paradigmata:

Procedurální (imperativní) programování: Základem algoritmus, přesně daná posloupnost kroků.

Objektově orientované programování: Založeno na objektově orientovaném přístupu k problému.

Deklarativní programování: Opak imperativního programování (říkáme co, ne jak)

Funkcionální programování: Varianta dekalarativního programování, založeno na lambda kalkulu.

22. Dělení programovacích jazyků

Kritéria pro dělení programovacích jazyků:

- *Podle způsobu zápisu instrukcí*
  Nižší programovací jazyky.
  Vyšší programovací jazyky.
- *Podle způsobu překladu*
  Kompilované jazyky.
  Interpretované jazyky.
- *Podle způsobu řešení problému*
  Procedurální jazyky.
  Neprocedurální jazyky.

23. Dělení programovacích jazyků

Nižší programovací jazyky:

Programování je prováděno ve strojových instrukcích.

Výsledný kód velmi rychlý, na programátora však tento přístup klade značné nároky.

Program je závislý na typu procesoru; lze jej spustit např. na procesorech firmy Intel a nikoliv na procesorech firmy Motorola.

Zástupci:strojový kód, Assembler.

```
mov al, 61h
```

Vyšší programovací jazyky:

Používají zástupné příkazy nahrazující skupiny instrukcí, které jsou následně překládány do strojového kódu.

Struktura programu je přehlednější, vývoj programu je jednodušší, program není závislý na typu procesoru.

24. Interpretované jazyky

Obsahují interpret, program vykonávající přímo instrukce v programovacím jazyce.

3 typy interpretru:

- přímé vykonávání zdrojového kódu
  Neprováděna kompilace, nevýhodou pomalost. Zástupce: Basic.
- překlad do mezikódu
  Zdrojový kód přeložen do mezikódu a vykonán.
- spuštění předkompilovaného mezikódu
  Předem vytvořen mezikód (bajtový kód) nezávislý na operačním systému. Zástupce: Java, C#.

Nevýhody:

Interpret výrazně pomalejší než kompilátor.

Pomalejší běh programu, větší spotřeba hardwarových prostředků.

Výhody:

Nezávislost na platformě, hardwaru.

25. Kompilované jazyky

Program je přeložen do strojového kódu a poté může být opakovaně spouštěn již bez přítomnosti interpretru.

Překlad se provádí kompilátorem, výsledkem bývá spustitelný soubor (např. exe na platformě Windows).

Výhody:

Kód několikanásobně rychlejší než u jazyků interpretovaných.

Nižší hardwarové nároky.

Běh bez interpreteru.

Výhody:

Program není univerzálně použitelný pod různými OS.

Složitější syntaxe.

Zástupci: C, C++, Fortran, Pascal.

26. Procedurální jazyky

Označovány jako imperativní jazyky.

Zaměřeny na algoritmus, řešení problému popsáno pomocí posloupnosti příkazů, tj. předpisem, jak úlohu vyřešit.

Většina programovacích jazyků umožňuje imperativní přístup.

Dělení do dvou skupin:

- *Strukturované jazyky:*
  Algoritmus rozdělen na dílčí kroky realizované podprogramy, které se spojují v jeden celek.
  Zástupce: C.
- *Objektově orientované jazyky:*
  Snaha modelovat a nacházet řešení úloh postupy blízkými skutečnému lidskému uvažování.
  Využití objektového modelu, každý objekt má určité chování a vlastnosti.
  Objekty spolu mohou komunikovat.
  Zástupce: Java, C#.

27. Neprocedurální jazyky

Označovány jako *deklarativní* jazyky.

Založeny na myšlence deklarativního programování:

- Definujeme pouze problém a nikoliv jeho řešení.
- Vlastní algoritmizaci problému řeší překladač.

Cílem eliminace častých chyb programátora.

Neobsahují cykly, příkazy pro opakování nahrazeny prostřednictvím rekurze.

Pořadí jednotlivých příkazů není důležité, kód nemusí být zpracováván lineárně, tj. od počátku.

Zástupci: SQL, Scheme.

28. Zásady pro zápis zdrojového kódu

Zápisu programového kódu nutno věnovat zvýšenou pozornost.

Nutno dodržovat níže uvedené zásady a pravidla.

Cílem je tvorba *snadno udržovatelného kódu*.

Vytvoření programu není jednorázová činnost.

Jak se vyvíjí znalosti a schopnosti programátora, objevují se nové postupy řešení, měl by na ně autor programu adekvátně reagovat.

Program by měl být psán přehledně a srozumitelně, aby ho bylo možné snadno modifikovat a doplňovat o novou funkcionalitu.

V opačném případě se mohou doba a úsilí vynaložené na implementaci i poměrně jednoduché změny rovnat době odpovídající kompletnímu přepsání programu.

29. Čitelnost a dodržování konvencí

Čitelnost:

Zdrojový kód by měl být zapsán tak, aby byl přehledný a snadno čitelný.

Není vhodné používat nejasné programové konstrukce znesnadňující pochopení činnosti konstrukce.

Nutno dodržovat níže obecná pravidla formátování zdrojového kódu.

Dodržování konvencí:

Nutnost dodržování syntaktických pravidel a konvencí formálního jazyku.

Pro každý formální jazyk existuje seznam konvencí.

Nedoporučuje se obcházet pravidla nestandardním způsobem.

Nestandardní konstrukce obtížněji čitelné, mohou u nich nastat vedlejší efekty, špatně se upravují a rozšiřují.

30. Formátování zdrojového kódu (1/4)

Zásada č. 1:

1 příkaz na jeden řádek.

```
if (i==10) a++;b--; System.out.println(b); //Spatne
if (i==10) a++; //Dobre
b--;
System.out.println(b);
```

Zásada č. 2:

Odsazování řádků v podřízených blocích.

```
if (i!=j){ //Spatne
a=i*j
b--;
System.out.println(b)}
if (i!=j) //Dobre
 {
   a=i*j
   b--;
   System.out.println(b)
```

31. Formátování zdrojového kódu (2/4)

Zásada č. 3:

Vynechávání řádků mezi logickými celky, podcelky, metodami,...

```
public void simplify()      //Zacatek jedne metody
{
      double x=new ArrayList<double>();
      double y=new ArrayList<double>();
        ...
}                           //Konec jedne metody
                            //Prazdny radek
public void clearAll() //Zacatek druhe metody
{
      X.clear();
      Y.clear();
}                           //Konec druhe metody
```

32. Formátování zdrojového kódu (3/4):

Zásada č. 4:

Důsledné používání komentářů (Co, jak, proč, kdy, kým), zásadně v AJ!

```
if (v == 1) //Vertex code
{
        //Compute first mid-point
        x1 = (x_point.get(v2) + x_point.get(v3)) / 2.0;
        y1 = (y_point.get(v2) + y_point.get(v3)) / 2.0;

        //Compute second mid-point
        x2 = (x_point.get(v1) + x_point.get(v3)) / 2.0;
        y2 = (y_point.get(v1) + y_point.get(v3)) / 2.0;
```

Zásada č. 5:

Rozlišování malých a velkých písmen (Case Sentivity!).

```
double a;
double A;  //Jedna se o jine promenne
sin(a);    //Spravne
Sin(a);    //Spatne
```

33. Formátování zdrojového kódu (4/4):

Zásada č. 6:

Délka řádku menší než šířka obrazovky.

Zásada č. 7:

Zalamování dlouhých řádků na logickém místě.

Zásada č. 8:

Rozumná délka podprogramů, max. 150 řádek.

Zásada č. 9:

Za čárkou a středníkem mezera, před a za závorkou nikoliv.

```
public void addPoint( double x,double y,double z )  //Spatne
{
   x_point.add(x);
   y_point.add(y);
   z_point.add(z);
}
public void addPoint(double x, double y, double z)  //Dobre
{
   x_point.add(x);
```

34. Jazyk a jeho používání

Zásada č. 10:

V programu nepoužívat více než jeden jazyk (nekombinovat ČJ a AJ).

Zásada č. 11:

Názvy proměnných, funkcí, tříd, komentářů uvádíme zpravidla v anglickém jazyce.

Zásada č. 12:

Nepoužívat diakritiku v komentářích, názvech projektů, názvech souborů, identifikátorech, metodách.

```
class Draw
{
   private int sirka, delka;  //Spatne
   private int width, length; //Dobre
   ...
}
```

35. Identifikátory proměnných, metod, tříd (1/3)

Zásada č. 13:

Co nejvýstižnější názvy identifikátorů proměnných, funkcí, tříd.

```
//Spatne
class A;
double k, l, m;
int x(double y);

//Spravne
class Graphics;
double scale, rotation, distance;
int degrees(double angle)
```

Zásada č. 14:

Nepoužívat podobné názvy identifikátorů.

```
double aaa, aab, aac;  //Spatne
```

36. Identifikátory proměnných, metod, tříd (2/3)

Zásada č. 14:

Nepoužívat identifikátory s délkou větší než 20 znaků.

Zásada č. 15:

Víceslovné identifikátory spojovat podtržítky

```
double maxangulardistortion;    //Spatne
double max_angular_distortion;  //Dobre
```

Zásada č. 16:

Jména konstant s velkými písmeny (odlišení od proměnných)

```
final double a_Bessel=6377397,1550;  //Spatne
final double A_BESSEL=6377397,1550;  //Dobre
```

Zásada č. 17:

Jména proměnných začínají malými písmeny.

```
double X_Jtsk;  //Spatne
double x_jtsk;  //Dobre
```

37. Identifikátory proměnných, metod, tříd (3/3)

Zásada č. 18:

Jména metod začínají malými písmeny.

```
double Rotation;  //Spatne
double rotation;  //Dobre
```

Zásada č. 19:

Jméno třídy začíná velkým písmenem.

```
class graphics  //Spatne
class Graphics  //Dobre
```

Zásada č. 21:

Ve zdrojovém kódu nepoužívat čísla bez identifikátoru.

Složité záměny hodnot v celém programu.

```
double s1, s;                 //Spatne
s=s1+125.55;

double s, s1, s2=125.55;  //Dobre
s=s1+s2;
```

38. Podmínky a jejich zápis

Zásada č. 22:

Podmínky zapisovat v pozitivní formě.

```
if (!(a < b))  //Spatne
if (a >= b)    //Dobre
```

Zásada č. 23:

Vyvarovat se nekonečných podmínek.

Pozor na kontradikce a tautologie !!!

```
while (x || !x)  //Spatne, tautologie, nekonecny cyklus
while (x && !x)  //Spatne, kontradikce, neprobehne ani jednou
if (true)        //Spatne, probehne vzdy
if (x || 1)      //Spatne,probehne vzdy
```

Zásada č. 24:

Nepoužívat příkazy pro skok, vedou k nepřehlednosti programu.

39. Komentáře

Pomáhají udržovat přehlednost kódu.

Co je nám jasné dnes, nemusí být jasné zítra, za měsíc, či za rok.

Zásada č. 25:

Komentovat významná místa v kódu (nepřehledná, složitá, ...).

Zásada č. 26:

Komentovat proměnné a jejich významy.

Zásada č. 27:

Komentovat hlavičky metod.

Zásada č. 28:

Komentovat změny v kódu.

Zásada č. 29:

Komentovat metody.

Zásada č. 30:

Rozumná délka komentářů.

Zásada č. 31: