Ивановский Государственный Университет
Математический факультет

Отчёт по учебной вычислительной практике

Выполнил студент математического
факультета 2 курса, 2 группы
Фомин Владимир Леонидович
Руководитель практики
Кручинин Алексей Евгеньевич

Иваново 2010

Реализация λ-исчисления Чёрча в функциональных языках программирования

Функциональное программирование

- **Язык программирования –** это формальная знаковая система, используемая для связи человека с компьютером; предназначена для описания данных (информации) и алгоритмов (программ) их обработки на компьютере.
- **Функциональное программирование** — это способ составления программ, в которых единственным действием является вызов функции, единственным способом расчленения программы на части является введение имени для функции и задание для этого имени выражения, вычисляющего значение функции, а единственным правилом композиции — оператор суперпозиции функции. Никаких ячеек памяти, ни операторов присваивания, ни циклов, ни, тем более, блок-схем, ни передач управления.
- *Функция является правилом,* сопоставляющим каждому элементу некоторого класса соответствующий ему единственный элемент из другого класса.
- Другими словами, для заданных двух классов элементов, соответственно называемых *областью определения и областью значений данной функции, каждому элементу из области определ*ения функция ставит в соответствие в точности один элемент из области значений.
- В строго функциональном языке значение выражения единственным образом определяется по значениям составляющих его частей. Таким образом, если некоторое выражение дважды встречается в одном и том же контексте, оно имеет одно и то же значение в обоих случаях. Функции, обладающие этим свойством, называются **чистыми**.

Описание языка Lisp

- **Атом - идентификатор произвольной длины.**
- Среди атомов есть один выделенный : **NIL** – этот атом означает ложь, а также пустое S-выражение. В некотором смысле **NIL** является аналогом пустого множества в теории множеств.
- Определим класс символьных выражений, которые называются *S-выражениями и образуют область определения для* функциональных программ, которые мы будем позднее конструировать. Каждая из следующих строк содержит S-выражение:
- *(ДЖОН СМИТ 33 ГОДА)*
- *((ДЭЙВ 17) (МЭРИ 24) (ЭЛИЗАБЕТ 6))*
- *((МОЙ ДОМ) ИМЕЕТ (БОЛЬШИЕ (СВЕТЛЫЕ ОКНА)))*
- Самым очевидным общим свойством, которое можно подметить в этих примерах, является использование скобок. Действительно, использование скобок в S-выражениях является основополагающим. Если изменить положение или количество скобок, то вместе с этим изменится структура и соответственно смысл самого выражения.
- Кроме скобок в приведенных примерах S-выражения состоят из элементов, которые называются *атомами. Вот примеры* атомов:
- *ДЖОН —127*
- *СМИТ МОЙ*
- 33 *АВ13*

- Атомы бывают двух типов — *символьные и числовые. Символьный* атом обычно является последовательностью букв, хотя может содержать и другие символы, включая цифры, но обязательно должен содержать по меньшей мере один символ, отличающий его от числа. Символьный атом рассматривается как одно неделимое целое. Мы никогда не расчленяем его на составляющие символы. К символьным атомам применяется только одна операция — сравнение, чтобы определить, одинаковые они или разные. Числовые атомы являются последовательностью цифр, возможно следующих за знаком, и обозначают, как обычно, десятичные числа. Вообще, S-выражение содержит смесь символьных и числовых атомов, хотя у нас будут и S-выражения с однотипными атомами

- **S-выражение (символическое выражение):**
- 1) атом есть S-выражение
- 2) (S-выражение-1 . S-выражение-2) - S-выражение
- 3) других S-выражений нет
- S-выражение1 называется **головой (head)** S-выражения, а S-выражение2 - его **хвостом (tail).**

Список:

- **NIL – список**
- **если x – список, а y- атом или список, то (y . x) - список.**
  Изображается список из элементов **A,B,C** следующим образом **(A B C)**
  В точечной нотации это выглядит следующим образом: **(A . (B . (C . Nil))).**
  Правило перехода от дот-нотации к списочной можно выразить в виде:
  "**.(" заменяется на " " - пробел, ". Nil" опускается**

- **(CAR L)**
- Функция возвращает голову списка L
- **(CDR L)**
- Функция возвращает список L без первого элемента (хвост списка L)
- **(CONS E L)**
- Функция добавляет к L элемент E (составить список, голова которого E, хвост - L)

- > (car '(a b c))
- > A
- > (car '(ЖАРКОЕ СОЛНЦЕ))
- >ЖАРКОЕ
- Апостроф показывает интерпретатору, что то, что идет вслед за ним - суть данные, которые нет необходимости пытаться вычислить.
- > (cdr '(a b c))
- > (B C)
- > (cons 'a 'b)
- > (A . B)
- Пусть x - некоторый список. В общем случае верно:
- > (cons (car 'x) (cdr 'x))
- > x

Предикаты

Предикат суть функция, результат которой - логическое значение.

В Лиспе есть две константы: **NIL** и **T**. Под истиной понимается любое значение отличное от **NIL**.

(ATOM E)

Если **E** - атом, то значение функции **(ATOM E)** равно **T**, в противном случае значение функции **NIL**

Примеры:

```
>(atom 'a)
 T
>(atom '(a))
NIL
```

EQ A B)

Значение функции **(EQ A B)** есть **T,** если **A и B -** равные атомы**, NIL** в противном случае**.**

Примеры:

```
(eq ( car '(ЖАРКОЕ СОЛНЦЕ))   'жаркое)
T
> (eq  'солнце   'жаркое)
> NIL
```

(car '(ЖАРКОЕ СОЛНЦЕ)) дает в результате атом

(NULL E)

Предикат возвращает **T, если E - пустой список и NIL в противном случае.**

Примеры:

```
> (null (cdr '(b)))
T
> (null '(b))
NIL
```

(COND (L1 E1)(L2 E2)...(LN EN)(T ENN))

- Если **L1** не **NIL,** то E1 и вычисления прекращаются, в качестве значения формы возвращается значение **E1**. Если **L2** не **NIL**, то **E2,** вычисления прекращаются, в качестве значения формы возвращается значение **E2**. … Иначе вычисляем **ENN.**

```
>(cond
((= 4 5) (princ "4 = 5"))
((/= 4 5) (princ "4 <> 5"))
(t (princ "hello world"))
)
4 <> 5
"4 <> 5"
```

В данном случае **L1 = (= 4 5), L2 =(/= 4 5)**

```
> (= 4 5)
NIL
> (/= 4 5)
T
```

(DEFUN имя_функции (параметры)
выражение1
выражение2
…
выражениеN)

Под "выражениемі" понимается любая форма. Например, вызов функции - это форма, значение ее - значение функции на входных данных.

Функция вычисляющая факториал

```
> (defun N! (n)
(cond
((eq n 0) 1)
(t (* n (N! (- n 1))))
)
)
>N!
```

Вызов функции

- > (N! 45)
- >119622220865480194561963161495657715064383733760000000000

Синтаксис лямбда-выражения аналогичен синтаксису DEFUN, за исключением того, что вместо DEFUN используется LAMBDA и имя функции опускается.

```
> (lambda (x) (+ x x) )
#<CLOSURE :LAMBDA (X) (+ X X)>
```

Для того, чтобы использовать лямбда-выражение организуют лямбда-вызов, который имеет следующий синтаксис:

(«лямбда-выражение» . «список фактических параметров»)

- Пример лямбда-вызова
- > ((lambda (x) (+ x x))10)
- 20
- > ((lambda (x y) (+ (* x x) (* y y))) (+ 1 2) (+ 2 3))
- 34
- Последний лямбда-вызов вычисляет сумму квадратов 3 и 5.

(EVAL E)
Функция вычисляет S-выражение **E**
(QUOTE E)
блокировка вычисления **(QUOTE E)** равносильно **'E**

```
> (car '( b c))
B
> '(car '( b c))
(CAR '(B C))
> (eval '(car '( b c)))
B
> (defun ex_eval () (eval '(defun g (x)(+ x 2)))
(g 4)
)
ex_eval
> (ex_eval)
6
```

Арифметические функции

(+ x1 x2 ... xN)
возвращает сумму своих аргументов.
Аналогично для "*****", "**/**" **и** "**-**"
(EXPT x y)
x в степени **y**
(MOD x y)
остаток от **x/y**
(SQRT x)
квадратный корень из x.

Справочник функций для Bee Lisp:
http://www.beelisp.ru/online-help/default.htm

Задача о ферзях.

Программа, на языке Lisp, расставляющая m ферзей на доске размера m x m таким образом, чтобы эти ферзи не били друг друга

Расставить на шахматной доске с числом клеток *m*m* *m* ферзей так, чтобы ни один из них не находился под боем другого.

Каждая вертикальная линия (*i* =const) должна содержать ровно одного ферзя.

Каждая горизонтальная линия (*j*=const) должна содержать только одного ферзя.

Каждая диагональ, параллельная главной диагонали (*i-j* = const), должна содержать не более одного ферзя.

Каждая диагональ, параллельная побочной диагонали (*i+j* = const), должна содержать не более одного ферзя.

Список параметров основных функций

- *m* - количество ферзей
- *i* – номер вертикали для клетки (*i, j*)
- j – номер горизонтали для клетки (*i, j*)
- *i-j* – номер диагонали, параллельной главной, для клетки (*i, j*) , *1-m<=i-j<=m-1*
- *i+j* – номер диагонали, параллельной побочной, для клетки (*i, j*) , *2<=i+j<=2*m*

Описание переменнных z, a, b, c

- *z* – список длины *m* номеров позиций ферзей по горизонтали , расставленных на вертикальных линиях.
- Позиция числа в списке *z* равна номеру вертикали *i*, на которой стоит ферзь, а значение этого числа равно номеру горизонтали *j*, на которой стоит этот ферзь.
- *a* – список длины *m* горизонтальных линий, которые не находятся под боем, содержащий элементы *T* и *NIL*.
- Позиция элемента в списке a равна номеру *j* горизонтальной линии. Если этот элемент равен *T*, то на этой горизонтальной линии нет ферзя. Если Если этот элемент равен *Nil*, то на этой горизонтальной линии стоит ферзь.
- b – список длины 2*m-1* линий, параллельных главной диагонали, которые не находятся под боем. Список содержит элементы *T* и *NIL*.
- Позиция элемента в списке *b* равна номеру линии (i-j). Если этот элемент равен *T*, то на этой линии нет ферзя. Если Если этот элемент равен *Nil*, то на этой линии стоит ферзь.
- *c* – список длины 2*m-1* линий, параллельных побочной диагонали, которые не находятся под боем. Список содержит элементы *T* и *NIL*.
- Позиция элемента в списке *b* равна номеру линии (i+j). Если этот элемент равен *T*, то на этой линии нет ферзя. Если Если этот элемент равен *Nil*, то на этой линии стоит ферзь.

Замечание о переменных *z, a b, c*

- Списки *z, a b, c* являются глобальными переменными. Значения этих переменных являются выходным результатом следующих 6 функций: *z0(m, zn); a0(m, an); b0(m, bn); c0(m, cn); fight(i, j, m); nofight(i, j, m)*.
- Всякий раз при вызове этих функций значения этих переменных изменяются. Тем не менее, если всюду включить эти переменные в качестве формальных параметров в заголовок тех функций, значения которых зависят от этих переменных, то все эти функции станут чистыми функциями.

Решение задачи о ферзях.

- 1. Производится попытка поставить ферзя на свободное поле вертикальной линии с номером *i*: сначала номер горизонтали *j = 1*; если поле *(i ,j)* занято, то при j<m делается попытка поставить ферзя на более высокое по вертикали поле *(i ,j+1)*
- 2. Если ферзь успешно поставлен на свободное поле вертикальной линии с номером i, то при *i<m* делается попытка поставить ферзя на свободное поле вертикальной линии с номером *i+1*:
- 3. Если свободных полей на вертикальной линии с номером i нет (*j=m && tryd=nil*), то освобождаются три линии (горизонталь и две диагонали) , проходящие через ферзя, стоящего на линии с номером *i-1*
- 4. Если *i=m*, то производится распечатка решения задачи. После распечатки освобождаются три линии, проходящие через поле *(i ,j)*, где *i=m*.