1. Úvod

http://www.kiv.zcu.cz/~jezek_ka/vyuka/PGS

Obsah předmětu =komparativní studie programovacích jazyků

1. Vývoj programovacích jazyků, styly a vlastnosti
2. Paralelní programování přehled
3. Paralelní programování v jazyce Java
4. Paralelní programování v jazyce Java, synchronizace
5. Python
6. Skripty v Pythonu
7. Python a XML
8. Logické programování
9. Prolog
10. Funkcionální programování
11. Lisp
12. Porovnání vlastností jazyků
13. Úvod do překladačů

Podmínky absolvování

Podmínky udělení zápočtu:

- (vypracování programů)

Forma zkoušky písemný test a příklady – archivuje se

Termíny zkoušky: řádný
1.opravný
2.opravný (poslední)

Vývoj programovacích jazyků

Konec 40. let – odklon od strojových kódů

Pseudokódy:

- Pseudooperace aritm. a matem. funkcí
- Podmíněné a nepodmíněné skoky
- Autoinkrement. registry pro přístup k polím

Vývoj programovacích jazyků

50. léta

- **první definice vyššího programovacího jazyka (efektivita návrhu programu)**
- **FORTRAN (formula translation - Backus), vědeckotechnické výpočty, komplexní výpočty na jednoduchých datech, pole, cykly, podmínky**
- **COBOL (common business lang.), jednoduché výpočty, velká množství dat, záznamy, soubory, formátování výstupů**
- **ALGOL (algorithmic language - Backus, Naur), předek všech imperativních jazyků, bloková struktura, rekurze, volání param. hodnotou, deklarace typů**
- **nové idee - strukturování na podprogramy, přístup ke globálním datům (Fortan), bloková struktura, soubory, …**
- **stále živé - Fortran, Cobol**

Vývoj programovacích jazyků

první polovina 60. let

- **začátek rozvoje neimperativních jazyků**
- **LISP (McCarthy) - založen na teorii rekurzivních funkcí, první funkcionální jazyk, použití v UI (symbolické manipulace)**
- **APL - manipulace s vektory a s maticemi**
- **SNOBOL (Griswold) - manipulace s řetězci a vyhledávání v textech, podporuje deklarativní programování**
- **vzniká**
- **potřeba dynamického ovládání zdrojů,**
- **potřeba symbolických výpočtů**
- **----------**

Vývoj programovacích jazyků

pozdní 60. léta

- **IBM snaha integrovat úspěšné koncepty všech jazyků - vznik PL/1 (moduly, bloky, dynamické struktury)**
- **nové prvky PL/1 - zpracování výjimek, multitasking. Nedostatečná jednotnost konstrukcí, komplikovanost**
- **ALGOL68 - ortogonální konstrukce, první jazyk s formální specifikací (VDL), uživatelsky málo přívětivý, typ reference, dynamická pole**
- **SIMULA67 (Nygaard, Dahl) - zavádí pojem tříd, hierarchie ke strukturování dat a procedur, ovlivnila všechny moderní jazyky, corutiny**
- **BASIC (Kemeny) - žádné nové konstrukce, určen začátečníkům, obliba pro efektivnost a jednoduchost, interaktivní styl programování**
- **Pascal (Wirth) - k výuce strukturovaného programování, jednoduchost a použitelnost na PC zaručily úspěch**

Vývoj programovacích jazyků

70. léta

- **důraz na bezpečnost a spolehlivost**
- **abstraktní datové typy, moduly, typování, dokazování správnosti, práce s výjimkami**
- **CLU (datové abstrakce), Mesa (rozšíření Pascalu o moduly), Concurrent Pascal, Euclid (rošíření Pascalu o abstraktní datové typy)**
- **C (Ritchie) - efektivní pro systémové programování, efektivní implementace na různých počítačích, slabé typování**
- **Scheme - rozšířený dialekt LISPu**
- **PROLOG (Colmeraurer) - první logicky orientovaný jazyk, používaný v UI a znalostních systémech, neprocedurální, „inteligentní DBS odvozující pravdivost dotazu“**

Vývoj programovacích jazyků

80. léta

- **Modula2 (Wirth) - specifické konstrukce pro modulární programování**
- **další rozvoj funkcionálních jazyků - Scheme (Sussman, Steele, MIT), Miranda (Turner), ML (Milner) - typová kontrola**
- **ADA (US DOD) syntéza vlastností všech konvenčních jazyků, moduly, procesy, zpracování výjimek**
- **průlom objektově orientovaného programování - Smalltalk (Key, Ingalls, Xerox: Datová abstrakce, dědičnost, dynamická vazba typů), C++ (Stroustrup 85- C a Simula)**
- **další OO jazyky - Eiffel (Mayer), Modula3, Oberon (Wirth)**
- **OPS5, CLIPS - pro zpracování znalostí**

Vývoj programovacích jazyků

- **90. léta**
- **jazyky 4.generace, QBE, SQL - databázové jazyky**
- **Java (SUN) - mobilita kódu na webu, nezávislost na platformě**
- **vizuální programování (programování ve windows) - Delphi, Visual Basic, Visual C++**
- **ADA95**
- **jazyky pro psaní CGI (common gateway interface) skriptů:**

✓ **Perl (Larry Wall - Pathologically Eclectic Rubbish Lister) - nástroj pro webmastery a administrátory systémů,**

✓ **JavaScript - podporován v Netscape i v Explorer,**

✓ **VBScript,**

✓ **PHP, Python – freeware**

✓ **Ruby**

- **2000 C Sharp (C#)**

Vývoj programovacích jazyků (Fortran)

Fortran 0 – 1954 neimplementován

Fortran 1 – 1957 (hlavní kriterium=efektivita)

- Bez dynamické paměti
- Rychlý přístup k polím (posttest for cyklus)
- Jména max 6 znaková
- Formátovaný I/O
- Uživatelské podprogramy
- Třícestná selekce (aritmetický IF)
- 18 člověkoroků trvala implementace

Vývoj programovacích jazyků (Fortran)

Fortran II – 1958

- Separátní překlad

Fortran IV – 1960-1962

- Explicitní deklarace typů
- Logický IF příkaz tvaru if bool.výraz příkaz
- Podprogramy s parametry
- ANSI standard vyšel v r.1966

Vývoj programovacích jazyků (Fortran)

Fortran 77 - 1978

- Manipulace s řetězci znaků
- IF THEN ELSE příkaz
- Log. hodnotou řízený cyklus

Fortran 90 - 1990

- Moduly, Kontrola typů parametů
- Dynamická pole, Ukazatele
- Case příkaz
- Rekurze

HP Fortran 95

OOP Fortran 2003

Concurrent Fortran 2008

Fortran 2015

Fortran 2018

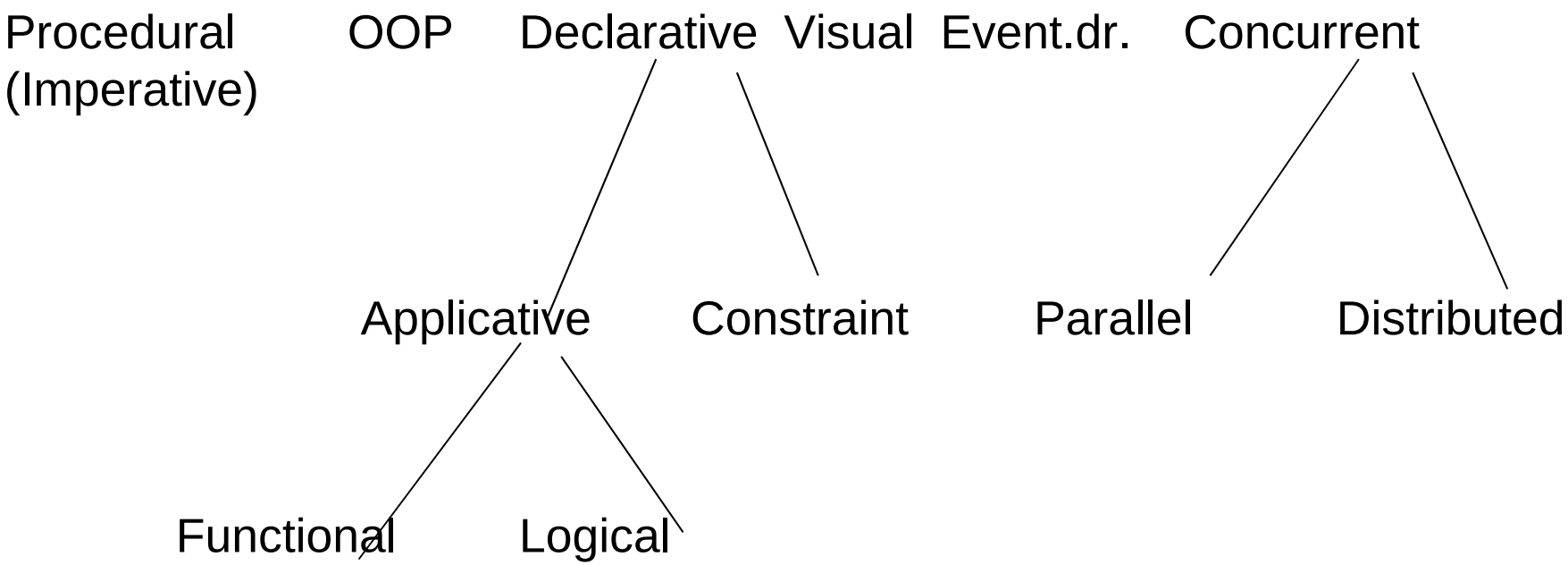
Paradigmata programování

Paradigma = **souhrn způsobů formulace problémů,**
metodologických prostředků řešení,
metodik zpracování a pod.

- **Procedurální (imperativní) programování**
- **Objektově orientované programování**
- **Generické programování**
- **Komponentově orientované programování**
- **Událostní programování (event-driven prog.)**
- **Deklarativní programování**
- **Aplikativní programování**
- **Funkcionální programování**
- **Logické programování**
- **Programování ohraničeními (constraint progr.)**
- **Vizuální programování**
- **Aspektově orientované programování**
- **Souběžné programování - paralelní**
  - distribuované

V češtině místo souběžné se používá také pojem paralelní

Paradigmata programování –jejich vztahy



Jsou ještě další styly: aspektový, komponentový, generický, …
Koexistence paradigmat je ve většině užívaných jazyků

Procedurální (imperativní)

Základem je přiřazování. Příkazy se provádějí sekvenčně. Strukturované. Modulární

OOP

Program je množina objektů, ty mají stav, jméno, chování). Předávají si zprávy

Generické programování

Využívání možnosti parametrizace obecného popisu datových typů, funkcí, …, které představují vzory, dle nichž jsou dodatečně konkrétní t., f.,…instalovány.

Komponentově orientované programování

Využívání prefabrikovaných komponent (násobná použitelnost, nezávislost na kontextu, slučitelnost s ostatními komponentami, zapouzdřitelnost). Někteří to nepovažují za paradigma a řadí je do OOP.

Constraint progr.

Vyjadřují výpočet pomocí relací mezi proměnnými (obdoba spreadsheet výpočtu)

Např. celsius = (fahr -32)*5/9 --definuje relaci, není to přiřazení.

Často kombinace constraint logic programming

Vizuální programování

Specifikuje program interaktivně pomocí grafických prvků (ikon, formulářů), např. LabVIEW. Microsoft Vis.Stud. (VB, VC#,..) nejsou vizuální jazyky, ale textové.

Událostní programování

Tok programu je určen akcemi uživatele (např. click myší) nebo zprávami z jiných programů. Je opakem dávkového (batch) programování.

Př. verze sečtení čísel dávkově

read a number (from the keyboard) and store it in variable A[0]
read a number (from the keyboard) and store it in variable A[1]
print A[0]+A[1]

Př.event-driven verze téhož

```
set counter K to 0
repeat {
    if a number has been entered (from the keyboard)
          { store in A[K] and increment K
            if K equals 2 print A[0]+A[1] and reset K to 0
          }
}
```

Aspektově orientované programování

```
void transfer(Account fromAccount, Account toAccount, int amount) {
    if (fromAccount.getBalance() < amount) {
        throw new InsufficientFundsException();
    }
    fromAccount.withdraw(amount);
    toAccount.deposit(amount);
}
```

Takový transfer peněz má vady (nezkouší autorizaci, není transakční)

Pokus ošetřit vady rozptýlí kontroly přes celý program (obvykle metody, moduly). Viz %

AOP se snaží řešit problém modularizováním těchto záležitostí do aspektů, tj. separovaných částí kódu, ze kterých se tyto záležitosti (např zajištění perzistence=trvalosti) pohodlněji spravují.

```
void transfer(Account fromAccount, Account toAccount, int amount) {
    if (!getCurrentUser().canPerform(OP_TRANSFER)) {
            throw new SecurityException();
    }
    if (amount < 0) {
            throw new NegativeTransferException();
    }
    if (fromAccount.getBalance() < amount) {
            throw new InsufficientFundsException();
    }
    Transaction tx = database.newTransaction();
    try {
       fromAccount.withdraw(amount);
       toAccount.deposit(amount); tx.commit();
      systemLog.logOperation(OP_TRANSFER, fromAccount, toAccount,
    amount);
    }
    catch(Exception e) { tx.rollback();
}   }
```

Imperativní programovací styl

Program pro nalezení největšího společného dělitele v C

```
#include <stdio.h>

int nsd(int u, int v)
{   if (v == 0) return u;
    else return nsd (v, u % v);
}

main()
{  int x, y;
   printf("vstup dvou celych cisel:\n");
   scanf("%d%d",&x,&y);
   printf("nsd z %d a %d je %d\n",x,y,nsd(x,y));
   getchar(); getchar();
}
```

```
import java.io.*;
class IntWithNsd
{ public IntWithNsd( int val ) { value = val; }
  public int getValue() { return value; }
  public int nsd ( int v )
  { int z = value;
    int y = v;
    while ( y != 0 )
    { int t = y;
      y = z % y;
      z = t;
    }
    return z;
  }
  private int value;
}
class NsdProg
{ public static void main (String args[])
  { System.out.println(" vstup dvou celych cisel:");
    BufferedReader in =
      new BufferedReader(new InputStreamReader(System.in));
    try
    { IntWithNsd x = new IntWithNsd(Integer.parseInt(in.readLine()));
      int y = Integer.parseInt(in.readLine());
      System.out.print(" nsd z " + x.getValue() + " a " + y + " je ");
      System.out.println(x.nsd(y)); /*objektu x zasila zpravu nsd*/
    }
    catch ( Exception e)
    { System.out.println(e);
      System.exit(1);
} } }
```

Logický styl

Příklad výpočtu největšího společného dělitele v jazyce Prolog.

Program definuje fakta a predikáty

```
nsd(U, V, U) :- V = 0 .
nsd(U, V, X) :- not(V = 0),
                Y is U mod V,
                nsd(V, Y, X).
```

Zápis má význam: predikát nsd s argumenty U, V, U platí, když buď V = 0 nebo V ≠ 0 , pak Y přiřaď U mod V a pak vyvolej rekurzivně predikát (logickou fci) nsd s argumenty V, Y, X.

Při ověřování platnosti predikátů se výpočet snaží dosadit do proměnných takové hodnoty, aby predikáty platily.

Lze se pak dotazovat na jejich pravdivost a Prolog vypíše hodnoty proměnných, pro které jsou predikáty true.

?- nsd(12, 16, Vysledek). Odpoví Vysledek = 4

Funkcionální styl

Příklad výpočtu největšího společného dělitele v jazyce LISP (definuje funkci)

Funkce nsd s argumenty u, v má hodnotu u pokud v =0,
jinak má hodnotu rekurzivně vyvolané funkce nsd s
s prvým argumentem v a druhým argumentem je
hodnota u modulo v.

```
(defun nsd (u v)
  (if (= v 0) u
     (nsd v (mod u v))))
```

Funkci pak lze vyvolat zápisem

(nsd 12 16) a výpočet vrátí hodnotu nsd(12 16) tj. 4

```
with TEXT_IO; use TEXT_IO;              --“GENERICKÝ STYL v Jazyce ADA“
procedure nsd_prog is
  function nsd (u, v: in integer) return integer is
    y, t, z: integer;
  begin
    z := u;
    y := v;
    loop
      exit when y = 0;
      t := y;
      y := z mod y;
      z := t;
    end loop;
    return z;
  end nsd;
  package INT_IO is new INTEGER_IO(Integer); --vygeneruje balik ze sablony
  use INT_IO;
  x: Integer;
  y: Integer;
begin
  put_line("vstup dvou celych cisel:");  get(x);  get(y);
  put("nsd z ");  put(x);  put(" and ");  put(y);
  put(" je ");  put(nsd(x,y));
  new_line;
end nsd_prog;
```

Globální kriteria na programovací jazyk

1. Spolehlivost (typová kontrola, výjimky,…)
2. Efektivita (programování, překládání, výpočtu, údržby)
3. Strojová nezávislost (pro různé platformy)
4. Čitelnost a vyjadřovací schopnosti
5. Řádně definovaná syntax a sémantika (tvar a význam)
6. Úplnost v Turingově smyslu (zapsatelnost jakéhokoliv algoritmu)

Globální kriteria na programovací jazyk

Ad 1

Typová kontrola (při překladu i při výpočtu)

Zpracování výjimečných situací (k zachycení chybových a nestandardních situací při výpočtu)

Ad 2

Ef. překladu (konstrukce pohodlně přeložitelné)

Ef. výpočtu (efektivní jsou statické datové typy, málo efektivní je garbage collector = čistič paměti)

Globální kriteria na programovací jazyk

Ad 4

- jednoduchost (kontra příklad: C=C+1; C+=1; C++; ++C) více možností k zapsání téhož je matoucí
- Ortogonalita (malá množina primitivních konstrukcí, z té lze kombinovat další konstrukce. Všechny kombinace jsou legální) kontra př. v C:
  - struktury mohou být funkční hodnotou
  - ale pole nemohou
- Strukturované příkazy
- Strukturované datové typy
- Podpora abstrakčních prostředků
- Expresivita (např. vhodnější jsou dynamické typy oproti statickým, existují jen tam, kde jsou třeba. Zhorší ale efektivitu)
- Uniformita = podobné věci mají vypadat podobně (kontra př. C deklarace fcí nekončí středníkem ale deklarace dat končí středníkem

Globální kriteria na programovací jazyk

Ad 4

- Strojová čitelnost

  = existence algoritmu překladu s lineární časovou složitostí, což je podmíněno existencí bezkontextové syntaxe jazyka

- Humánní čitelnost

  silné odvisí od způsobu abstrakcí. Abstrakce rozdělujeme na

  abstrakce dat (int, real, struct, …)

  abstrakce řízení běhu programu (if, while, funkce,…)

  bohaté komentování

  čitelnost je v kontra se zapisovatelností (více práce s identifikátory a s komentáři)

Globální kriteria na programovací jazyk

Ad 5

Syntax = forma či struktura výrazů, příkazů a programových jednotek

Sémantika = význam výrazů, příkazů a programových jednotek

Definici jazyka potřebují

- návrháři jazyka
- implementátoři
- uživatelé jazyka

Globální kriteria na programovací jazyk

Ad 6

- Turingův stroj = jednoduchý ale neefektivní počítač použitelný jako formální prostředek k popisu algoritmu
- Programovací jazyk je úplný v Turingově smyslu, jestliže je schopný popsat libovolný výpočet (algoritmus)
- Co je potřebné pro Turingovu úplnost?

  Téměř nic: Stačí

  - ✓ celočíselná aritmetika a
  - ✓ celočíselné proměnné spolu se sekvenčně prováděnými příkazy zahrnujícími
  - ✓ přiřazení a
  - ✓ cyklus (While)

Syntax

Formálně je jazyk množinou vět

Věta je řetězcem lexémů (terminálních symbolů)

Syntax lze popsat:

- Rozpoznávacím mechanismem - automatem (užívá jej překladač)
- Generačním mechanismem – gramatikou (to probereme)

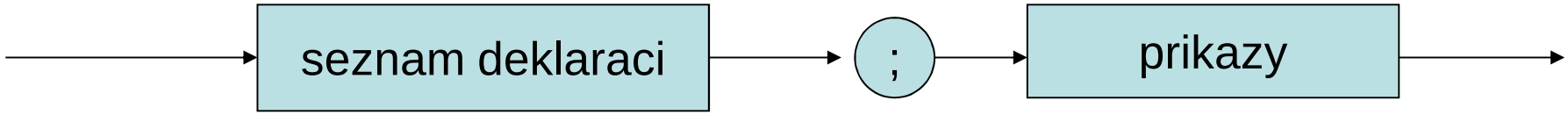
Syntax

Gramatika je čtveřice (N, T, P, S)
N je množina neterminálních symbolů (níže jsou označeny < >
T je množina terminálních symbolů = lexémů jazyka
P je množina pravidel
S je počáteční symbol (pro programovací jazyk je S = <program>
Bezkontextová gramatika má pravidla tvaru:
neterm.symbol → řetězec symbolů terminálních i neterm.
Backus Naurova forma (BNF) – používá metajazyk: < >, | , někdy
místo → také ::= má význam „přepisuje se“
| má význam „nebo“
Př.
<program> → <seznam deklaraci> ; <prikazy>
<seznam deklaraci> →
<deklarace> |
<deklarace>;<seznam deklaraci>
<deklarace> → <spec. typu> <sez. promennych>

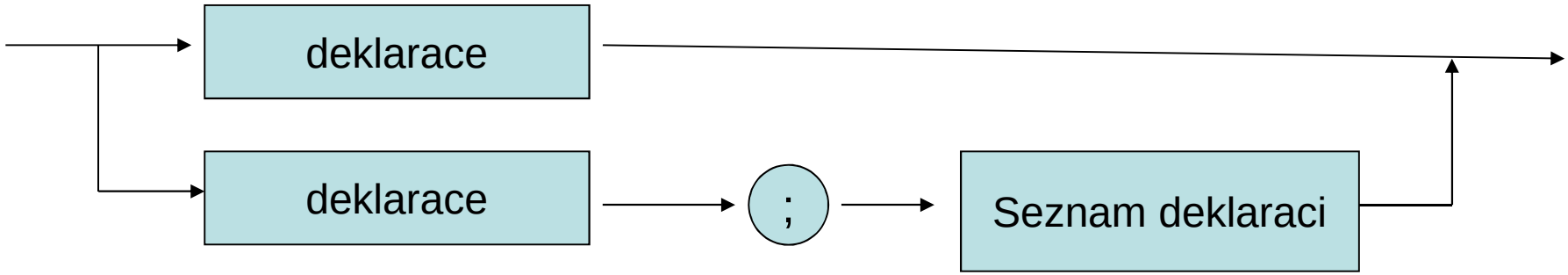
Syntax

Syntaktické diagramy=jiná možnost zápisu syntaxe

program



Seznam deklarací



Syntax

Derivování (značíme =>) je odvození řetězce podle pravidel gramatiky

Př.

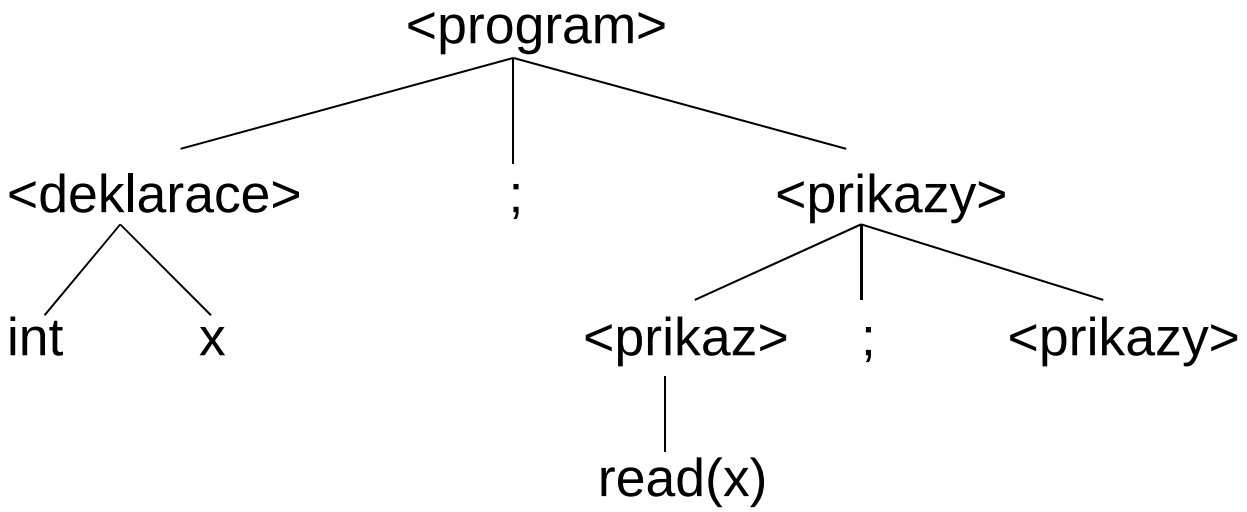
<program> => <seznam deklaraci> ; <prikazy> =>

$\Rightarrow$ <deklarace> ; <prikazy> =>int x ; <prikazy> =>

$\Rightarrow$ int x;<prikaz> ; <prikazy> => int x ; read(x);<prikazy> =>

$\Rightarrow$ int x ; read(x) ; <prikazy> => …

Derivační strom = grafické vyjádření struktury



Sémantika

Sémantiku rozlišujeme statickou a dynamickou

A. Statická sémantika vyjadřuje kontextové závislosti, ty nelze vyjádřit bezkontextovou syntaxí (např je-li x deklarováno jako int, musí být v příkazech používáno jako int)

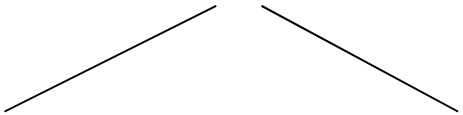
B. Dynamická sémantika vyjadřuje vlastní význam zápisu. Její formalizace je obtížnější.

B. používají různé formalismy

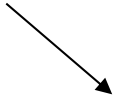
- Operační sémantiky chápou program jako předpis operací automatu. Význam operací je dán posloupností stavu automatu.
- Axiomatické sémantiky vyjadřují význam pomocí podmínek (logických výrazů), které platí před a po provedení příkazu
- Denotační sémantiky vyjadřují stav programu matematickou funkcí. Matem. objekty označují (denotují) význam jazykových entit.

Překlad jazyka

- Kompilátor: dvoukrokový proces překládá zdrojový kód do cílového kódu. Následně uživatel sestaví a spustí cílový kód
- Interpret: jednokrokový proces, „zdrojový kód je rovnou prováděn“
- Hybridní: např. Java Byte-code – soubory *.class



interpretován nebo přeložen JIT kompilátorem



*.exe soubor

Klasifikace chyb

- Lexikální - např nedovolený znak
- Syntaktické -chyba ve struktuře
- Statické sémantiky –např nedefinovaná proměnná, chyba v typech. Způsobené kontextovými vztahy. Nejsou syntaktické
- Dynamické sémantiky – např dělení 0. Nastávají při výpočtu, neodhalitelné při překladu. Nejsou syntaktické
- Logické – chyba v algoritmu

- ❖ Kompilátor je schopen nalézt lexikální a syntaktické při překladu
- ❖ Statické sémantiky chyby může nalézt až před výpočtem (program může být překládán po částech a jejich vazby lze ověřit až poté)
- ❖ Nemůže při překladu nalézt chyby v dynamické sémantice, projeví se až při výpočtu
- ❖ Žádný překladač nemůže hlásit logické chyby
- ❖ Interpret obvykle hlásí jen lexikální a syntaktické chyby když zavádí program

Klasifikace chyb př. Java

```
public int nsd ( int v, u# )  {
    int  a = v
    b = v;
    while  ( b != 0 ) {
        int p = b;
        b = a % b;
        a = p;
    }
    return b;
}
```

lexikální chyba, neznámý symbol

syntaktická chyba, chybí středník

statická sém., nedeklarované b

sémant.dynamická, dělení nulou

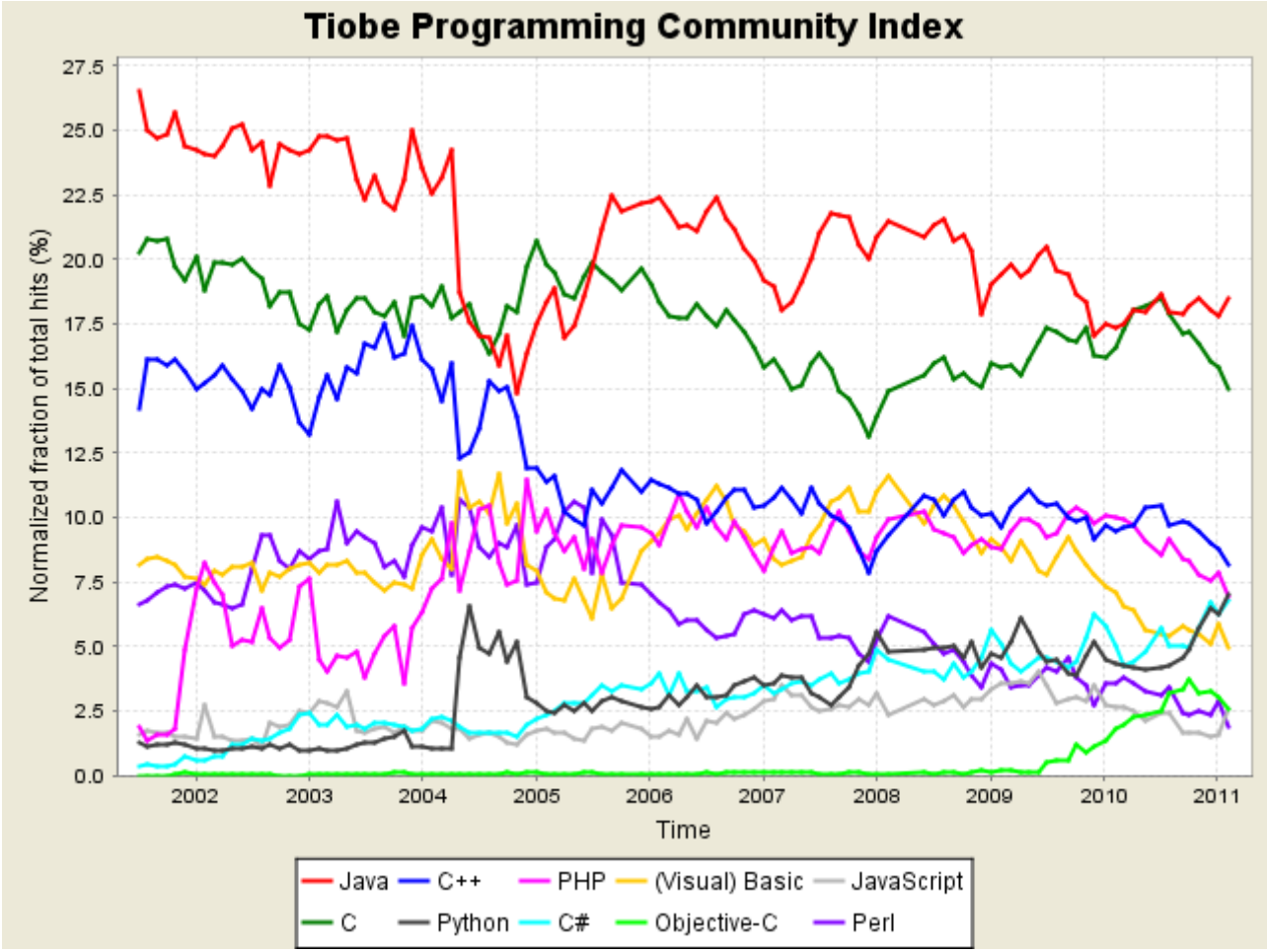
logická, má vracet a

Pořadí jazyků

Následující obrázek uvádí rating jazyků na základě frekvence dotazů z webu na jednotlivé jazyky

zdroj:

https://www.bitdegree.org/tutorials/most-used-programming-languages


Tiobe Programming Community Index
Normalized fraction of total hits (%)
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
Time
Java
C++
PHP
(Visual) Basic
JavaScript
C
Python
C#
Objective-C
Perl

Feb 2017	Feb 2016	Programming Language	Ratings	Change
1	1	Java	16.676%	-4.47%
2	2	C	8.445%	-7.15%
3	3	C++	5.429%	-1.48%
4	4	C#	4.902%	+0.50%
5	5	Python	4.043%	-0.14%
6	6	PHP	3.072%	+0.30%
7	9	JavaScript	2.872%	+0.67%
8	7	Visual Basic .NET	2.824%	+0.37%
9	10	Delphi/Object Pascal	2.479%	+0.32%
10	8	Perl	2.171%	-0.08%
11	11	Ruby	2.153%	+0.10%
12	16	Swift	2.125%	+0.75%
13	13	Assembly language	2.107%	+0.28%
14	38	Go	2.105%	+1.81%
15	17	R	1.922%	+0.73%
16	12	Visual Basic	1.875%	+0.02%
17	18	MATLAB	1.723%	+0.63%
18	19	PL/SQL	1.549%	+0.49%
19	14	Objective-C	1.536%	+0.13%
20	23	Scratch	1.500%	+0.71%

Zdroje

- Materialy PGS na Webu http://www.kiv.zcu.cz/~jezek_ka/vyuka/PGS
- **Základní:** Zakhour, Sharon. *Java 6 : výukový kurz*. Vyd. 1. Brno : Computer Press, 2007. ISBN 978-80-251-1575-6.
  Základní: Pilgrim, Mark. *Ponořme se do Python(u) 3*. Edice CZ.NIC, 2010. ISBN 978-80-904248-2-1.
  Doporučená: Sebesta, Robert W. *Concepts of Programming Languages*. 10. vydání. Addison Wesley, 2012. ISBN 978-0-13-139531-2.
  Doporučená: Bieliková, Mária; Návrat, Pavol. *Funkcionálne a logické programovanie*. Slovenská technická univerzita, 2009. ISBN 978/80/227/3225-3.
  Doporučená: Herout, Pavel. *Java a XML*. České Budějovice, 2007. ISBN 978-80-7232-307-4.
  Doporučená: Siebel, Peter. *Practical Common Lisp*. Apress, 2005. ISBN 978-1-59059-239-7.
  H.Schildt: Java2 Příručka programátora
- Ježek, Racek, Matějovič: Paralelní architektury a programy
- **Doporučená:** Scott, Michael L. *Programming Language Pragmatics*. Morgan Kaufmann, 2009. ISBN 9780123745149.
  Doporučená: *Stránky předmětu PGS (Portál ZČU)*
- **Doporučená:** Harms D., McDonald K. *Začínáme programovat v jazyce Python*. Computer Press, 2008.
- On-line katalogy knihoven