Проектиране и Тестиране на Софтуер
ТУ, кат. КС, летен семестър 2012

Лекция 52

Тема:

Функционално Програмиране (Лисп)

Съдържание:

- Стилове на Програмиране
- Функционално Програмиране
- Част 1
- Основни понятия
- Част 2
- Въведение в Лисп

Стилове на Програмиране

Дейностите в програмирането не се подчиняват на единни водещи принципи. Съществуват стилове на програмиране, които могат да се класифицират така:

- Процедурно програмиране (procedure based programming);
- Логическо програмиране (logic programming);
- Функционално програмиране (functional programming).

Какво е ФП?

ФП е стил на програмиране, при който:

1/ писане на функционална програма означава дефиниране на една или няколко функции;

2/ единственото действие по време на изпълнение е извикване на функция.

Фактори за развитие на ФП

- λ-смятамне (А.Чърч, A. Church)
- Езикът LISP (J.McCarthy)
- John Backus и статията му:

"Can Programming be Liberated from the von Neumann Style? A Functional Style and its Algebra of Programs.",

Com of the ACM, vol21, no8, pp613-641

ФП

Лисп е езикът за функционално програмиране

LISP – LISt Programming.

Създаден е с цел обработка на списъчни структури данни и потенциално приложение в областта на AI, expert systems, knowledge based systems.

LISP – J.McCarthy, MIT, 1958

Lisp има два основни наследника:

Scheme–Sussman, MIT, 1975; **Common Lisp,** 1984

Сродни езици: **ML, LCF, Miranda, Haskell**

Лисп история

Създадена спецификация през 1958.

Най-разпространени диалекти: Common Lisp и Scheme.

Lisp е създаден като средство за означаване в компютърни програми на Alonzo Church's lambda calculus.

Името идва от "List Processing". Linked lists са основни data structures за езика, като и първичният Lisp source code се представя в списъчна форма. По този начин Lisp програми манипулират source code като data structure.

Общата форма на код и данни дават специфика на Лисп синтаксиса. Програмите се пишат като заградени в скоби списъци (parenthesized lists) или *s-expressions*. Извикване на функция се записва като списък с начало име на ф-ия и следвано от аргументи: (f x y z).

Лисп: 1958-1962

- Lisp е създаден през 1958 в MIT. McCarthy публикува в Communications of the ACM през 1960 статия "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I". (Part II не е публикувана.) Авторът показва, че с няколко примитивни операции и функцинална нотация, е възможно да се създаде Turing-complete language for algorithms.
- Lisp е реализиран първо от Steve Russell на IBM 704 computer. Russell демонстрира, че функцията за оценяване на изрази *eval* може да бъде реализирана като Lisp interpreter.

Лисп история

Information Processing Language (IPL) е първият ПЕ, поддържащ работа със списъци и рекурсия, като Lisp.

McCarthy's въвежда своя нотация скобков "M-израз“, която се превежда в S-изрази. Например, M-изразът car[cons[A,B]] е напълно еквивалентен на S-израза (car (cons A B)). След реализацията на Лисп, като практически по-удобни са възприети S-изразите, докато употребата на M-изрази отпада.

Два асемблерски макроса за IBM 704 са приети като примитиви за декомпозиране на списъци: car (Contents of Address Register) and cdr (Contents of Decrement Register). Lisp диалектите още ползват car and cdr (pronounced: [kar] and ['kədər]) за операции връщат началото на списък и остатъка на списък.

Функционално програмиране

А.П.Ершов: ФП е специфична техника на програмиране, при която съставянето на програми е свързано само с дефиниране на функции. Единственото действие, при изпълнение на една функционална програма, е обръщение към функции.

Още: ФП е програмиране без разпределение на памет, без присвояване, без цикли, без блоксхеми и без предаване на управлението. Изброените действия се поемат от практическите реализации на системите за ФП.

Част 1

Основни понятия

Съдържание:

- Концепцията Функция
  - Прости функции
  - Функционални форми
- Основни принципи на ФП
- Оценяване на изрази

Functional Programming

COMPOSITION OF FUNCTIONS

=

PROGRAMS

Математически функции

- Математическа Функция е съответствие м/у елементите на едно множество, domain set, и елементите на друго множество, range set.
- Прости функции (Simple functions)
- Функции от по-висок ред - Higher order functions (functional forms):

Основни принципи на ФП

За резултатно ФП е необходимо:

- Фиксирано множество от базови (base, primitive, standard, generic) функции;
- Механизъм за конструиране на нови функции от съществуващото базово множество функции.

Примери

Дадена е базова функция: max(x,y)

Задача: Да се композира функция, която намира най-големия от три параметъра

Решение:

greatest(a,b,c) ≡ max(a, max(b,c)) ≡
max(max(a,b), c)

Примери

Дадени са базови ф-ии: max(x,y),
greatest(x,y,z)

Задача: Да се композира функция, която намира най-големия от шест параметъра

Решение:

max(greatest(a,b,c), greatest(d,e,f))

greatest(max(a,b), max(c,d), max(e,f))

И така, за ФП са необходими:

1/ базово множество функции;

2/ механизъм за композиция на функции;

3/ нищо друго.

Изисквания към базовите функции:

- ефективна машинна реализация;
- леснота за ползване.

Пример

Базовото множество следва да включва функции за аритметчни операции, като

add(x,y) ≡ x + y sub(x,y) ≡ x - y

mul(x,y) ≡ x * y div(x,y) ≡ x / y

С употребата на тези функции е възможно произволен АИ да се представи във функционален запис

a+b*c

(2*x + 3*y) / (x-4)

Пример

add(x,y) ≡ x + y

sub(x,y) ≡ x - y

mul(x,y) ≡ x * y

div(x,y) ≡ x / y

a+b*c се трансформира в

add(a, mul(b,c))

Пример

add(x,y) ≡ x + y

sub(x,y) ≡ x - y

mul(x,y) ≡ x * y

div(x,y) ≡ x / y

(2*x + 3*y) / (x-4) се трансформира в

div(add(mul(2,x), mul(3,y)) , sub(x,4))

Израз, представен във функционален запис, се нарича ***expression with applicative structure.***

Всеки израз има два компонента:

операнд операция

Операндът означава стойност.

Операцията означава функция.

ФП борави с *applicativity* и *functionality*.

Три термина са от значение:

FUNCTION

APPLICATION

EXPRESSION EVALUATION

Summary

Най-общо, една ФП програма е колекция от извиквания на функции.

Да се изпълни ФП програма означава да се оценят съответните извиквания на функции.

Оценката (Evaluation) включва:

a/ оценка на аргументите на функцията;

b/ приложение на функцията върху оценените аргументи.

Част 2

Въведение в Лисп

Съдържание

- Списъци
- Безкраен цикъл на Лисп машина:

  Read, Evaluate, Write

  Read-Eval-Print Loop or REPL
- Описание на потребителски дефинирани имена и функции
- Обработка на списъци в Лисп

Лисп и диалекти

Най-напред: какво е списък.

List е основна структура данни. Това е единствената структура данни, налична в Лисп и се използва за описание, представяне и съхранение (describe, present and store) на програмен код и данни.

Pure LISP data structures: **atoms** и **lists:**

- Атоми са символи, които имат формата на идентификатори или числени константи;
- Списъци се представят като техните елементи се заграждат в скоби.

Списъци

Структурата списък, в която елементите са атоми, се представя като

(A B C D)

Структурата списък, в която елементите са атоми или списъци (Nested list structures) се представя по същия начин като

(A (B C) D (E (F G)))

Вътрешното представяне се илюстрира като свързан списък.

Лисп синтаксис-model of simplicity

Програмен код и данни имат еднаква форма: списък, заграден в скоби

Даден е списък: (A B C D)

- Интерпретиран като данни, това е списък с 4 елемента.
- Интерпретиран като код, това е извикване (application) на функция на име A над нейните три аргумента B, C, and D.

The LISP machine infinite loop

Популярните Лисп машини (Scheme, Clisp) са реализирани като интерпретатор с безкраен цикъл, който изпълнява следните три действия: **Read-Eval-Print Loop or REPL**

Read; **Evaluate**; **Write**.

Лисп машината повтарящо изпълнява:

1/ чете израз, въведен от потребителя като списък с изключение на литерали и имена.

2/ интерпретира (оценява-evaluates) израза.

3/ извежда резултат

Evaluation and R-E-P Loop

Lisp languages are frequently used with an interactive command line, which may be combined with an integrated development environment. The user types in expressions at the command line, or directs the IDE to transmit them to the Lisp system.

- Lisp *reads* the entered expressions,
- Lisp *evaluates* them,
- Lisp *prints* the result.

For this reason, the Lisp command line is called a "read-eval-print loop", or *REPL*.

The basic operation of the REPL is as follows.

Evaluation and R-E-P Loop

To implement a Lisp REPL, it is necessary only to implement these three functions and an infinite-loop function. (Naturally, the implementation of **eval** will be complicated, since it must also implement all special operators like **if**.) This done, a basic REPL itself is but a single line of code**: (loop (print (eval (read))))**.

Expression evaluation

Интерпретция на израз означава неговата оценка

- Литерали (константи) се оценяват сами. Ако се въведе число, то просто се извежда (see next slide).
- Изрази, които представят обръщение към функции (системни или потребителски), се оценяват така:

Първо, оценява се всеки един от параметрите.

Второ, функцията се прилага над своите предварително оценени параметри.

Диалог с Лисп машина

А/ вход – непосредствена числена стойност

```
=>312 cr
312
```

В/ вход – израз като извикване на първични функции в списъчен формат. Изрази, представени в префиксен списъчен формат, се наричат Cambridge Polish notation

```
=>(+  220  370 )
590
=>(-  90  64 )
26
=>(*  25  8 )
200
=>(/  10  2 )
5
```

Комбинация

Списъчни префксни записи като разгледаните дотук се наричат ***combination(s)***.

Най-левият елемент е операция.

Останалите елементи в списъка са операнди.

Две предимства на този формат:

- Позволява описание на процедури с произволен брой аргументи (operands).

=>(+ 22 44) =>(+ 3 5 7) =>(+ 11 22 33 44)

66 15 110

- Позволява вграждане на комбинации като аргументи.

=>(+ (* 3 5) (- 10 6))

19

Без ограничение върху нивата на вграждане.

How does Lisp machine display results?

Процес, наречен ***combination evaluation***, се провежда в следния ред:

A/ всички подизрази (combinations) се оценяват.

B/ функцията се прилага над своите оценени аргументи.

Последователността A/ B/ е рекурсивна, защото някои подизрази могат да имат аргументи, които самите те са подизрази и т.н., и т.н..

Пример

Ако изразът се представи като дърво, процесът на оценката му (the combination evaluation) се провежда като възходящ обход (bottom-up) на дървото от листата (короната) към корена.

(* (+ 2 (* 4 6))
(+ 3 5 7)
)

Пример за илюстрация

Употреба на имена във ФП

- Императивно програмиране: Имена се задават чрез оператори за дефиниция, декларация, инициализация и присвояване на стойности.
- ФП: Имена се задават по един само начин чрез базова функция. В Scheme това е *define*. В Clisp това е *setq*.

```
=>(define  size 12)
size
```

Казва се: името *size* се свързва със стойност 12.

В своята най-проста форма, ф-ия *define* служи да свърже името със стойността на израз.

(define symbol expression)

Други примери

```
=>( define  size  2 )
size
=>( define  pi  3.1415 )
pi
=>( define  two_pi  ( *  2  pi ) )
two_pi
=>( define  rad  10 )
rad
=>( *  5  size )
10
=>( *  pi  ( *  rad  rad ) )
314.15
=>( *  pi  ( *  size  rad ) )
62.83
```

Други примери

```
=>( define  circlearea  ( *  pi  rad  rad ) )
circlearea
=>circlearea
314.15
=>( define  circlelen  ( *  two_pi  rad ) )
circlelen
=>circlelen
62.83
```

Ф-ии за конструиране на ф-ии

В Scheme потрбителски функции се дефинират със същата базова функция *define*.

В тази си форма *define* взима два списъка като параметри.

Първият е прототип на извикването с име на ф-ията и следван от формалните параметри в списъчен вид.

Вторият е израз, с който се обвързва името на ф-ията.

(define (<function_name> <parameters>)
<body>
)

<parameters> се разделят със space(s), не със comma(s) а <body> е последователност от изрази, всички представени във формата на списъци.

Functions for constructing functions

В Clisp потребителски функции се дефинират с базова функция *defun*.

В тази си форма *defun* предшества името на функцията и два списъка.

Първият списък съдържа параметрите на функцията.

Вторият е израз, с който се обвързва името на ф-ията.

(defun <function_name> (<parameters>)
<body>
)

<parameters> се разделят със space(s), не със comma(s) а <body> е последователност от изрази, всички представени във формата на списъци.

Примери

```
( define   ( <function_name>   <parameters>  )
           <body>
)
```

```
( define   ( sqr  x )
             ( *   x   x )
)
```

```
( define   ( cube  x )
             ( *   x   x   x )
)
```

```
( define   ( reciproc  x )
             ( /   1   x )
)
```

Работа с дефинирани функции

- Явно извикване за изчисляване на изрази

```
=>(sqr 15)          =>(sqr 21)          =>(sqr (sqr 3))
225                 441                 81
```

- Дефиниция на нови функции

```
=>(define  (sum_of_squares  x  y)  (+  (*  x  x) (*  y  y ) ) )
sum_of_squares
=>(sum_of_squares  3  4 )
25
=>(define  ( f  a )  (sum_of_squares (+  a  1) (*  a  2 ) ) )
f
=>(f  5)
136
```

Поток на управление

За по-голяма функционалност на дефинираните функции са необходими средства за реализиране на разклонение в потока на управление.

Припомням условен израз по McCarthy's

$[b_1 \rightarrow e_1, b_2 \rightarrow e_2, \ \ldots \ b_{n-1} \rightarrow e_{n-1}, e_n]$

и неговата Лисп нотация в три версии:

Версия 1

(cond (<b_1> <e_1>)

(<b_2> <e_2>)

. . .

(<b_{n-1}> <e_{n-1}>)

(<b_n> <e_n>)

)

Версия 2

(cond (<b_1> <e_1>)

(<b_2> <e_2>)

. . .

(<b_{n-1}> <e_{n-1}>)

(else <e_n>)

)

Версия 3

(if <predicate>
<then-expression>
<else-expression>
)

Работа с условни изрази

Да се дефинира функция abs(x) = x, if x>0
= 0, if x=0
= -x, if x<0

```
( define  ( abs  x )
        ( cond  (  (>  x  0 )   x )
                (  (=  x  0 )   0 )
                (  (<  x  0 )  ( -  x)  )
        )
)
```

Две други версии

```
( define  ( abs  x )
          ( cond  ((<  x  0 )  ( -  x) )
                    (  else     x )
          )
)
```

Две други версии

```
( define  ( abs  x )
          (  if  (<  x  0)    (-  x)   x )
)
```

Логически изрази

Пример 1: 5 < x < 10

```
(and (> x 5) (< x 10) )
```

Пример 2: дефиниране на собствен предикат >=

```
(define (>= x y )
        (or (> x y) (= x y) ) )
```

ИЛИ

```
(define (>= x y ) (not (< x y) ) )
```

Рекурсия в Лисп

Функция връща сумата на първите n естествени числа

Процедурна версия (C/C++):

```
int sumr(int n){
                    if (n==0) return n;
                    else return n + sumr(n-1); }
```

Функционална списъчна Лисп версия:

```
(define (sumr  n)
      ( if  (=  n  0)   0   (+  n  (sumr  (-  n  1) ) ) )
)
```

Деф. На Ф-ии чрез λ-нотация

λ-нотация служи за дефиниране на безименни функции

(lambda (<parameters>) <body>)

(lambda (x) (+ x 5))

((lambda (x) (+ x 5)) 10)

Обработка на Списъци в Лисп

Лисп синтаксис

- Задаване на списък:
  - (A B C D) извиква функция A(B,C,D)
  - ‘(A B C D) списък от 4 инфо елемента
- Именоване на списък

  (define lis ‘(A B C D))

  (setq lis ‘(A B C D))

Примитиви за списъци

- Списъчен селектор (List selector) car.
- car връща първия елемент (главата) на списък

(car ‘(A B C D))	returns A
(car ‘((A B) C D))	returns (A B)
(car ‘A)	грешка
(car ‘(A))	returns A

Примитиви за списъци

- Списъчен селектор (List selector) cdr.
- cdr връща остатъка на списъка, след като началото (главата) се отдели

(cdr ‘(A B C D))	returns (B C D)
(cdr ‘((A B) C D))	returns (C D)
(cdr ‘A)	грешка
(cdr ‘(A))	returns () – empty list

Примитиви за списъци

- Списъчен конструктор (List constructor) cons.
- cons създава списък от двата си аргумента

(cons ‘A ‘())	returns (A)
(cons ‘A ‘(B C))	returns (A B C)
(cons (car lis) (cdr lis))	returns lis

Примитиви за списъци

- Предикати:
- null(<argument>)
  - Връща T (true), ако аргумент е празен списък
- atom(<argument>)
  - Връща T (true), ако аргументът е атом
- eq(<argument> <argument>)
  - Връща T (true), ако двата аргумента описват еднакви обекти – атоми или списъци

Примери

- (car (cons ‘A ‘B))
- (car (cons ‘A (cons ‘B1 ‘B2)))
- (car (cons ‘A1 ‘A2) ‘B))

- (cdr (cons ‘A ‘B))
- (cdr (cons ‘A (cons ‘B1 ‘B2)))
- (cdr (cons ‘A1 ‘A2) ‘B))

Деф. На ф-ии за списъци

- Функция връща дължината на списък

```
( define  (len  x)
     (if (null x)
                    0
                    (+  1  (len (cdr x) ) )
     )
)
```

Деф. На ф-ии за списъци

- Функция връща дължината на списък

```
( defun len (x)
    (if (null x)
                0
                (+  1  (len (cdr x) ) )
    )
)
```

Деф. На ф-ии за списъци

- Търси дължината на списък lis

  (setq lis ‘(A B C D))

- Извикване на функцията

  (len lis) връща 4

Упражнения

- Task 1. Опишете във функционална списъчна нотация дефиниция на функция, която връща стойността на елемент номер *n* от редицата на Фибоначи.

Упражнения

- Task 2. Опишете във функционална списъчна нотация дефиниция на функция, която връща факториала на цяла стойност *n*.

Упражнения

- Task 3. Опишете във функционална списъчна нотация дефиниция на функция, която връща най-големия общ делител на две цели положителни стойности.

Благодаря
За
Вниманието