ДЕКЛАРАТИВНОЕ ПРОГРАММИРОВАНИЕ

Лекция 4. Символы в Лиспе.

Символы в Лиспе.

Определение. Символ – это имя, состоящее из букв, цифр, специальных знаков и обозначающее какой-либо предмет, объект, вещь, действие из реального мира.

В Лиспе символы могут обозначать любые лисповские объекты, включая функции.

Символ является структурным объектом, состоящим из четырех компонент, соответствующих имени, значению, а также связанных с символом определению функции и списку свойств.

Для нового символа интерпретатор резервирует память для возможного значения, определения функции и других свойств. Символы сохраняются в памяти в списке объектов, который содержит как созданные пользователем, так и внутрисистемные символы. В Коммон Лиспе и ряде новых Лисп-систем, в том числе newLISP-tk, возможно использовать несколько различных списков объектов, каждый из которых именуется пакетом или контекстом (в newLISP-tk).

Функции чтения свойств символа в Коммон Лиспе.

Для чтения значений различных системных свойств символа в Коммон Лиспе, а также в библиотеке COMMON.LSP GCLisp'a существуют специальные функции, приведенные в таблице 1.

Таблица 1. Функции чтения свойств символа.

Характеристика	Форма представления	Функция чтения
Имя	Набор литер	(SYMBOLP *имя*)
Значение	Произвольный лисповский объект	(SYMBOL-VALUE *имя*)
Определение функции	Лямбда-выражение	(SYMBOL-FUNCTION *имя*)
Список свойств	Список точечных пар	(SYMBOL-PLIST *имя*)

Свойства символов.

В Лиспе с символом можно связать именованные свойства. Свойства символа записываются в хранимый вместе с символом список свойств :

(<имя символа> (<имя свойства 1> <значение свойства 1>) . . .

(<имя свойства N> <значение свойства N>))

Для работы со списками свойств в основных диалектах Лиспа имеются три встроенные функции :

1.Включение свойства в список свойств.

(put <символ> <свойство> <значение свойства>)

2. Просмотр значения заданного свойства.

(get <символ> <свойство>)

3. Удаление заданного свойства из списка свойств.

(remprop <символ> <свойство>)

Показанное описание свойств символов используется при создании динамических баз данных и текстовых редакторов.

Свойства символов в Коммон Лиспе.

В Коммон Лиспе функции наподобие PUT не существует.

Поскольку свойства символов находятся в связанных с символами ячейках памяти, для присваивания значений которым в Коммон Лиспе используется обобщенная функция присваивания SETF, присваивание свойств в Коммон Лиспе осуществляется суперпозицией функций SETF и GET :

(setf (get <символ> <свойство>) <значение свойства>)

Здесь вызов функции GET возвращает в качестве значения ячейку памяти для данного свойства, содержимое которой обновляет произведенный вызов SETF. Присваивание будет работать и в том случае, когда у символа ранее не было такого свойства.

Псевдофункция SETF меняет физическую структуру списка свойств.

Формирование списка свойств (GCLisp).

<u>Постановка задачи.</u> Дан символ с некоторым именем *name*. Требуется сформировать список свойств.

Данная задача решается в три этапа. Сначала необходимо задать (ввести) количество свойств, затем ввести список названий свойств и только затем – значения свойств.

```
; Формирование списка свойств символа
; Головная функция формирования списка свойств
(defun f1 (name)
 (print (pack* "Введите количество свойств символа "
               name))
 (f3 name (f2 name (read))))

; Функция формирования списка названий свойств
(defun f2 (name num)
 ((zerop num) nil)
 (print (pack* "Введите название "
               num
               "-го свойства символа " name " : "))
 (cons (read)(f2 name (- num 1))))

; Ввод значений свойств
(defun f3 (symb_name prop_name_list)
 ((null prop_name_list) nil)
 (print (pack* "Введите значение свойства "
               (car prop_name_list)
               " символа " symb_name))
 (put symb_name (car prop_name_list)(read))
 (f3 symb_name (cdr prop_name_list)))
```

Здесь в качестве вспомогательной используется функция конкатенации : (pack* *строка1 … строкаN*)

Удаление заданного свойства (GCLisp).

Постановка задачи. **Дан список символов lst и некоторое свойство prop. Требуется : удалить свойство prop у всех тех символов списка lst, у которых это свойство имеется.**

Для решения данной задачи воспользуемся функциями get и remprop.

```
; Вариант 1
(defun f4 (lst prop)
 ((null lst) nil)
 (remprop (car lst) prop)
 (f4 (cdr lst) prop))
; Вариант 2 - неправильный
(defun f5 (lst prop)
 ((null lst) nil)
 ((get (car lst) prop)(remprop (car lst) prop))
 (f5 (cdr lst) prop))
```

```
; Подготовка тестовых
; данных
(put one 'Num 'Nechet)
(put two 'Num 'Chet)
(put A 'Sym 'A-lat)
(put B 'Sym 'B-lat)
(put three 'Num 'Nechet)
(put four 'Num 'Chet)
(put C 'Sym 'C-lat)
(put D 'Sym 'D-lat)
```

Результатом вызова (f4 '(one A two B) 'Num) будет удаление свойства Num у символов one и two.

НО ! вызов (f5 '(three C four D) 'Num) не приведет к удалению свойства Num у символа four, поскольку в случае обнаружения искомого свойства у одного из символов списка рекурсивного вызова функции f5 уже не произойдет.

Удаление заданного свойства у символов списка : вариант с использованием LET.

Еще один вариант решения задачи связан с использованием локального определения LET.

```
; Удаление заданного свойства - вариант с использованием
; локального определения LET
(defun f6 (lst prop)
 (let
   ((obj_list lst)
    (property prop))
  ((null obj_list) nil)
  (remprop (car obj_list) property)
  (f6 (cdr obj_list) property)
 )
)
```

Изменение значения заданного свойства (GCLisp).

Постановка задачи. Дан список символов lst и некоторое свойство prop. Требуется : заменить текущее значение свойства prop заданным значением val у всех тех символов списка lst, у которых это свойство имеется.

```
; Изменение значения заданного свойства
(defun chngprop (lst prop val)
 ((null lst) T)
 ((and (get (car lst) prop)
       (chngprop (cdr lst) prop val))
  (put (car lst) prop val))
 (chngprop (cdr lst) prop val))
; Тестовый набор данных
(put one 'What_is_it Number)
(put two 'What_is_it Number)
(put A 'What_is_it Symbolic_value)
(put B 'What_is_it Symbolic_value)
(setq input_list '(one A two B))
```

Вызов (chngprop input_list 'What_is_it 'Atom) приводит к тому, что у всех элементов списка input_list, для которых определено свойство What_is_it, в качестве нового значения этого свойства будет задано Atom.

Изменение значения свойства на заданное (GCLisp).

Постановка задачи. **Дан список символов lst и некоторое значение old_val свойства prop. Требуется : заменить значение old_val свойства prop новым значением new_val у всех тех символов списка lst, у которых это свойство имеется.**

```
; Замена заданного значения заданного свойства новым
(defun chngprop2 (lst prop old_val new_val)
 ((null lst) T)
 ((and (equal (get (car lst) prop) old_val)
       (chngprop2 (cdr lst) prop old_val new_val))
  (put (car lst) prop new_val))
 (chngprop2 (cdr lst) prop old_val new_val))
; Тестовый набор данных
(put one 'What_is_it Number)
(put two 'What_is_it Number)
(put A 'What_is_it Symbolic_value)
(put B 'What_is_it Symbolic_value)
(setq input_list '(one A two B))
```

Вызов (chngprop2 input_list 'What_is_it 'Symbolic_value 'Symbol) приводит к тому, что у всех элементов списка input_list, значением свойства What_is_it которых является Symbolic_value, в качестве нового значения этого свойства будет задано Symbol.

Использование списков свойств символов при разработке текстового редактора (GCLisp).

Постановка задачи. Требуется обеспечить выполнение функций редактора для разных режимов работы при нажатии определенных клавиш. С нажатием одной и той же клавиши в разных режимах могут быть связаны разные функции. Отдельные клавиши задействуются только в определенных режимах : F3 – загрузка текста из файла (при активизации главного меню), ↑, ↓, ← и → - перемещение курсора, F2 – сохранение текста в файле (в режиме правки текста).

Решение. Свяжем с символом, соответствующим ASCII-коду каждой из задействованных клавиш, список свойств. Каждому из режимов, где клавиша будет задействована, поставим в соответствие название свойства, а имени вызываемой по нажатию клавиши функции – значение свойства. Для вызова самой функции воспользуемся функционалом MAPC.

Пример для клавиш Enter и Backspace в режиме редактирования (смотри файл muledit.lsp) :

```
(MAPC '(LAMBDA (PAIR)
                    (PUT (ASCII (CAR PAIR)) 'EDITOR (CADR PAIR)))
      '((13 EDITOR-ENTER)
        (8  EDITOR-BACKSPACE)))
```

Литература.

1 Хювенен Э., Сеппянен Й. Мир Лиспа. Т.1. – М.:Мир, 1990. С. 61-63, 168-172 , 326-327.

Пример : машинный словарь основ.

1. качестве примера использования списков свойств символов для построения динамических БД рассмотрим представление в памяти машинного словаря основ для задачи морфологического анализа. Согласно приведенному в [2] описанию, для каждой основы в словаре приводится порядковый номер (десятиричное число), буквенный код основы, номер флективного класса и номер основоизменительного класса. Рассмотрим принципиальную возможность построения подобного словаря с применением имеющихся в GCLisp е средств работы с файлами на внешних носителях и списками свойств. Каждой представленной в словаре основе мы поставим в соответствие символьный объект с именем, соответствующем буквенному коду основы по словарю (azot, balk, ball, bank1, bank2). Порядковый номер основы по словарю, номера флективных и основоизменительных классов будем рассматривать как свойства соответствующего символа, описываемые списком свойств. Для создания динамической базы данных в памяти таблицу основ мы представим как объект, имеющий в качестве свойств данные конкретных основ.

2. newLISP-tk подобная динамическая БД может быть организована на основе ассоциативных списков.

Морфологическая классификация слов.

В основу построения алгоритмов морфологического анализа и синтеза слов положено разбиение всех слов на морфологические классы.

<u>Определение.</u> Морфологический класс определяет характер изменения буквенного состава форм слов.

Изменение форм слов может быть связано с изменением буквенного состава либо основы слова, либо его окончания.

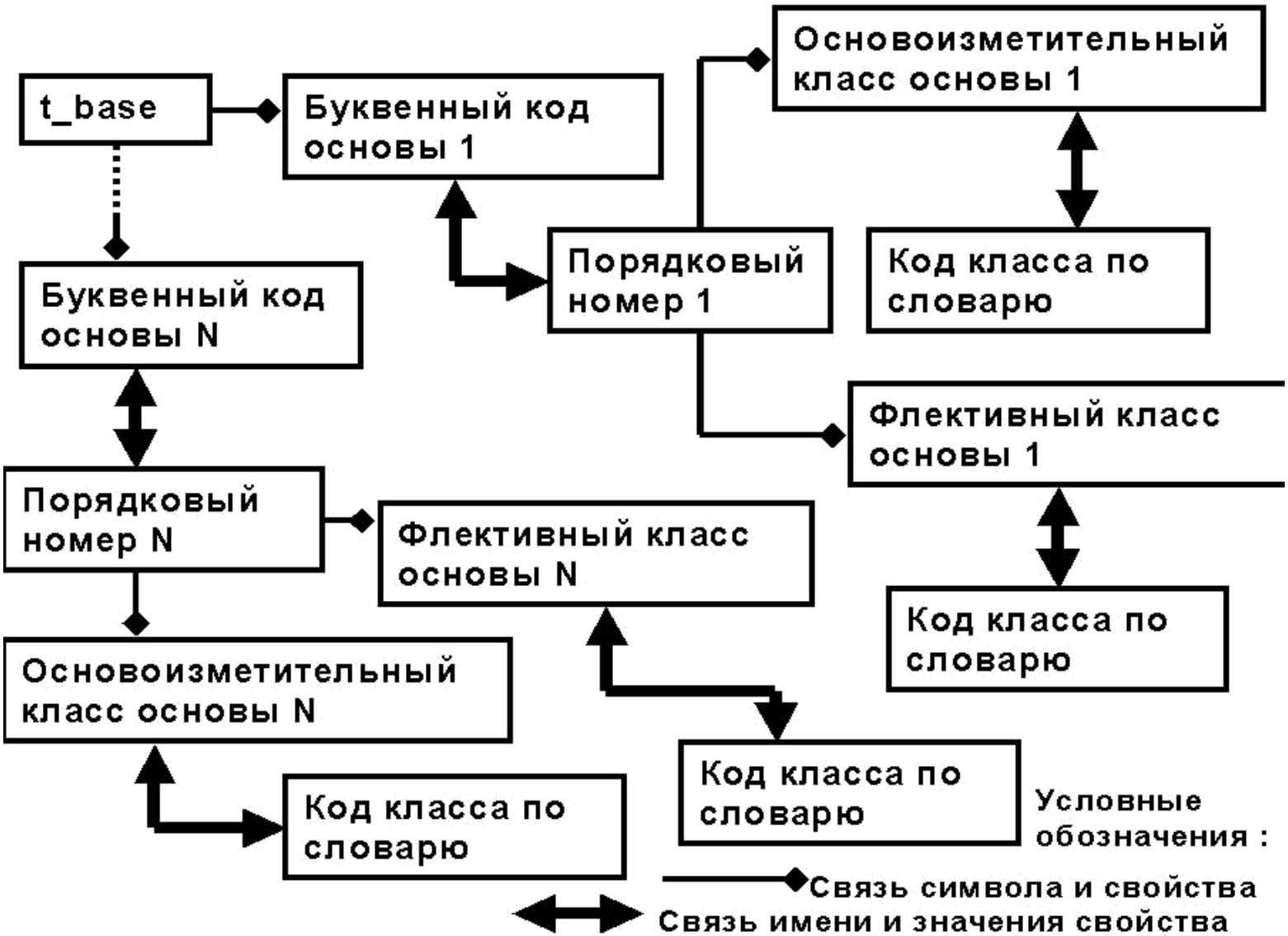
В силу вышесказанного морфологические классы слов принято подразделять на :

1). Основоизменительные классы, характеризующие систему изменения основ;

2). Флективные классы слов.

Флективные классы определены для изменяемых слов на основе анализа их синтаксической функции и систем падежных, личных и родовых окончаний. Флективный класс характеризуется либо системой признаков, либо словом-представителем.

Представление словаря основ с использованием списков свойств.



Этап 1 : считывание информации из внешних файлов.

В соответствие с особенностями ввода/вывода в Лиспе для облегчения последующего назначения свойств символам при формировании базы данных в памяти ЭВМ информацию словаря основ можно хранить в отдельных файлах : буквенных кодов основ (basesymb.txt), порядковых номеров основ (basenumb.txt), кодов основоизменительных классов (bchangcl.txt), кодов флективных классов (flect_cl.txt). Для записи компонент файлов мы создаем соответствующие списки. Рассмотрим пример для файла буквенных кодов основ.

```
(defun load_bases_symb_codes (file_name)
 (setq bases_symb_codes_list nil)
 (rds file_name)
 (loop
   ((not (listen)))
   (setq bases_symb_codes_list (cons (read-line)
bases_symb_codes_list))) (rds) bases_symb_codes_list)
```

Вызов : (load_bases_symb_codes basesymb.txt)

Этап 2 : построение списков свойств.

В соответствии с выбранной структурой для представления информации словаря основ описание каждого из свойств с множественным значением будет представляться списком : (<имя свойства> (<значение 1> … <значение N>)). Так, для основоизменительных классов мы будем иметь следующее описание :

(basechange_class (<основоизменительный класс основы 1>. . . <основоизменительный класс основы N>)), для флективных классов :

(flect_class (<флективный класс основы 1 > . . . <флективный класс основы N>)).

На основе считанных из файлов списков кодов основоизменительных (basechange_class_list) и флективных классов (flect_class_list) формируем списочное описание свойств с множественным значением :

(list (list ‘basechange_class basechange_class_list) (list ‘flect_class flect_class_list))

Построение списков свойств (продолжение).

1. Формирование БД с составной структурой свойств описываемых

2.объектов

```
(defun db_complex_props_make (db_name props vals vals_of_props_list)
; db_name - имя объекта
; props - список названий ключевых свойств объекта
; vals - список значений ключевых свойств объекта
; vals_of_props_list - списочное описание неключевых свойств объекта
;                    (<атрибут> <список значений атрибута>)
  ((null vals_of_props_list)(db_make db_name props vals))
  ((db_make db_name props vals)
        (complex_props_make vals vals_of_props_list))
)
```

Формирование БД в оперативной памяти.

```
; Функция формирования БД в оперативной памяти
(defun db_make (db_name props vals)
; Списки имен и занчений свойств имеют различную длину
 ((or
   (and    (null    props)(not    (null
  vals)))      (and      (not      (null
  props))(null vals)) ) nil)
; Условие окончания рекурсии
 ((and (null (cdr props))(null (cdr vals)))
 (put db_name (car props)(car vals)))
 ((put db_name (car props)(car vals))
  (db_make db_name (cdr props)(cdr vals)))
)
```

Формирование структуры свойств.

```
(defun complex_props_make (vals
 vals_of_props_list) ((and (null (cdr vals))
      (not (null vals_of_props_list)))
 (complex_prop_make (car vals) vals_of_props_list))
 ((complex_prop_make (car vals) vals_of_props_list)
  (complex_props_make (cdr vals)(vals_of_props_make vals_of_props_list))
 ))
```

Функция complex_props_make с помощью вспомогательной функции complex_prop_make для каждого символа из списка vals ставит в соответствие свойства и их значения из списка vals_of_props_list. Каждый элемент списка vals_of_props_list соответствует описанию списочному описанию свойства с множественным значением (см. слайд 3). В содержательной интерпретации список vals содержит описание значений ключевых свойств объектов (в нашем случае – это порядковые номера основ по словарю). Вспомогательная функция vals_of_props_make для каждого очередного объекта с уже сформированным списком свойств удаляет описания значений его свойств из списка vals_of_props_list.

Формирование элементов списка свойств отдельного объекта.

```
(defun complex_prop_make (val
 vals_of_props_list) ((null (cdr vals_of_props_list))
  (put val (caar vals_of_props_list)
           (caadar vals_of_props_list)
  ))
 ((put val (caar vals_of_props_list)
           (caadar vals_of_props_list))
  (complex_prop_make val (cdr vals_of_props_list)))
)
```

Удаление описания значений свойств очередного объекта.

```
(defun vals_of_props_make (vals_of_props_list)
 ((null vals_of_props_list) nil)
 (cons (list (caar vals_of_props_list)(cdadar vals_of_props_list))
       (vals_of_props_make (cdr vals_of_props_list))
 ))
```

Полный текст программы формирования БД словаря основ в оперативной памяти приводится в файле basedict.lsp.

Тестовый пример.

После загрузки программы : GCLisp.com basedict.lsp и последовательного вызова функций считывания информации из внешних файлов :

(load_bases_symb_codes basesymb.txt)

(load_bases_numbers basenumb.txt)

(load_basechange_classes bchangcl.txt)

(load_flect_classes flect_cl.txt)

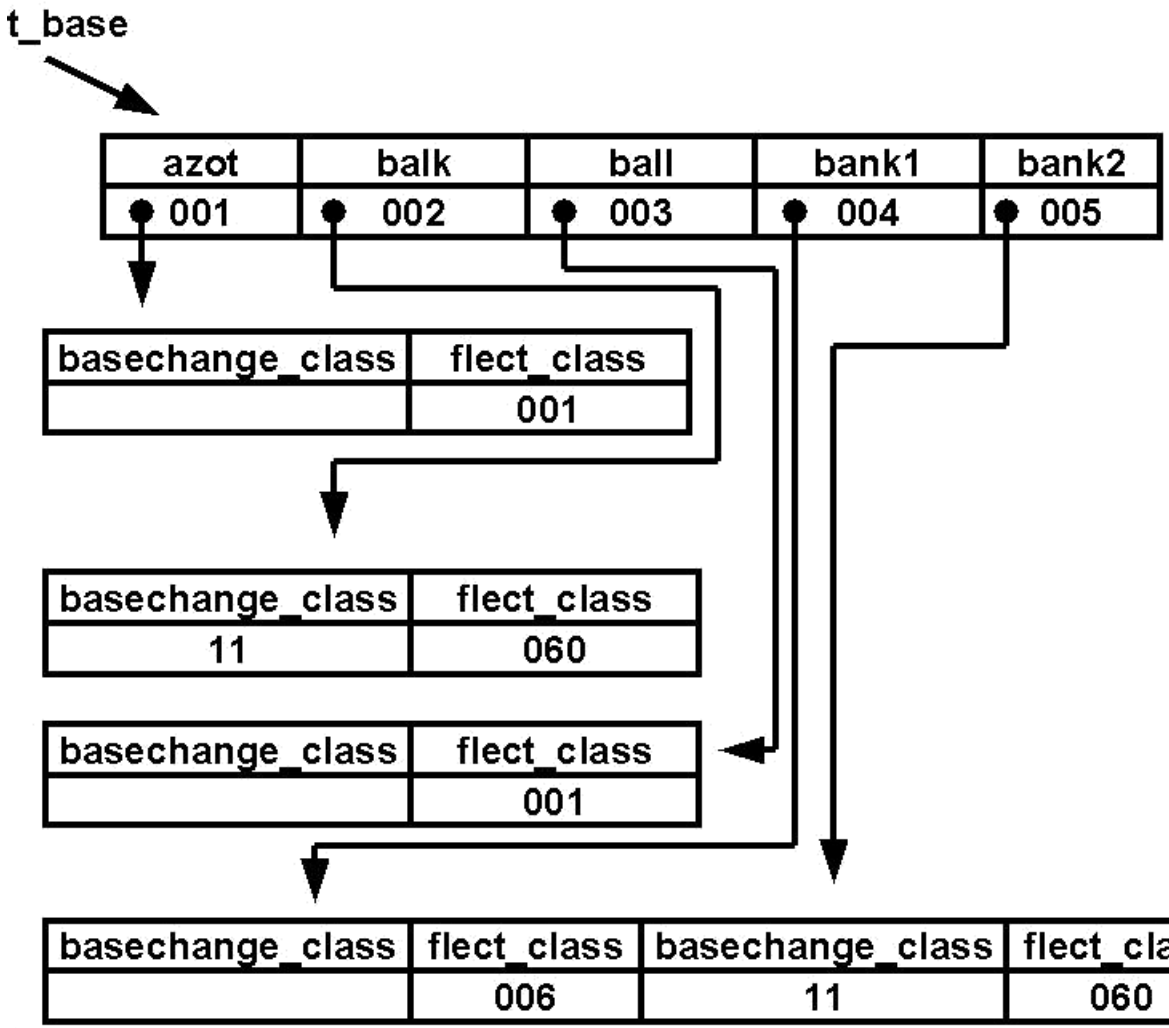
с помощью вызова функции test5 :

(test5 ‘t_base ‘basechange_class ‘flect_class) строится совокупность списков свойств в соответствии с приведенной на слайде 3 схемой.

Для просмотра свойств элементов созданной структуры достаточно вызвать функцию get из командной строки интерпретатора :

(get t_base “balk”) выдает номер основы по словарю основ, то есть “002”; (get “002” basechange_class) для основы с заданным номером выдает код основоизменительного класса, то есть “11”; (get “002” flect_class) для основы с заданным номером выдает трехразрядный восьмеричный код флективного класса, то есть “060”.

Сформированная БД для тестового примера.



Если слово имеет неизменяемую основу, то номер основоизменительного класса в словаре не указывается.

Применительно к табличной модели данных получаем, что имя символа соответствует либо названию таблицы (имени отношения), либо заглавию табличной строки , названия свойств – заглавиям столбцов, значения свойств – содержимому ячеек.

Строки как тип данных.

Строки относятся к простым типам данных. В Лиспе используется рассмотрение строк как одномерных массивов или векторов , элементами которых являются знаки. Многие более общие функции, определенные для массивов и последовательностей (чтение, сравнение элементов), наследуются строками.

Кроме этого, в Лиспе определен и ряд специальных функций для работы с этим типом данных :

- **Функции сравнения строк;**
- **Функции преобразования атомов и списков в строки и наоборот;**

Функции GCLisp’a по работе со строками.

Функция (findstring <строка 1> <строка 2>) показывает, начиная с какой позиции в строке 2 содержится строка 1 в качестве подстроки. Пример : (findstring ‘tri ‘string metter) выдает в качестве результата 1.

Функция pack преобразует список символов в строку. Пример : (pack ‘(s t r l n g m e t t e r)) выдает в качестве результата строку STRINGMETTER.

Функция print-length выдает в качестве результата длину строки до первого пробела. Пример : (print-length ‘string metter) возвращает в качестве результата 6.

Функция (string-left-trim <строка 1> <строка 2>) в случае, когда строка 2 начинается строкой 1, возвращает в качестве результата подстроку строки 2 (до первого пробела) после вхождения строки 1. В противном случае возвращается пустая строка . Пример : (string-left-trim ‘at ‘atom and list) возвращает в качестве результата строку OM.

Функции GCLisp’a по работе со строками.

Функция (char <строка> <n>) возвращает в качестве результата n-й символ строки, отсчет производится с 0. Пример : (char “cat Kuzya” 5) возвращает \u в качестве результата.

Функция (string= <строка 1> <строка 2>) возвращает результат сравнения строк, при этом строчные и заглавные буквы различаются.

Функция (substring <строка> <n> <m>) возвращает в качестве результата подстроку исходной строки, начиная с n-го и кончая m-м символом. Отсчет символов производится с 0. Аналогичная функция (slice <строка> <n> <m>) имеется в newLISP-tk, но аргумент <m> в ней – длина выделяемой подстроки.

Функция (unpack <строка>) преобразует строку в список символов , который возвращается в качестве результата. Пример : (unpack “Эксперимент на собаках”) возвращает в качестве результата список : (Э к с п е р и м е н т | | н а | | с о б а к а х). В newLISP-tk ей соответствует функция explode.

Литература.

1. Хювенен Э., Сеппянен Й. Мир Лиспа. Т.1. – М.:Мир, 1990. С. 61-63, 168-172 , 326-327, 337-340.

2. Белоногов Г.Г. и Богатырев В.И. Автоматизированные информационные системы. Под ред. К.В. Тараканова. – М.: Сов. радио, 1973.