Ассоциативные списки.

Лекция 10.

Специальности : 230105, 010501

Структура ассоциативных списков.

Определение. Ассоциативный список или просто а-список (a-list) есть основанная на списках и точечных парах структура данных, описывающая связи наборов данных obji и ключевых полей keyi, для работы с которой существуют готовые функции.

Ассоциативные списки могут быть следующих двух видов :

((key1.obj1)(key2.obj2) … (keyN.objN))

((key1 obj1)(key2 obj2) … (keyN objN))

Первый вид ассоциативных списков в muLISP'е имеет место в тех случаях, когда связанные с ключами являются атомарными объектами.

Ассоциативные списки применяются при работе с динамическими базами данных в оперативной памяти.

Создание ассоциативного списка.

С помощью встроенной функции PAIRLIS в muLISP'е формируется a-список из списка ключей keys и списка objects соответствующих им объектов. Формат вызова :

(pairlis keys objects a_list)

Третий аргумент функции pairlis есть формируемый a-список, в начало которого добавляются новые пары "ключ-объект". При вызове в качестве значения a_list либо задается nil, либо предполагается, что a_list был сформирован ранее.

В newLISP-tk a-список можно сформировать, используя функцию set, которая вычисляет значение имени - своего первого аргумента, и связывает с вычисленным значением значение второго аргумента.

Функция set выполняет связывание наподобие setq, но setq в отличие от set не вычисляет значения имени.

Пример : машинный словарь основ.

```
; Пример : построение в оперативной
; памяти машинного словаря основ слов с помощью
; ассоциативного списка.
(setq t_base nil)
(pairlis '(base_number symb_code basechange_class
           flect_class)
         '(("001" "002" "003" "004" "005")
           ("azot" "balk" "ball" "bank1" "bank2")
           ("" "11" "" "" "11")
           ("001" "060" "001" "006" "060"))
       t_base)
```

Как результат вызова функции pairlis значением t_base становится список :

```
((base_number "001" "002" "003" "004" "005")
 (symb_code "azot" "balk" "ball" "bank1" "bank2")
 (basechange_class "" "11" "" "" "11")
 (flect_class "001" "060" "001" "006" "060"))
```

Этот же пример в newLISP-tk.

```
(set 't_base '((base_number "001" "002" "003" "004" "005")
               (symb_code "azot" "balk" "ball" "bank1" "bank2")
               (basechange_class "" "11" "" "" "11")
               (flect_class "001" "060" "001" "006" "060")))
```

Как результат вызова функции set значением t_base становится список :

```
((base_number "001" "002" "003" "004" "005")
 (symb_code "azot" "balk" "ball" "bank1" "bank2")
 (basechange_class "" "11" "" "" "11")
 (flect_class "001" "060" "001" "006" "060"))
```

Поиск элементов в ассоциативном списке.

Ассоциативный список можно рассматривать как отображение множества ключей на множество соответствующих им объектов. В muLISP'е конкретные данные можно получить по значению ключа с помощью функции :

(assoc key a_list)

В качестве значения assoc возвращает пару "ключ-объект". Пример (для машинного словаря основ) :

```
(setq t_base nil)
(setq t_base
      (pairlis '(base_number symb_code
                 basechange_class flect_class)
               '(("001" "002" "003" "004" "005")
                 ("azot" "balk" "ball" "bank1" "bank2")
                 ("" "11" "" "" "11")
                 ("001" "060" "001" "006" "060")) t_base))
(assoc 'base_number t_base)
Результат :
(base_number "001" "002" "003" "004" "005")
```

В newLISP-tk функция *assoc* работает аналогично.

Поиск ключа по объекту (muLISP).

В muLISP’е функция RASSOC :

(rassoc obj a_list)

находит ключ по заданному объекту. В качестве значения rassoc возвращает пару “ключ-объект”, НО (!) только в том случае, если a_list есть список точечных пар. Пример :

```
(setq t_base1 nil)
(setq t_base1 (pairlis '(base_number symb_code
                         basechange_class flect_class)
                       '("001" "azot" "" "001") t_base1))
; Формирование t_base описано в предыдущем
; примере
```

Вызов :

```
(rassoc '("azot" "balk" "ball" "bank1" "bank2") t_base)
```

возвращает в качестве результата nil, а вызов :

```
(rassoc '"azot" t_base1)
```

возвращает в качестве результата точечную пару :

(symb_code . ”azot”)

Поиск данных по ключу в newLISP-tk.

Функция :

(lookup key a_list int_index)

находит в ассоциативном списке пару *“ключ-объект”* с тем же значением ключа, что у *key* и возвращает либо название объекта из того списка, который соответствует ключу *key (*в соответствии со значением целочисленного индекса *int_index*), либо название последнего объекта из этого списка (если индекс не указан).

Пример :

(lookup 'base_number t_base)

возвращает в качестве результата “005”, а

(lookup 'base_number t_base 2)

возвращает в качестве результата “002”.

Добавление элементов в ассоциативный список.

Ассоциативный список можно обновлять и использовать в режиме стека. Новые пары “ключ-объект” добавляются к нему только в начало списка, хотя в списке уже могут быть данные с тем же ключом. В muLISP’е добавление осуществляется функцией ACONS (в newLISP-tk функция ACONS отсутствует, ее заменяют показанной справа суперпозицией вызовов CONS) :

(acons key obj a_list)⇔(cons (cons key obj) a_list)

Поскольку ASSOC просматривает список слева направо и доходит лишь до первой пары с искомым ключом, то более старые пары остаются вне рассмотрения. Использование ассоциативных списков подобным образом может, в частности, решить проблему поддержки меняющихся связей переменных и контекста вычисления. С такой целью ассоциативные списки используются при программировании самого интерпретатора Лиспа. Преимущества : простота изменения связей и возможность возврата к значениям старых связей. Недостаток : поиск данных замедляется пропорционально длине списка.

Добавление элементов (muLISP).

Примеры.

```
(setq t_base2 nil)
(setq t_base2 (pairlis '(base_number symb_code
                         basechange_class)
                       '(("001" "002" "003" "004" "005")
                         ("azot" "balk" "ball" "bank1" "bank2")
                         ("" "11" "" "" "11")) t_base2))
(setq t_base3 nil)
(setq t_base3 (pairlis '(base_number symb_code
                         basechange_class)
                       '("001" "azot" "") t_base3))
```

Результатом вызова :

```
(setq t_base2 (acons flect_class
                     '("001" "060" "001" "006" "060") t_base2))
```

будет добавление пары
(flect_class "001" "060" "001" "006" "060") в начало
списка t_base2.

Результатом вызова :

```
(setq t_base3 (acons flect_class '"001" t_base3))
```

будет добавление пары (flect_class . "001")
в начало списка t_base3.

Добавление элементов (newLISP-tk).

```
(set 't_base2 '((base_number "001" "002" "003" "004" "005")
                (symb_code "azot" "balk" "ball" "bank1" "bank2")
                (basechange_class "" "11" "" "" "11")))
(set 't_base3 '((base_number "001")
                (symb_code "azot")
                (basechange_class "")))
```

Результатом вызовов :

```
(set 't_base2 (cons '(flect_class "001" "060" "001" "006" "060") t_base2))
(set 't_base3 (cons '(flect_class '"001") t_base3))
```

будет добавление соответствующих пар в начала списков t_base2 и t_base3, что равносильно вызовам ACONS в muLISP’e.

Модификация ассоциативных списков.

Ассоциативный список можно изменить путем :

- **Физического изменения данных, связанных с ключом;**
- **Физического удаления данных, связанных с ключом;**
- **Изменения ключа.**

Во всех трех случаях изменение ассоциативных списков происходит с применением структуроразрушающих функций. Описанные далее функции модификации ассоциативных списков в результате своего выполнения теряют старые значения ассоциативных списков.

Модификация данных (muLISP).

```
; Изменение данных, связанных с ключом
(defun putassoc (key obj a_list)
 ((null a_list) nil)
 ((equal (caar a_list) key)(rplacd (car a_list) obj))
 ((null (cdr a_list))(rplacd a_list (list (cons key obj))))
 (putassoc key obj (cdr a_list)))
;         lst будет изменен :
(setq lst nil)
(setq lst (pairlis '(Name Group Math Phys)
                   '(Ivanov 2091 4 5) lst))
(putassoc 'Group '2092 lst)
```

Значением lst изначально будет :

((Name . Ivanov)(Group . 2091)(Math . 4)(Phys . 5))

В результате вызова putassoc значение lst изменится на :

((Name . Ivanov)(Group . 2092)(Math . 4)(Phys . 5))

Удаление данных, связанных с ключом (muLISP).

```
; Удаление данных, связанных с ключом
(defun remassoc (key a_list)
 ((null (cdr a_list)) nil)
 ((and (equal (caar a_list) key)
       (rplaca a_list (cadr a_list)))
  (rplacd a_list (cddr a_list)))
 (remassoc key (cdr a_list)))
;          Из lst1 будет удален элемент :
(setq lst1 nil)
(setq lst1 (pairlis '(Name Group Math Phys)
                    '(Ivanov 2091 4 5) lst1))
(remassoc 'Group lst1)
Значением lst1 изначально будет :
    ((Name . Ivanov)(Group . 2091)(Math . 4)(Phys . 5))
В результате вызова remassoc
значение lst1 изменится на :
            ((Name . Ivanov)(Math . 4)(Phys . 5))
```

Изменение ключа (muLISP).

```
; Изменение ключа
(defun keyassoc (old_key new_key a_list)
 ((null a_list) nil)
 ((equal (caar a_list) old_key)
  (rplaca (car a_list) new_key))
 (keyassoc old_key new_key (cdr a_list)))
; В lst2 будет изменен один из ключей
(setq lst2 nil)
(setq lst2 (pairlis '(Name Group Math Phys)
                    '(Ivanov 2091 4 5) lst2))
(keyassoc 'Name 'First_name lst2)
```

Значением lst2 изначально будет :

((Name . Ivanov)(Group . 2091)(Math . 4)(Phys . 5))

В результате вызова
значение lst2 изменится на :

((First_name . Ivanov)(Group . 2091)(Math . 4)(Phys . 5))

Модификация данных (newLISP-tk).

Осуществляется функцией replace-assoc. Примеры :

(set 'lst '((Name Ivanov)(Group 2091)(Math 4)(Phys 5)))

1) Изменение данных, связанных с ключом (аналог putassoc)

В результате вызова :

(replace-assoc 'Group lst '(Group 2091))

значение lst изменится на ((Name Ivanov) (Group 2091) (Math 4) (Phys 5)).

2) Изменение ключа (аналог keyassoc)

В результате вызова :

(replace-assoc 'Name lst (cons 'First_name (last $0)))

значение lst изменится на :

((First_name Ivanov) (Group 2091) (Math 4) (Phys 5)).

Здесь $0 – системная переменная со значением пары “ключ-объект”.

3) Удаление данных, связанных с ключом.

В результате вызова :

(replace-assoc 'Group lst)

значение lst изменится на ((First_name Ivanov) (Math 4) (Phys 5)).

Анализ и формулировка размерности формул : постановка задачи.

Для анализа корректности формулы мы должны определить соответствие величин левой и правой части формулы. При этом складываемые и вычитаемые величины должны иметь одни и те же единицы измерения. Первым шагом мы определяем фигурирующие в формулах величины и их единицы измерения. Единицы измерения величин задаются исходя из описывающих их формул, формулы задают связи с другими величинами. Пример.

Пусть у нас исходными величинами будут масса (M), длина пройденного пути (L) и время (T). И есть формулы, описывающие величины, производные от исходных :

Скорость $v = \frac{L}{T} = L^1 * T^{-1}$ **Ускорение** $a = \frac{L}{T^2} = L^1 * T^{-2}$

Применение ассоциативных списков для описания величин в формулах.

Задаваемую формулой зависимость можно представить списком, задающим степень каждой из исходных величин в описываемом формулой выражении. Назовем этот список списком размерности.

Для рассмотренных величин это будет список (M L T), где M – степень массы, L – степень длины пути, T – степень времени.

Исходя из представленных формул, получаем списки размерности :

для скорости : (0 1 -1); для времени : (0 0 1);

для ускорения : (0 1 -2); для массы : (1 0 0).

для длины пути : (0 1 0);

Указанное соответствие можно описать ассоциативным списком, задав имена величин в качестве ключей, а списки размерности – в качестве соответствующих им объектов.

Анализ размерности формул : формирование исходной структуры.

```
; Формирование ассоциативного списка
; для представления размерностей величин
(setq lst_razm nil)
(setq lst_razm (pairlis '(v u a T M L)
                        '((0 1 -1)(0 1 -1)(0 1 -2)
                          (0 0 1)(1 0 0)(0 1 0)) lst_razm))
```

В результате значением lst_razm становится ассоциативный список, который описывает соотношение величин в оперирующих ими формулах :

```
((v 0 1 -1)(u 0 1 -1)(a 0 1 -2)(T 0 0 1)(M 1 0 0)(L 0 1 0))
```

Подсчет размерности.

Производится по следующим правилам :

Размерность (+ формула$_1$ формула$_2$)= $\begin{cases} размерность(формула_1), если\ размерности\ совпадают \\ ? - иначе \end{cases}$

Размерность (- формула$_1$ формула$_2$)= $\begin{cases} размерность(формула_1), если\ размерности\ совпадают \\ ? - иначе \end{cases}$

Размерность (* формула$_1$ формула$_2$)= $\begin{cases} Сумма(размерность(формула_1), размерность(формула_2)), если\ они\ не\ ? \\ ? - иначе \end{cases}$

Размерность (/ формула$_1$ формула$_2$)= $\begin{cases} Разность(размерность(формула_1), размерность(формула_2)), если\ они\ не\ ? \\ ? - иначе \end{cases}$

Подсчет степени исходных величин в выражениях.

```
; Вспомогательная функция определения длины
; списка
(defun list_len (lst)
 ((null lst) 0)
 (+ 1 (list_len (cdr lst))))
; Путь
(defun leng (a_list)
 ((equal (list_len a_list) 3)(cadr a_list))
 (caddr a_list))
; Время
(defun time (a_list)
 ((equal (list_len a_list) 3)(caddr a_list))
 (cadddr a_list))
; Масса
(defun massa (a_list)
 ((equal (list_len a_list) 3)(car a_list))
 (cadr a_list))
```

Размерность формул, содержащих умножение и деление.

```
; Вспомогательная функция - делает список
; из трех элементов
(defun tr (a b c)
 (list a b c))
; Размерность произведения величин
(defun mult (obj1 obj2)
 ((equal obj1 '?) nil)
 ((equal obj2 '?) nil)
 (tr (+ (massa obj1)(massa obj2))
    (+ (leng obj1)(leng obj2))
    (+ (time obj1)(time obj2))))
; Размерность частного величин
(defun devide (obj1 obj2)
 ((equal obj1 '?) nil)
 ((equal obj2 '?) nil)
 (tr (- (massa obj1)(massa obj2))
    (- (leng obj1)(leng obj2))
    (- (time obj1)(time obj2))))
```

Функция совпадения размерностей.

```
; Функция определяет совпадение размерностей
(defun sov (obj1 obj2)
 ((equal obj1 '?) nil)
 ((equal obj2 '?) nil)
 ((and (equal (massa obj1)(massa obj2))
       (equal (leng obj1)(leng obj2))
       (equal (time obj1)(time obj2))) t) nil)
```

Написание основной функции.

Аргументы : формула form и список lst.

Результат : размерность формулы – в случае ее корректности,

? – в противном случае.

Условие окончания рекурсии : мы дошли до символа, обозначающего конкретную величину :

((atom form)(assoc form lst))

Полное описание функции :

```
(defun razm (form lst)
 ((atom form)(assoc form lst))
 (let* ((d1 (razm (cadr form) lst))
        (d2 (razm (caddr form) lst)))
 ((equal '+ (car form))(if (sov d1 d2) d1 '?))
 ((equal '- (car form))(if (sov d1 d2) d1 '?))
 ((equal '* (car form))(mult d1 d2))
 ((equal '/ (car form))(devide d1 d2)) '?))
```

Анализ и формулировка размерности формул : вариант для newLISP-tk.

```
(define (list_len lst)
 (cond
  ((null? lst) 0)
  (true (+ 1 (list_len (rest lst))))
 ))

(define (leng a_list)
 (cond
  ((= (list_len a_list) 3)(nth 1 a_list))
  (true (nth 2 a_list))
 ))

(define (time_st a_list)
 (cond
  ((= (list_len a_list) 3)(nth 2 a_list))
  (true (nth 3 a_list))
 ))

(define (massa a_list)
 (cond
  ((= (list_len a_list) 3)(nth 0 a_list))
  (true (nth 1 a_list))
 ))

(define (tr a b c)
 (list a b c))
```

```
(define (mult obj1 obj2)
 (cond
  ((= obj1 '?) nil)
  ((= obj2 '?) nil)
  (true (tr (+ (massa obj1)(massa obj2))
            (+ (leng obj1)(leng obj2))
            (+ (time_st obj1)(time_st obj2))
        ))))

(define (devide obj1 obj2)
 (cond
  ((= obj1 '?) nil)
  ((= obj2 '?) nil)
  (true (tr (- (massa obj1)(massa obj2))
            (- (leng obj1)(leng obj2))
            (- (time_st obj1)(time_st obj2))
        ))))

(define (sov obj1 obj2)
 (cond
  ((= obj1 '?) nil)
  ((= obj2 '?) nil)
  ((and (= (massa obj1)(massa obj2))
    (= (leng obj1)(leng obj2))
    (= (time_st obj1)(time_st obj2))) true)
  (true nil)
 ))
```

Анализ и формулировка размерности формул : вариант для newLISP-tk (продолжение).

```
(define (razm form lst)
 (cond
  ((atom? form)(assoc form lst))
  (true (letn ((d1 (razm (nth 1 form) lst))
               (d2 (razm (nth 2 form) lst)))
         (cond
          ((= '+ (nth 0 form))(if (sov d1 d2) d1 '?))
          ((= '- (nth 0 form))(if (sov d1 d2) d1 '?))
          ((= '* (nth 0 form))(mult d1 d2))
          ((= '/ (nth 0 form))(devide d1 d2))
          (true '?)
         )
        ))))

; Формирование ассоциативного списка для представления размерностей
; величин
; Заглавные и строчные буквы - различаются !
(set 'lst_razm '((v 0 1 -1)(u 0 1 -1)(a 0 1 -2)
                 (t 0 0 1)(m 1 0 0)(l 0 1 0)))
```

Тестовый пример.

Вызов функции razm для выражения, соответствующего формуле v=u+a*t (где u – начальная скорость) :

(razm '(+ u (- (* a t) v)) lst_razm)

дает результатом список :

(u 0 1 -1),

показывающий степени величин M, L и T в рассматриваемом выражении :

- **для массы это 0;**
- **для длины пройденного пути это 1;**
- **для времени это –1,**

что соответствует соотношению L и T для скорости.