LISP - "LISt Processing"

dr inż. Tomasz Białaszewski

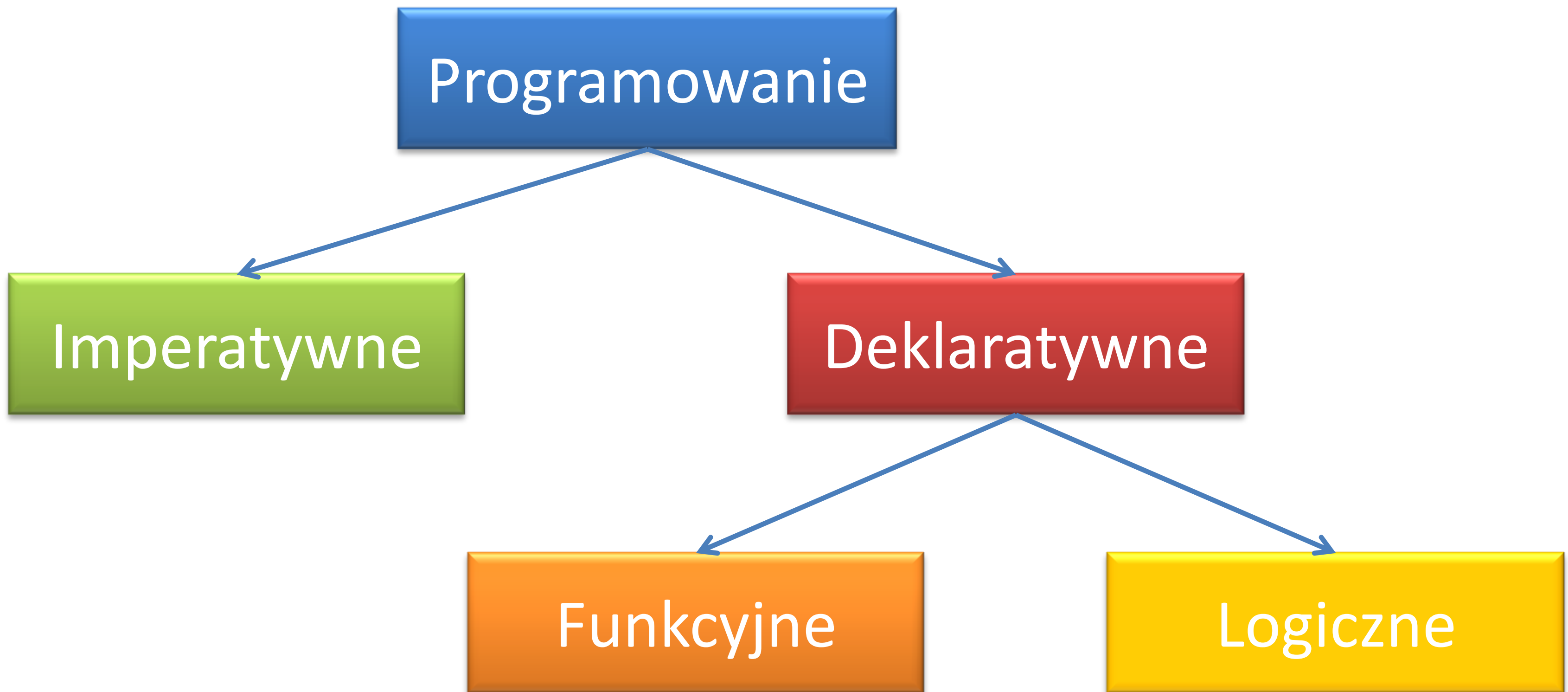
Katedra Systemów Decyzyjnych

Email: bialas@eti.pg.gda.pl

http://www.eti.pg.gda.pl/katedry/ksd/pracownicy/Tomasz.Bialaszewski/

Wprowadzenie

Rodzaje języków programowania



Wprowadzenie

Programowanie imperatywne:

- silnie związane z architekturą komputera (model von Neuman'a)
- obliczenia w zakresie zmiany stanu programu
- sekwencja rozkazów do wykonania przez komputer
- np. FORTRAN, ALGOL, COBOL, BASIC, Pascal, C/C++, JAVA, PHP

Wprowadzenie

Programowanie deklaratywne:

- co program powinien wykonać bez opisywania jak to ma zrobić
- związki z logiką matematyczną
- brak efektów ubocznych
- funkcyjne i logiczne programowanie

Wprowadzenie

Programowanie logiczne:

- wnioskowanie logiczne jako deklaratywna reprezentacja języka programowania
- **Prolog** (**PRO**gramowanie w **LOG**ice) – język programowania stosowany w problemach sztucznej inteligencji
- program wyrażony za pomocą termów, relacji reprezentowanych jako fakty i reguły

Wprowadzenie

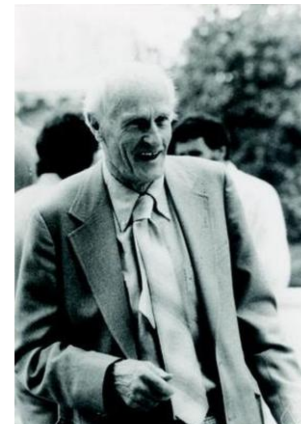
Programowanie funkcyjne:

- przeprowadzanie obliczeń na matematycznych funkcjach
- unikanie stanu i zmiennych programu
- **rachunek lambda** (rachunek-λ) - formalny system zaprojektowany do badań związanych z funkcjami matematycznymi i rekurencją

Wprowadzenie

rachunek-λ (A. Church & S. C. Kleene, 1930):

- mocny i elegancki model obliczeń matematycznych
- Abstrakcyjny model obliczeń (kuzyn maszyny Turinga)
- pozbawiony stanu



Wprowadzenie (rachunek-λ)

- brak skutków ubocznych (stanu)
- brak zmiany wartości argumentów funkcji
- posiada trzy konstrukcje:
  - definicja
  - zadanie
  - aplikacja
- dwa specjalne operatory
  - symbol ‘λ’
  - znak ‘.’

Wprowadzenie (rachunek-λ)

Definicja funkcji: λ x . Body

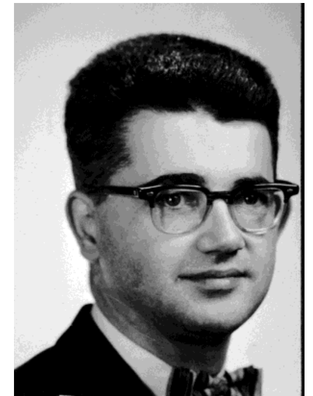
Funkcja dla która pobiera pojedynczy argument, oblicza wyrażenie zawarte w ciele funkcji i zwraca wynik

Przykład. Zdefiniowanie powójnej wartości x i dodanie liczby 3:

$$\lambda x . 2*x+3$$

Wprowadzenie

LISP - **LISt Processing** (John McCarthy, 1958):



- rodzina języków programowania
- składnia języka zawarta w nawiasów
- drugi najstarszy język programowania
- Lista – główna struktura danych
- kod programu reprezentowany przez listę

Wprowadzenie (LISP)

- praktyczny matematyczny opis programu w rachunku lambda
- język programowania stosowany w sztucznej inteligencji:
  - struktury drzewiaste
  - automatyczne zarządzanie pamięcią
  - dynamiczne typowanie
  - kompilacja własnego kodu źródłowego

Wprowadzenie (LISP)

Kod programu LISP-owego:

- struktura danych
- s-wyrażenie (lista otoczona nawiasami)
- lista z operatorem funkcji jako pierwszy element, zaś kolejne stanowią jej argumenty

(f arg1 arg2 arg3)

Wprowadzenie (LISP)

- Pierwsza implementacja:
  - Steve Russell (IBM 704)
- LISP interpreter – obliczanie wyrażeń LISP-a
- Dwa podstawowe makra języka – podstawowe operacje na listach:
  - **car** (**C**ontents of the **A**ddress part of **R**egister number)
  - **cdr** (**C**ontents of the **D**ecrement part of **R**egister number)

Wprowadzenie (LISP)

- wyrażenie „Register” określa „Memory Register” („Memory Location”)
- dialekty Lispu nadal używają operacji **car** i **cdr** (wymowa: /ˈkɑr/ i /ˈkʊdər/)
- rezultaty operacji:
  - **car** zwraca pierwszy element listy (głowa)
  - **cdr** zwraca resztę listy (ogon)

Wprowadzenie (LISP)

- Trudny do implementacji kompilator z zastosowaniem techniki lat 70
- Procedury zwalniania pamięci (Daniel Edwards):
  - praktyczna implementacja LISP-a na komputerach ogólnego przeznaczenia
  - problem efektywności
- Maszyny LISP-u:
  - dedykowany sprzęt do uruchamiania programów w środowisku LISP

Wprowadzenie

- Maszyny LISP-u pionierem w wprowadzaniu obecnych technologii:
  - procedury zwalniania pamięci
  - laserowe drukowanie
  - okienkowe systemy
  - mysz komputerowa
  - procedury grafiki wysokiej rozdzielczości
  - mechanizmy renderowania

Wprowadzenie

- Dialekty LISPa – odmiany podstawowego jądra języka:
  - Scheme
  - Common Lisp
  - Clojure
- Zastosowania jako język skryptowy:
  - Emacs Lisp w edytorze Emacs-a
  - Visual Lisp w AutoCAD-zie
  - Nyquist w Audacity

Wprowadzenie

Common LISP:

- następca MacLisp-a
- duży standard języka zawierający wiele wbudowanych typów danych, funkcji i makr
- zorientowany obiektowo (Common LISP Object System, CLOS)
- zapożyczone funkcje ze Scheme-a:
  - zakres leksykalny
  - zamknięcia leksykalne

Wprowadzenie

Clojure:

- dynamiczny dialekt LISP-a uruchamiany na wirtualnej maszynie JAVY
- język skryptowy do programowania wielowątkowego
- uruchamiany przez interpreter lub kompilowany do kodu bajtowego
- pełna integracja z językiem JAVA

Wprowadzenie

Scheme (1975, Gerald Sussman & Guy Steele Jr.)

- Pierwszy dialekt LISPa zawierający:
  - zakres leksykalny
  - procedury klasy pierwszej
  - kontynuacje
- Pierwsza forma języka - przeznaczona do badań naukowych i dydaktyki
- Wspieranie tylko kilku zdefiniowanych form składniowych języka oraz procedur

Wprowadzenie (Scheme)

- Pierwsze implementacje oparte na interpreterze – bardzo wolne
- Obecne kompilatory Scheme-a generują kod na równi z kodem dla języka C i Fortran
- Impementacyjne cechy:
  - rekurencja ogonowa
  - pełna kontynuacja (niekoniecznie występuje Common LISPie)

Wprowadzenie (Scheme)

- Jasna i prosta semantyka
- Kilka różnych sposobów zapisywania wyrażeń
- Szeroki wybór modeli programowania:
  - imperatywny
  - funkcyjny
  - style przekazywania komunikatów

Wprowadzenie (Scheme)

- Interpreter szczególnie popularny w językach skryptowych:
  - SIOD oraz TinyScheme w GIMPie („Script-fu“)
  - LIBREP (na podstawie Emacs Lisp) w menadżerze Sawfish
  - Interpreter Guile zastosowany w GnuCash

Składnia Scheme-a

Program w języku Scheme może zawierać:

- słowa kluczowe
- zmienne
- formy struktury
- stałe (liczby, znaki, łańcuchy, cytaty wektorów, list lub symboli itp.)
- znaki spacji
- komentarze

Składnia Scheme-a

Identyfikatory (słowa kluczowe, etykiety, zmienne, i symbole) – konstruowane ze zbioru:

- małych liter od `a` do `z`
- dużych liter od `A` do `Z`
- cyfry od `0` do `9`
- znaków `? ! . + - * / < = > : $ % ^ & _ ~ @`

Składnia Scheme-a

Identyfikatory nie mogą zaczynać się od:

- `@`
- jakiegokolwiek znaku zaczynającego liczbę tzn.:
  - cyfra
  - znak (+)
  - znak (−)
  - kropka (.)

***Wyjątek**: +, -, ..., są identyfikatorami*

***Np**: hi, Hello, x, x3, ?$&*!!!*

Składnia Scheme-a

Identyfikatory muszą być ograniczone:

- znakiem spacji
- nawiasami
- znakiem podwójnego cytatu (`"`)
- znakiem komentarza (`;`)

Uwaga: wszystkie implementacje muszą rozpoznawać dowolny ciąg identyfikatorów zgodnie z powyższymi regułami

Składnia Scheme-a

- Brak ograniczeń na długość identyfikatora
- Identyfikatory mogą być zapisane w dowolnej kombinacji małych i dużych liter
- Wielkość nie jest ważna
- Dwa identyfikatory różniące się wielkością liter są identyczne

Przykładowo: `abcde`, `Abcde`, `AbCdE`, `ABCDE` *są tymi samymi identyfikatorami*

Składnia Scheme-a

- System drukuje identyfikatory zwykle albo z małych albo z wielkich liter niezależnie od charakteru wprowadzanych znaków
- Struktury danych są zamknięte okrągłymi nawiasami, np. :

```
(a b c)
(* (- x 2) y)
```

- Pusta lista ()

Składnia Scheme-a

- Niektóre implementacje pozwalają na stosowanie nawiasów kwadratowych `[]` w miejsce okrągłych w celu polepszenia czytelności kodu
- Wartości logiczne reprezentowane przez identyfikatory `#t` and `#f`
- Wyrażenia warunkowe Scheme-a
  - `#f` traktowane jako fałsz
  - Wszystkie inne obiekty są prawdą np.: `3`, `()`, `"false"`, oraz `nil` przyjmują wartość prawdy

Składnia Scheme-a

- Wektory - `#(a vector of symbols)`
- Łańcuchy - `"This is a string"`
- Znaki rozpoczynające się przez `#\`, np., `#\a`
- Wielkość ma znaczenie dla znaków i łańcuchów, w odróżnieniu od identyfikatorów
- Liczby: – `12, 1/2, 1.4, 1e2456,`

  `1.3777-2.7i, -1.2@73`

Składnia Scheme-a

- Wyrażenia mogą rozciąga się na kilka linii – brak wymaganego znaku kończącego
- Liczba białych znaków (spacji) oraz końca linii nie jest znacząca
- Programy są zwykle wcięte, aby pokazać w sposób czytelny struktury kodu

Składnia Scheme-a

Komentarze:

- pomiędzy znakiem (;) końcem linii
- umieszczane na tym samym poziomie wcięcia co wyrażenie
- wyjaśnianie procedur umieszczane przed nimi bez wcięcia

Przykładowo:

```
;;; The following procedures
```

Konwencje w nazewnictwie

- Nazwy predykatów zakończone znakiem (`?`)

  Przykładowo: `eq?`, `zero?`, and `string?`

- Typy predykatów tworzone z ich nazw

- Nazwy procedur rozpoczynające się do prefiksu `char-`, `string-`, `vector-` , `list-`

  np. : `string-append`

Interakcja Scheme-a

- System interakcji oparty na zasadzie „wczytaj-wyznacz-wydrukuj” („read-evaluate-print”, REPL)
- System interakcji Scheme-a:
  - czyta każde wyrażenie wprowadzone z klawiatury,
  - ocenia je a następnie
  - drukuje jego wartość

Proste wyrażenia

- Liczby są stałymi:

```
12345678784978 ⇒ 12345678784978
3/4            ⇒ 3/4
3.141592653    ⇒ 3.141592653
2.2+1.1i       ⇒ 2.2+1.1i
7/2            ⇒ 3½
1.14e-10       ⇒ 1.14e-010
```

Proste wyrażenia

- Procedury arytmetyczne +, -, *, /

```
(+ 1/2 1.2 3/4)   ⇒  2.45
(- 1/2 5/6 3/4)   ⇒  -1 11/12
(* 3 -2/3 5/4)    ⇒  -2 1/2
(/ 3 -2.5 5/4)    ⇒  -0.96
```

- Scheme stosuje notację prefiksową

(procedure arg1 arg2 … arg*n*)

Proste wyrażenia

- Zagnieżdżone procedury

```
(+(+ 2 2)(+ 2 2))      ⇒  8

(- 2 (* 4 1/3))        ⇒  2/3

(* 2 (* 2 (* 2 (* 2 2)))) ⇒  32

(/(* 6/7 7/2)(- 4.5 1.5)) ⇒  1.0
```

Proste wyrażenia

Lista:

- Podstawowa struktura danych
- Sekwencja obiektów otoczona (…) np.:

```
(1 2 3 4 5)      ;lista liczb
("a" "list")     ;lista łańcuchów
(4.2 "hi")       ;lista
((1 2) (3 4))    ;zagnieżdżona lista
```

Proste wyrażenia

Pytanie:
Jak Scheme odróżnia listę obiektów

$(obj_1\ obj_2\ \ldots)$

od stosowania procedury

(*procedure arg* ...)?

Odpowiedź:
Musimy wyraźnie podpowiedzieć systemowi, że będziemy traktować listę jako strukturę danych a nie jako aplikację procedury

Proste wyrażenia

- Wymuszenie cytowania za pomocą formy specjalnej - `quote`

```
(quote (1 2 3 4))   ⇒ (1 2 3 4)
(quote ("a" "bb")) ⇒("a" "bb")
(quote (+ 3 4))     ⇒(+ 3 4)
```

- Znak pojedynczego cudzysłowia (�)

```
'(1 2 3 4)          ⇒(1 2 3 4)
'((1 2)(3 4))      ⇒((1 2)(3 4))
'(/ (* 2 -1) 3) ⇒(/ (* 2 -1) 3)
```

Proste wyrażenia

- Wyrażenie `quote`:
  - nie jest aplikacją procedury
  - zatrzymuje ocenę wyrażenia
  - jest inną formą składni Scheme-a
- Scheme obsługuje kilka innych form składni
- Każda z form składni jest inaczej traktowna

Proste wyrażenia

Przykład

`(quote hello)` ⇒ `hello`

- Symbol `hello` musi być cytowany w celu zapobieżenia traktowania `hello` jako zmiennej
- Symbole i zmienne w Scheme-ie są podobne do symboli i zmiennych w matematycznych wyrażeniach i równaniach

Proste wyrażenia

- Cytując listę - Scheme traktuje formę w nawiasach jako listę niż jako zastosowanie procedury
- Cytując identyfikator - Scheme traktuje identyfikator jako symbol a nie zmienną
- Symbole są zwykle stosowane do:
  - reprezentowania zmiennych w sposób symboliczny w równaniach lub programach
  - jako słowa reprezentujące wyrażenia języka naturalnego

Proste wyrażenia

- Procedury Scheme-a do manipulacji listami:
  - `car` (pierwszy element listy - głowa)
  - `cdr` (reszta listy - ogon)

  Procedury te wymagą niepustej listy jako argumentu
- Przykładowo:

```
(car '(a b c))       ⇒   a
(cdr '(a b c))       ⇒   (b c)
(cdr '(a))           ⇒   ()
```

Proste wyrażenia

- Inne przykłady:

```
(car (cdr '(a b c)))    ⇒   b
(cdr (cdr '(a b c)))    ⇒   (c)
(car '((a b) (c d)))    ⇒   (a b)
(cdr '((a b) (c d)))    ⇒   ((c d))
```

Proste wyrażenia

- Procedura do konstruowania listy – `cons`:
  - wymaga dwóch argumentów
  - dodaje element na początek listy
  - określane jako połącz elementy w listę

```
(cons 'a '())                ⇒  (a)
(cons 'a '(b c))             ⇒  (a b c)
(cons 'a (cons 'b ()))       ⇒  (a b)
(cons '(a b) '(c d))     ⇒  ((a b) c d)
```

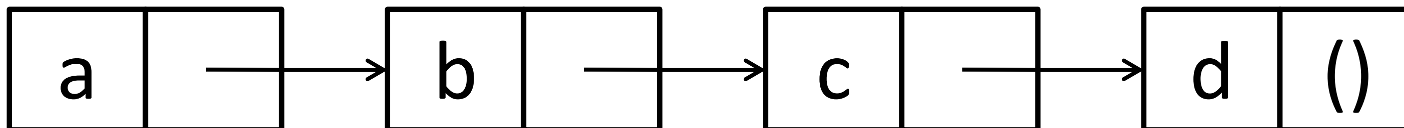
Proste wyrażenia

- Przykładowe wywołania procedury `cons`:

```
(car (cons 'a '(b c))) ⇒ a

(cdr (cons 'a '(b c))) ⇒ (b c)

(cons (car '(a b c))
      (cdr '(d e f)))  ⇒ (a e f)

(cons (car '(a b c))
      (cdr '(a b c)))  ⇒ (a b c)
```

Proste wyrażenia

- Uwagi do `cons`:
  - procedura `cons` tworzy *parę*
  - `cdr` z pary nie musi być listą
- Lista to sekwencja par
- Każdy ogon z pary jest następnym elementem pary w sekwencji



Proste wyrażenia

- ***Lista właściwa***:
  - `cdr` z ostatniej pary w liście jest listą pustą
  - lista pusta jest listą właściwą
  - każda para, dla której ogon jest lista właściwą to również jest to właściwa lista
- ***Lista niewłaściwa:***
  - drukowana w notacji kropkowane-pary

```
(cons 'a 'b)         ⇒  (a . b)
(cdr '(a . b))       ⇒   b
(cons 'a '(b . c))   ⇒  (a b . c)
```

Proste wyrażenia

- ***Kropkowana para*** - reszta (ogon) nie jest listą
- Lista właściwa zawsze drukowana bez kropek

```
'(a . (b . (c . ())))   ⇒   (a b c)
```

- Procedura `list` (podobna do `cons`)
  - wywoływana z dowolną liczbą parametrów
  - zawsze tworzy listę właściwą

```
(list 'a 'b 'c)      ⇒   (a b c)
(list 'a)            ⇒   (a)
(list)               ⇒   ()
```