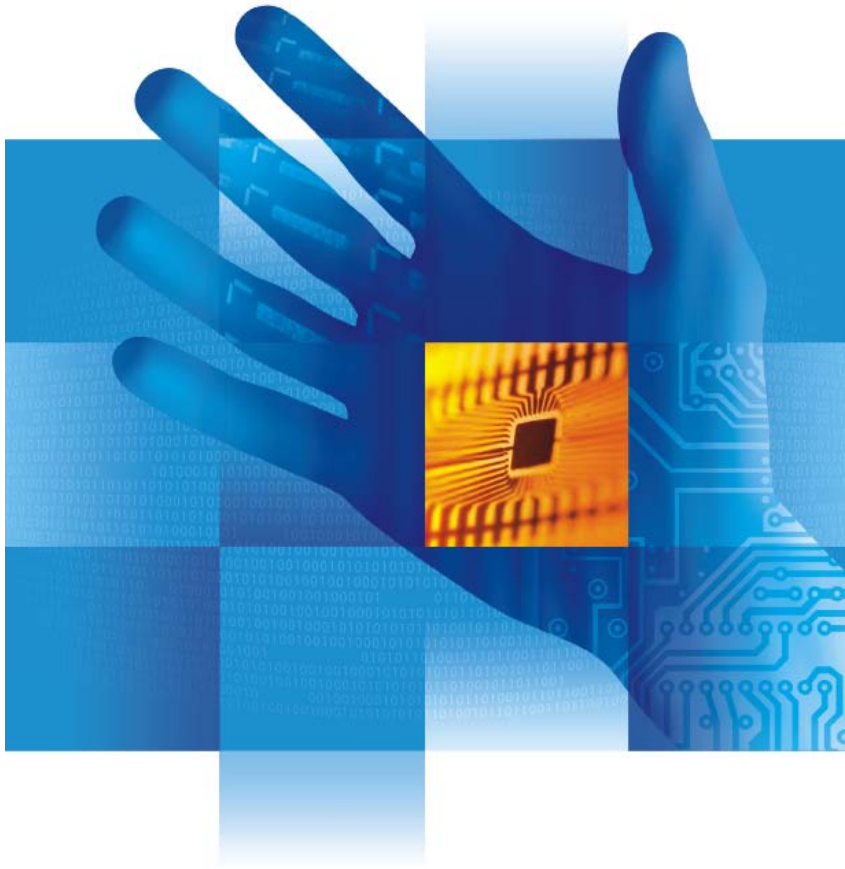
Základy programování

Programovací jazyky

doc. RNDr. Petr Šaloun, Ph.D.

VŠB-TUO, FEI

(přednáška připravena z podkladů Ing. Michala Radeckého)



Programovací jazyk

- Popis **výpočtů**, obvykle ve tvaru, jenž umožňuje provedení elektronickým počítačem (*program*)
- Standardizovaný nástroj pro komunikaci s počítačem
  - S jakými daty má počítač pracovat?
  - Jak se tato data budou ukládat a přenášet?
  - Které akce a kdy se mají provést?

Proč používáme programovací jazyky?

- **Zjednodušují přenos určitého typu informace**
  - Noty v hudbě
  - Matematické formule
  - Elektrotechnická schémata
- **Vyznačují se velkou přesností vyjádření**
  - Přirozené jazyky – vynechávání, gramatické chyby, víceznačnost
- **Jsou obvykle proveditelné na počítači**
  - Značkovací jazyky (HTML) – data, ne program
  - Specifikační jazyky (λ-kalkul) – teoretický výzkum

Úrovně jazyků

- Strojově orientované jazyky
  - instrukce, adresy
  - přímo závislé na konkrétním procesoru
  - náročný vývoj aplikací

- Jazyky vyšší úrovně
  - Delší doba překladu a generování cílového kódu
  - Větší nároky na čas a paměť v době běhu programu
  - Přenositelnost - podstatně menší závislost programů na konkrétním technickém vybavení
  - Čitelnost
  - Prostředky abstrakce - abstrakce dat a operací
  - Kontrola a detekce chyb - ještě před spuštěním programu

První překladače

Výraz "kompilátor"

- poč. 50. let - Grace Murray Hopper
- překlad jako „kompilace posloupnosti podprogramů z knihovny“
- „automatické programování“ - kompilace v dnešním slova smyslu, považovalo se za nemožné

1954-57 FORTRAN (FORmula TRANslator)

- John Backus, IBM
- problémově orientovaný, strojově značně nezávislý, optimalizace
- prokázala se vhodnost kompilovaných jazyků vysoké úrovně
- ad hoc struktury - komponenty a technologie se vymýšlely během výstavby překladače
- překladače se chápaly jako něco tajemného, složitého a drahého (**18 člověkoroků** - jeden z největších projektů v té době)

FORTRAN

```
C
C     Vypocet funkce faktorial
C
      INTEGER FUNCTION FACT(N)
      IMPLICIT NONE
      INTEGER N, I, F
      F = 1
      DO 10 I = 1,N
         F = F * I
10    CONTINUE
      FACT = F
      END

      PROGRAM P1
      IMPLICIT NONE
      INTEGER N, F, FACT
      READ(*,*) N
      F = FACT(N)
      WRITE(*,*) "Fact = ", F
      END
```

Vyšší jazyky

1958-59 LISP 1.5 (List Processing)

- John McCarthy, M. I. T.
- první funkcionální jazyk - implementace lambda-kalkulu
- možnost imperativního stylu programování

1958-60 ALGOL 60 (Algorithmic Language)

- J. Backus, P. Naur
- bloková struktura, složené příkazy, rekurze
- syntax poprvé popsána formálně *gramatikou* (BNF)
- koncem 60. let se stal nejpopulárnějším jazykem v Evropě
- základ mnoha dalších programovacích jazyků

Historie – ALGOL 60

```
begin
  integer N;
  ReadInt(N);

  begin
    real array Data[1:N];
    real sum, avg;
    integer i;
    sum:=0;
    for i:=1 step 1 until N do
      begin real val;
        ReadReal(val);
        Data[i]:=if val<0 then -val else val
      end;
    for i:=1 step 1 until N do
      sum:=sum + Data[i];
    avg:=sum/N;
    PrintReal(avg)
  end
end
```

Vyšší jazyky

1960 COBOL (Common Business Oriented Language)

- pro vytváření rozsáhlých programů k vládním a obchodním účelům
- formalizovaný anglický text, čitelný pro manažery
- zavedl propracované záznamové struktury

1964 BASIC (Beginners All-Purpose Symbolic Instruction Code)

- John G. Kemeny, Thomas E. Kurz, Dartmouth University
- 1975 Tiny BASIC běží na mikropočítači s 2KB RAM
- 1975 Bill Gates, Paul Allen prodávají firmě MITS

1963-64 PL/I (Programming Langugage I)

- kombinace jazyků COBOL, FORTRAN, ALGOL 60, snaha, aby obsahoval "všechno pro všechny" => příliš složitý
- zavedl konstrukce pro souběžné zpracování a výjimky

Vyšší jazyky

- Strukturované programování
  - **1968-71 Pascal**
    - Niklaus Wirth, ETH Zurich
    - jednoduchý jazyk, určen pro výuku programování
    - P-kód – instrukce virtuálního procesoru
    - specializované procesory pro P-kód
  - **1972 C**
    - Dennis Ritchie
    - určen částečně pro návrh přenositelných operačních systémů
- Modulární programování
  - **1980 Modula-2**
  - **1980-83 Ada**

Objektově orientované programování

1964-67 SIMULA 67

1972 Smalltalk

- Alan Kay, Xerox - původně pouze experimentální jazyk
- čistě objektově orientovaný - vše je objekt, předávání zpráv
- první jazyk podporující GUI s okny

1982-85 C++

- Bjarne Stroustrup, AT&T Bell Labs
- odvozen z jazyka C => mnoho nebezpečných vlastností, např. dynamické přidělování paměti bez GC, aritmetika s ukazateli
- 1997 ISO a ANSI standard

1984-85 Objective C

1994-95 Java

- James Gosling, Sun Microsystems
- původně pro vestavná zařízení, později široké použití v rámci WWW, strojově nezávislý binární kód (Java bytecode), použití just-in-time překlad

2000-02 C#

- Anders Hejlsberg, Microsoft, základní jazyk architektury .NET

Paradigmata programování

- Paradigma programování určuje *styl programování.*
  - Paradigma programování reprezentuje a definuje jakým způsobem vnímá programátor vykonávání programu.
  - Například objektově orientované paradigma programování definuje program jako kolekci komunikujících objektů.
  - Různé programovací jazyky implementují různá paradigmata programování.
    - Jazyk může být založen na jednom, ale klidně i na více paradigmatech.
    - Často není striktní – jazyk obsahuje více paradigmat a záleží na programátorovi, jaký kód vyprodukuje (strukturovaný program v C++).
    - Jednotlivé paradigmata mají většinou určité specifické vlastnosti.
- Existuje celá řada paradigmat pro programování
- Vznikají stále nové

Paradigmata programování

- Annotative programming (as in Flare language)
- Aspect-oriented programming (as in AspectJ)
- Attribute-oriented programming (might be the same as annotative programming) (as in Java 5 Annotations, pre-processed by the XDoclet class; C# Attributes)
- Class-based programming, compared to Prototype-based programming (within the context of object-oriented programming]
- Concept-oriented programming is based on using concepts as the main programming construct.
- Constraint programming, compared to Logic programming
- Data-directed programming
- Dataflow programming (as in Spreadsheets)
- Flow-driven programming, compared to Event-driven programming
- Functional programming
- Imperative programming, compared to Declarative programming
- Intentional Programming
- Logic programming (as in Mathematica)
- Message passing programming, compared to Imperative programming
- Object-Oriented Programming (as in Smalltalk)
- Pipeline Programming (as in the UNIX command line)
- Policy-based programming
- Procedural programming, compared to Functional programming
- Process oriented programming a parallel programming model.
- Recursive programming, compared to Iterative programming
- Reflective programming
- Scalar programming, compared to Array programming
- Component-oriented programming (as in OLE)
- Structured programming, compared to Unstructured programming
- Subject-oriented programming
- Tree programming
- Value-level programming, compared to Function-level programming

Imperativní a deklarativní jazyky

- Imperativní jazyky
  - Program má implicitní stav, který se modifikuje konstrukcemi programovacího jazyka.
  - Explicitní pojem „*pořadí*“ příkazů
    - Vyjadřuje, jak se má program vyhodnocovat
  - Vychází z aktuální (Von Neummanovy) architektury počítačů
    - Jednoduchá a efektivní realizace
- Deklarativní jazyky
  - Program nemá implicitní stav.
  - Program je tvořen výrazy, ne příkazy.
  - Popisujeme co se má spočítat, ne jak.
    - Není dáno pořadí příkazu.
  - Efektivní implementace vyžaduje komplexní optimalizace.

Klasifikace jazyků

- **Imperativní** - programy jsou posloupnosti základních příkazů (nejčastěji přiřazení) s odpovídajícími řídicími strukturami (např. cykly), jež určují, které příkazy se budou provádět a *v jakém pořadí*.
  C, Pascal, Fortran, JSI
- **Objektově orientované** - programy jsou kolekcí komunikujících objektů. Často se s těmito jazyky spojuje dědičnost a polymorfismus.
  Simula, Smalltalk-80, C++, Java, C#

Klasifikace jazyků

- **Deklarativní jazyky** - popisujeme, co se má vypočítat, ne jak.
- **Logické** - programy jsou kolekcemi tvrzení
  v konkrétní logice (nejčastěji predikátové)
  Prolog, Goedel
- **Funkcionální** - programy se popisují jako soustavy rovností s aplikacemi funkcí na hodnoty.

```
(define (faktorial n)
  (if (= n 0)
    1
    (* n (faktorial (- n 1))))))

(faktorial 5)
```

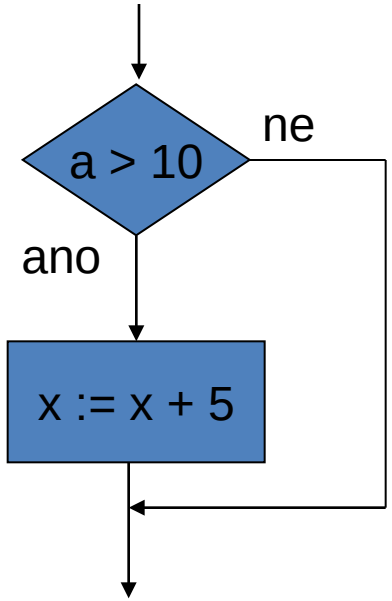
```
termostat:- repeat,
                  write('Je topeni zapnute(a/n/k)?
                        <k..konec programu>'),
read(T), (T='k',!; write('Zadejte teplotu:'),
read(X), akce(X,Akce,T),
nl,write(Akce), nl, fail).


akce(X,' Vypnete topeni!',T):- X>25, T='a', !.
akce(X,' Zapnete topeni!',T):- X<20, not(T='a'), !.
akce(X,' Odpocivejte, neni potreba nic delat.',T).
```

Specifikace programovacích jazyků

- **Jak má vypadat správně napsaný program?**
  - SYNTAXE
  - Formální jazyky, gramatiky, automaty
- **Co má program dělat?**
  - SÉMANTIKA
  - Predikátový počet, lambda kalkul, atributové gramatiky

Specifikace programovacích jazyků

- **Syntaxe** - struktura jazykových konstrukcí
  - Textové jazyky (C, Pascal, Java)
  - Grafické jazyky (vývojové diagramy, UML)




```
if  a > 10  then begin
      x := x + 5;
end
```

Specifikace programovacích jazyků

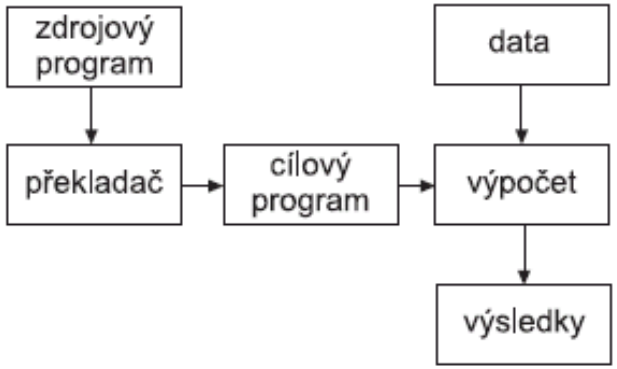
- **Sémantika** – význam jazykových konstrukcí
  - Statická sémantika – v době překladu
  - Dynamická sémantika – v době běhu
- Příklad: Co znamená X + 1 ?
  - X je celé číslo: *Přičti k hodnotě proměnné X jedničku.*
  - X je řetězec: *Převeď konstantu 1 na řetězec a připoj na konec řetězce uloženého v proměnné X.*
  - X je objekt: *Zavolej metodu "operator +“ s parametrem 1.*
  - X je reálné číslo: *Převeď 1 na reálné číslo 1.0 a přičti k hodnotě proměnné X.*

Překladač

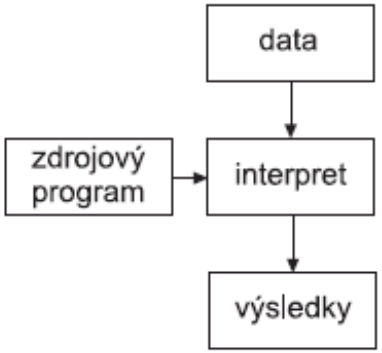
- *Analýza* zdrojového textu, vyhledání chyb
  - Základní stavební prvky – identifikátory, čísla, řetězce, operátory, oddělovače, …
  - Programové konstrukce – deklarace, příkazy, výrazy
  - Kontextové vazby – definice/užití, datové typy
- *Syntéza* cílového programu / *Interpretace*
  - Strojový jazyk (nebo JSI)
  - Jazyk virtuálního procesoru (JVM, CLR)

Typy překladače

- Kompilační překladač



- Interpretační překladač



Typy překladače

- Interpret je mnohem pomalejší než kompilátor
  - Je potřeba analyzovat zdrojový příkaz pokaždé, když na něj program narazí.
  - Interpret je 10 x až 100 x pomalejší.
- Interpret má ale i výhody.
  - Nezávislost na platformě.
  - Snadnější ladění – v době chyby máme k dispozici všechny informace.
  - Možnost měnit program za běhu - Smaltalk.

Typy překladače

- Inkrementální překlad
  - Umožňuje po drobné opravě přeložit jen změněnou část
  - Možnost provádění drobných změn během ladění programu
- Just-in-time překlad
  - Generování instrukcí virtuálního procesoru (Java VM - .class, .NET CLR – jazyk IL)
  - Překlad až v okamžiku volání podprogramu
  - Optimalizace podle konkrétního procesoru

Překlad C++


zdrojový text
knihovní deklarace
překladač
objektový kód
knihovní defini
spojovací program
proveditelný kód

Smalltalk

- Existují i jiné objektově orientované jazyky než Java nebo C++…
- "When I invented the term 'object-oriented' I did not have C++ in mind." -- Alan Kay

- Jedním z „jiných“ objektově orientovaných jazyků je jazyk Smalltalk
  - Ve skutečnosti neexistuje jazyk Smalltalk. Existuje celá řada „variant“ jazyků obsahujících Smalltalk v jejich názvu.
  - Obvykle je pod pojemem Smalltalk rozuměn jazyk Smalltalk-80.

Smalltalk

- Smalltalk je *čistě* objektově orientovaný jazyk.
  - Koncepce tříd a objektů
  - Základní myšlenkou je, že vše je objekt a objekty spolu komunikují prostřednictvím zpráv
- Objekty mohou v jazyce Smalltalk provádět právě tři činnosti
  - Udržovat stav (reference na další objekty)
  - Přijímat zprávy od sebe a nebo od jiných objektů
  - V rámci reakce na zprávu posílat zprávy jiným objektům.
- Vše v jazyce Smalltalk je objekt.
  - Každý objekt je instancí nějaké třídy. Třídy jsou také objekty.
  - Každá třída je instancí nějaké *metatřídy*.
  - Metatřídy jsou všechny instancí třídy `Metaclass`.
  - Blok zdrojového kódu je taky objekt
    - Například tělo metody – zprávy

Smalltalk

- Jazyk Smaltalk integruje kompletní mechanismus reflexe.
- Obvykle realizuje „image-based persistence“
  - Prostředí pro jazyky jako Java oddělují zdrojový kód od stavu programu.
    - Zdrojový kód je nahrán při startu aplikace.
    - Po ukončení jsou ztraceny všechny data kromě těch, které byly explicitně uloženy.
  - V jazyce Smalltalk je vše objektem, tedy například i třídy, a vše je uloženo jako jeden „image“.
  - Ten může být snadno „obnoven“.
  - Vývojové prostředí je obvykle součástí prostředí. Není využíván žádný „e

```
result := a > b
        ifTrue: [ 'promenna a ma vetsi hodnotu' ]
        ifFalse: [ 'promenna a ma mensi hodnotu' ]

hello
        Transcript show: 'Hello, World!'
```

Co je to JavaScript

Zdroj: http://petr.vaclavek.com

- Skriptovací programovací jazyk (interpretovaný, zpracovává se přímo zdrojový kód) určený pro řešení dynamiky WWW stránek na straně klienta.
- Vlastnosti
  - Součást zdrojového kódu HTML
  - Multiplatformní
  - Závislý na interpretačním prostředí (prohlížeči)
  - Objektově orientovaný, ale beztřídní (prototypy)
  - Case-senzitivní
  - Syntaxí podobný jazykům typu C/C++/Java (někdy volnější, např. ;)
  - Beztypový
  - Není to Java

Konstrukce JavaScriptu

```
document.write(“Ahoj“);
document.write("Tohle 'jsou' uvozovky");
document.write("Tohle \"jsou\" uvozovky“ + “ – zase”);
```

```
var p1 = 10;
var p2 = “10.5”;
p3 = “ahoj”;
var p4 = true;
document.write(p1 + p2); //1010.5
p2 = 10.5;
document.write(p1 + p2); //20.5
```

```
var pole2 = ["mrkev", "brambory", "kvetak"] //std. jednorozměrné
for(i=0;i<pole2.length;i++){
   document.write(pole2[i] +" ")
}

pole2[“br”] = “brambory”;

var pole = new Array("HTML", "DHTML", "XHTML");
document.write(pole.valueOf()); //HTML,XHTML,XHTML
document.write(pole.toSource()); //"["HTML", "DHTML", "XHTML"]"
```