УДК 004.9

Б. Л. Файфель

Саратовский государственный университет им. Н. Г. Чернышевского
ул. Астраханская, 89, Саратов, 410012, Россия

Главное управление Банка России по Саратовской области
ул. Советская, 2, Саратов, 410028, Россия
E-mail: catstail@narod.ru

HOMELISP – ПРОСТАЯ РЕАЛИЗАЦИЯ ЯЗЫКА ЛИСП 1.5 ДЛЯ ЦЕЛЕЙ ОБУЧЕНИЯ

HomeLisp – это простая реализация языка Лисп 1.5, предназначенная для обучения языку. Переменные в HomeLisp имеют динамическую область видимости. Ядро языка включает функции стандартного Лиспа, а также функции, обеспечивающие простейшую графику, диалоги, работу с двоичными данными, работу с системным реестром, файлами и COM-объектами. Интегрированная среда разработки содержит экранный дизайнер для разработки диалогов. В пакет HomeLisp входит также ActiveX-библиотека, позволяющая вызывать функции Лиспа из любой программной среды, поддерживающей COM, а также WEB-компонента для запуска Лиспа на WEB-сервере IIS. В статье описаны особенности реализации HomeLisp.

Ключевые слова: HomeLisp, Лисп, преподавание.

Введение

Изучение языка Лисп является необходимой составной частью образования грамотного программиста. Динамические структуры данных и рекурсию проще и естественнее изучать в Лиспе. Программы работы со списковыми структурами на Лиспе записываются гораздо лаконичнее, чем на Паскале или Си. Кроме того, Лисп может служить удобным полигоном для демонстрации по крайней мере двух парадигм программирования: операторной и функциональной. Этим можно объяснить, что учебные курсы по Лиспу входят в программы обучения практически всех ведущих ВУЗов мира, готовящих IT-специалистов.

Однако популяризация Лиспа в России сталкивается с неожиданной трудностью – отсутствуют простые реализации языка, пригодные для первоначального изучения новичками.

Программное средство, предназначенное для обучения, должно, по мнению автора, удовлетворять определенным требованиям. В их числе можно назвать следующие:

- бесплатность и доступность;
- внятный, простой и дружественный графический интерфейс;
- адаптация к самой массовой операционной системе – Microsoft Windows;
- обязательная возможность программирования задач, решаемых промышленными реализациями языка (графика, диалоги, файлы, использование COM, реестр системы).

Первое требование в современных условиях представляется вполне обоснованным и не требует особых комментариев. Что же касается интерфейса, то для начинающего программиста интерфейс не должен быть как слишком сложным («подавляющим» пользователя, как, например, в системе LispWorks), так и «минималистским» (muLisp, XLisp).

Представляется вполне разумным то обстоятельство, что операционная система Windows будет доминировать на современных компьютерах еще достаточно долгое время. В этих условиях переход на систему Linux, что обычно бывает продиктовано первым из требований, приведенных выше, ставит обучаемого в заведомо невыгодные условия: после завершения обучения ему с высокой степенью вероятности придется работать именно в Windows. Представляется нецелесообразным подход, при котором обучаемый после прохождения курса не будет иметь достаточного представления о программировании в самой массовой операционной системе.

И, наконец, трудно признать методически разумным подход, при котором рекурсия, динамические структуры данных, функционалы изучаются в Лиспе исключительно на абстрактных примерах [1; 2] [1], (не связанных с практическими приложениями). У обучаемого в этом случае возникает убеждение, что Лисп пригоден только для решения оторванных от повседневности задач. Подобное мнение никак не может способствовать популяризации идей Лиспа.

Все сказанное привело автора настоящего сообщения к мысли разработать собственную реализацию Лиспа в соответствии с приведенными выше принципами. За основу реализации была взята схема, описанная С. С. Лавровым и Г. С. Силагадзе в их работе [3], а в качестве языка реализации использовался Microsoft Visual Basic 6.0.

Схема Лиспа, описанная авторами [3], в свете современных воззрений не лишена некоторых недостатков. В их числе, вероятно, следует назвать динамическую область видимости переменных и отсутствие функций-макросов.

Оба недостатка не являются принципиальными. Так, если отказаться от хранения значений переменных в едином ассоциативном списке, то нетрудно реализовать лексическую область видимости (что дает возможность строить замыкания). Макросы также реализуются достаточно легко. В описываемой ниже версии HomeLisp [2] лексическая область видимости пока не реализована, а макросы реализованы без поддержки обратной блокировки.

Программные продукты подобного класса разрабатывались и раньше (см., например, [4]), но при реализации HomeLisp сделана попытка создать среду, в которой можно не только разрабатывать стандартные Windows-приложения с графическим интерфейсом, но и использовать возможности Windows в максимально полном объеме (файлы, графика, реестр, COM).

Краткое описание языка

Входной язык HomeLisp соответствует версии Лисп 1.5, но диапазон функций несколько расширен. Поддерживаются функции типа EXPR, MACRO и FEXPR. Реализована сборка мусора, автоматическим запуском которой можно управлять посредством настроек. Допустим и прямой вызов функции GC.

В качестве алфавита можно использовать все знаки (включая русские буквы). HomeLisp поддерживает следующие примитивные типы данных:

- целые числа неограниченной разрядности;
- числа с плавающей точкой двойной точности;
- битовые шкалы размером в 32 бита;
- символьные строки.

Кроме примитивных типов в ядро встроены сложные типы:

- графические окна;
- диалоги и элементы управления;
- файлы;
- большие двоичные объекты;
- COM-объекты.

При реализации HomeLisp автор счел целесообразным не использовать отдельные библиотеки, а включить все перечисленные выше возможности в ядро. Этот подход имеет как плюсы, так и минусы. Несомненным плюсом реализации принципа «всё в одном» является простота установки и настройки (когда для работы с графикой не нужно устанавливать одну дополнительную библиотеку, для создания диалоговых программ – другую и т. д.).

Среда разработки

Система включает простую среду разработки. Среда позволяет исполнять команды Лиспа (консоль), сохранять состояние Лисп-машины, использовать графические окна, разрабатывать диалоги и строить исполняемые файлы.

[1] Сайт информационных технологий Курганского университета: http://it.kgsu.ru/Lisp/oglav.html
[2] Сайт проекта HomeLisp: http://homelisp.ru

Консоль включает две области – область ввода и область отклика. Размеры областей можно менять. Следует отметить, что принятый подход несколько отличается от классических консолей Лиспа в других системах (обычно область ввода и область отклика – одно и то же окно). Выделение области ввода позволяет, на взгляд автора, более комфортно редактировать вводимые S-выражения. Поддерживается стек введенных пользователем команд.

В описываемой версии сохранение состояния выполняется в текстовый файл, в который заносятся все определенные пользователем функции, переменные и константы. Сохраненное состояние может быть впоследствии загружено (как и любой набор функций, содержащийся в текстовом файле).

HomeLisp позволяет строить исполняемые файлы типа «псевдо-EXE». Такой файл содержит двоичный код ядра Лиспа, необходимые функции и так называемое стартовое S-выражение. При запуске такого файла на исполнение сначала инициализируется ядро, затем загружаются все заданные программистом функции, после чего начинает выполняться стартовое S-выражение, из которого активизируется все приложение.

В качестве отладочных средств кроме традиционной трассировки используется дампирование – вывод в текстовый файл полной информации о процессе вычисления. Впоследствии эти текстовые файлы могут быть просмотрены.

Для облегчения программирования разработан аппарат шаблонов, позволяющий при редактировании S-выражения вставить вызов нужной функции.

Реализован специальный режим вычисления, позволяющий получить статистику вызовов функций всех видов при вычислении данного S-выражения. Ниже показывается, как можно использовать режим статистики на примере простой функции, вычисляющей сумму элементов одноуровнего числового списка. Используется рекурсивная функция sumlist и макро msumlist. При этом переменной Lst предварительно присваивается в качестве значения список чисел от 1 до 1 000:

```
(defun sumlist (x) (cond ((null x) 0)
        (t (+ (car x) (sumlist (cdr x))))))

(defmacro msumlist (x) (cons '+ (eval x)))
```

Ниже приводится результат вычисления обеих функций с подробной статистикой:

```
(sumlist Lst)

==> 500500 (затраченное время – 0.5 сек)

*** Статистика обращений ***
CAR ..... 1000
CDR ..... 1000
PLUS ..... 1000
SUMLIST ..... 1001
COND ..... 1001
NULL ..... 1001

(msumlist Lst)

==> 500500 (затраченное время – 0.05 сек)

*** Статистика обращений ***
EVAL ..... 1
CONS ..... 1
PLUS ..... 1
```

Видно, что макро вычисляется в данном случае значительно быстрее, поскольку требует всего три вычисления встроенных функций (тогда как классическое рекурсивное решение требует несколько тысяч вычислений).

HomeLisp как COM-сервер. Web-компонента

В поставку HomeLisp кроме описанной выше среды разработки входит ActiveX-библиотека, содержащая COM-объект «Lisp». Этой библиотекой можно пользоваться двояко: создавать объект «Lisp» из любой среды, поддерживающей COM-автоматизацию (например, C++, VB/VBA) или разрабатывать Лисп-скрипты, используя один из прилагаемых скриптовых интерпретаторов (WLISP.EXE или CLISP.EXE). Скриптовые интерпретаторы отличаются друг от друга средой исполнения: WLISP.EXE исполняется без создания консольного окна. Оба интерпретатора поддерживают все возможности HomeLisp.

Имеется возможность запускать HomeLisp на Web-сервере (IIS Microsoft). В этом случае (при наличии достаточно производительного сервера) возможно создание интранет-учебного класса для изучения Лиспа. Язык, реализованный в составе Web-компоненты HomeLisp, лишен ряда возможностей базового HomeLisp: в нем оставлены только операции со списками.

Графика

Графика HomeLisp включает простейшие примитивы (поставить точку, нарисовать прямую, прямоугольник, окружность, эллипс и т. д.) В качестве контейнера для графических операций служат графические окна, для создания которых имеется специальная функция. Графические окна можно создавать в любом разумном количестве. Свойства графических окон можно модифицировать, меняя соответствующие флаги в списке свойств окна посредством функции PUTPROP. Обеспечивается сохранение графики в форматах BMP и GIF, а также загрузка изображений из файлов формата BMP, GIF, JPG. Возможна работа с несколькими графическими окнами одновременно.

Ниже приводится исходный текст простейшей анимационной программы, в которой реализовано движение шарика в прямоугольном контейнере с отражением от стенок:

```
(grwCreate 'w 300 300 "Анимация" &HFFFFFF)
(grwShow 'w)
(grwScale 'w -105 105 -105 105)
 (prog (x1 y1 x2 y2 vx vy k p Clr n)
   (setq x1 0)
   (setq y1 0)
   (setq p 1.0)

 @cl ;; Выбор цвета шарика

   (setq n (rnd 4))
   (cond ((<= n 0) (go @cl)))
   (setq Clr (getel '(_RED _GREEN _BLUE _YELLOW) n))

 @vx ;; Выбор x-компоненты скорости

   (setq vx (* (rnd 10.0) p) )
   (cond ((<= (abs vx) 1.0) (go @vx)))

 @vy ;; Выбор y-компоненты скорости

   (setq vy (* (rnd 10.0) p) )
   (cond ((<= (abs vy) 1.0) (go @vy)))
```

```
 (grwSetParm 'w  ;; идентификатор окна
       1   ;; ширина линий
       0   ;; стиль заливки
      Clr   ;; цвет заливки
     _WHITE  ;; цвет переднего плана
 )
 (grwCircle 'w   ;; идентификатор окна
      x1   ;; X центра
      y1   ;; Y центра
      5   ;; Радиус
     Clr   ;; цвет
 )

Loop
 (setq x2 (+ x1 (* vx 3)))
 (setq y2 (+ y1 (* vy 3)))
 (grwSetParm 'w  ;; идентификатор окна
       1  ;; ширина линий
       0  ;; стиль заливки
     _WHITE  ;; цвет заливки
     _WHITE  ;; цвет переднего плана
 )
 (grwCircle 'w   ;; идентификатор окна
      x1   ;; X центра
      y1   ;; Y центра
      5   ;; Радиус
    _WHITE   ;; цвет
 )
 (grwSetParm 'w  ;; идентификатор окна
       1  ;; ширина линий
       0  ;; стиль заливки
      Clr  ;; цвет заливки
     _WHITE  ;; цвет переднего плана
 )
 (grwCircle 'w   // идентификатор окна
       x2  // X центра
       y2  // Y центра
       5  // Радиус
      Clr  // цвет
 )
 (grwRefresh 'w 3)

  ;; Обработка нажатия клавиш
  ;; “Esc” – выход;
  ;; “N” – новый шарик;
  ;; “+” - увеличить скорость;
  ;; “-“ – уменьшить скорость.

 (setq k (grwInkey 'w))
 (cond ((eq k 27) (return ‘OK!))
   ((eq k 109) (prog nil (setq p (* 0.75 p)) (setq vx (* p vx))
         (setq vy (* p vy)) ))
   ((eq k 107) (prog nil (setq p (* 1.25 p)) (setq vx (* p vx))
         (setq vy (* p vy)) ))
   ((eq k 78) (prog nil
```

```
        @vx
          (setq vx (* p (rnd 10.0)))
          (cond ((<= (abs vx) 1.0) (go @vx)))
        @vy
          (setq vy (* p (rnd 10.0)))
          (cond ((<= (abs vy) 1.0) (go @vy)))
        @cl
          (setq n (rnd 4))
          (cond ((<= n 0) (go @cl)))
          (setq Clr (getel '(_RED _GREEN _BLUE _YELLOW) n))
          (grwCls 'w)
          (setq x2 0)
          (setq y2 0)

         )
      )
    )

    ;; Проверка координат

    (cond ((greaterp x2 95) (setq vx (- 0 vx))))
    (cond ((greaterp y2 95) (setq vy (- 0 vy))))
    (cond ((lessp x2 -95) (setq vx (- 0 vx))))
    (cond ((lessp y2 -95) (setq vy (- 0 vy))))
    (setq x1 x2)
    (setq y1 y2)
    (go Loop)
  )
```

Диалоги

Диалог – это модальное окно Windows, на котором могут размещаться элементы управления следующих типов:

- метки;
- командные кнопки;
- поля ввода;
- списки (простые и выпадающие).

Для создания диалогов и размещения на нем элементов управления служат специальные функции. Свойства созданных диалогов и элементов управления можно модифицировать, меняя соответствующие флаги в списках свойств.

Каждому элементу управления можно назначить процедуру-событие (которая оформляется как функция Лиспа). В описываемой версии событийная модель очень проста: для меток, списков и кнопок поддерживается событие CLICK, для полей ввода – KEY_DOWN.

Диалог может быть создан как путем вызова соответствующих функций, так и с помощью простого экранного дизайнера. Дизайнер позволяет создать окно диалога, разместить на нем элементы управления, а затем сгенерировать Лисп-код, выполняющий эти действия. На рис. 1 показан вид экранного дизайнера с формой диалога вычисления НОД (наибольшего общего делителя) двух целых. На рис. 2 приведено задание процедуры события для кнопки «Вычислить» с рис. 1.

Сравнивая код, генерируемый дизайнером HomeLisp для работы с диалогом, с кодом, который требуется для организации диалогов в системе X-Windows (с библиотекой Tk [3]), можно отметить, что объемы кода и трудоемкость написания вполне сопоставимы.

[3] См. цикл статей В. В. Водолазкого: http://lisp.ru/page.php?id=4

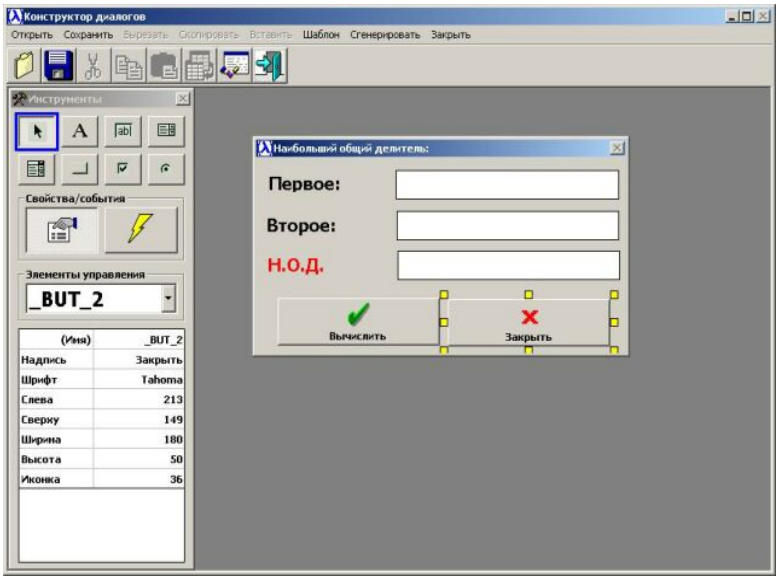

Рис. 1. Общий вид экранного дизайнера

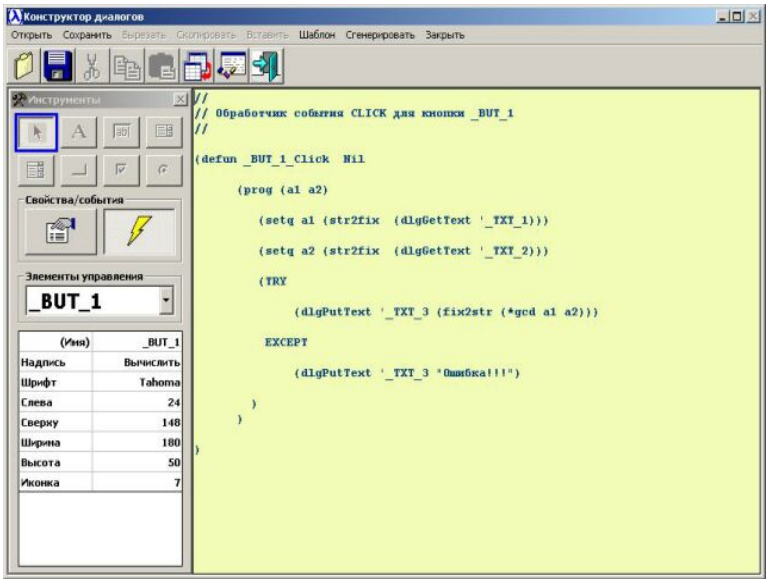

Рис. 2. Задание процедуры-события

Файлы, большие двоичные объекты, COM-объекты

Работа с файлами реализована в HomeLisp традиционно: поддерживается как последовательный метод доступа (текстовые файлы), так и прямой метод доступа (двоичные файлы). Ниже приведен пример, в котором содержимое файлов с заданным расширением собирается в файле «a.out»:

```
;; печать содержимого директории

(defun printd (ext)
  (prog nil
   (cond ((filexistp "a.out") (sysErase "a.out")))
```

```
  (mapcar (sysdir (strCat (syshome) "\" ext) &H3F) 'printf)
 )
)

;; Печать одного файла

(defun printf (f)
     (prog (fi fo)

      (filOpen 'fi  ;; идентификатор файла
          f   ;; имя файла
          _INPUT ;; метод доступа
      )

      (filOpen 'fo   ;; идентификатор файла
          "a.out" ;; имя файла
          _APPEND ;; метод доступа
      )

      (filPutLine 'fo "***")
      (filPutLine 'fo (strcat "Файл: " f))
      (filPutLine 'fo "***")

    @loop
      (filPutLine 'fo (filGetLine 'fi))
      (cond ((filEOF 'fi) (go @Clo))
        ( t (go @loop))
      )
    @Clo
      (filClose 'fi)
      (filPutLine 'fo "")
      (filPutLine 'fo "")
      (filClose 'fo)
      (return f)
)    )
```

Для решения задачи применен классический отображающий функционал mapcar. Его первым аргументом является список имен файлов, формируемый функцией sysDir, а вторым аргументом – функция printf, организующая печать отдельного файла.

Большие двоичные объекты (BLOBы) – это динамически выделяемые области памяти. Доступ к содержимому областей реализован посредством набора специальных функций. Имеется возможность загружать в BLOB информацию из двоичного файла; реализован символьно-шестнадцатеричный дамп. Ниже показывается, как с помощью файлов и BLOBов можно исследовать структуру двоичных файлов.

```
(filOpen 'fi "pkzip.exe" _BINARY_READ)  ;; открытие двоичного файла
(bloCreate 'b1 300)                     ;; создание BLOBа размеров 300 байтов
(filGetBlo 'fi 'b1)                     ;; загрузка данных в BLOB из файла
(bloDump* 'b1)                                ;; дамп
(filClose 'fi)                          ;; закрытие файла

00000001 | 4D 5A B6 00 53 00 01 00 | 06 00 4E 08 FF FF 4C 0A | MZ¶.S.....N.яяL. |
00000017 | 00 02 00 00 00 01 F0 FF | 52 00 00 00 14 31 50 4B | ......рR....1PK |
00000033 | 4C 49 54 45 20 43 6F 70 | 72 2E 20 31 39 39 30 2D | LITE Copr. 1990- |
00000049 | 39 32 20 50 4B 57 41 52 | 45 20 49 6E 63 2E 20 41 | 92 PKWARE Inc. A |
```

```
00000065 | 6C 6C 20 52 69 67 68 74 | 73 20 52 65 73 65 72 76 | ll Rights Reserv |
00000081 | 65 64 07 00 00 00 00 00 | 00 00 00 00 00 00 00 00 | ed..............  |
00000097 | B8 87 12 BA 46 0A 05 00 | 00 3B 06 02 00 72 1D B4 | ё‡.єF....;...r.r |
00000113 | 09 BA 1A 01 CD 21 B4 4C | CD 21 4E 6F 74 20 65 6E | .є..H!rLH!Not en |
00000129 | 6F 75 67 68 20 6D 65 6D | 6F 72 79 24 2D 20 00 FA | ough memory$- .ъ |
```

HomeLisp позволяет создавать COM-объекты, использовать их свойства и методы. Эта возможность языка позволяет практически в полной мере использовать преимущества COM-модели. В частности, имеется встроенная функция, позволяющая получить интерфейс COM-объекта в виде лисповского списка. Вот как выглядит, например, COM-интерфейс хорошо известного объекта «Word.Application»:

```
(comInterface “Word.Application”)  ;; запрос интерфейса

((QueryInterface Method VOID ("riid" EMPTY "ppvObj" VOID))
 (AddRef Method UI4 NIL)
 (Release Method UI4 NIL)
 (GetTypeInfoCount Method VOID ("pctinfo" UINT))
 (GetTypeInfo Method VOID ("itinfo" UINT "lcid" UI4 "pptinfo" VOID))
…
 (Parent Property-GET DISPATCH NIL)
 (Name Property-GET BSTR NIL)
 (Documents Property-GET EMPTY NIL)
 (Windows Property-GET EMPTY NIL)
 (ActiveDocument Property-GET EMPTY NIL)
…
)
```

Детали реализации

При реализации Лиспа в оперативной памяти традиционно выделяются две области – область хранения информационных ячеек атомов (так называемый список объектов) и область хранения списочных ячеек. Списочные ячейки содержат байт флагов и два указателя (CAR и CDR). Внутренняя структура атома тоже включает два указателя (первый – на список, задающий имя атома, и второй – на список свойств атома), байт флагов и, возможно, дополнительную информацию.

Главным (но не единственным) назначением байта флагов является возможность помечать объекты, которые не используются в данный момент времени. Эта возможность используется при сборке мусора.

Поскольку доступ к атому (символу) осуществляется по его имени, необходимы средства поиска атомов в ассоциативном списке. Для ускорения поиска обычно используют hash-таблицы.

Таким образом, главным моментом при реализации Лиспа является внутреннее представление списочной ячейки, атома и списка объектов. Поскольку Visual Basic поддерживает объекты, было принято решение использовать объектный подход: и атом, и списочная ячейка являются объектами, создающимися на основе соответствующих классов (clsCell – класс, описывающий списочную ячейку, и clsInfCell – класс, описывающий информационную ячейку атома). Очевидным плюсом объектного подхода является то обстоятельство, что в Visual Basic имеется встроенная сборка мусора при работе с объектами, что облегчает (но не отменяет) реализацию сборки мусора в HomeLisp.

Поскольку как внутренние функции (реализующие программную логику ядра), так и пользовательские функции в качестве параметров могут принимать объекты обоих типов (clsCell и clsInfCell), возникает проблема, так как Visual Basic – язык со строгой типизацией данных. Один из способов обойти проблему типизации состоит в том, чтобы не типизировать параметры явно, а описывать их как абстрактные объекты (так называемое позднее связыва-

ние). Подобный подход прост, но заметно снижает производительность, поэтому было принято решение разработать абстрактный класс aCell и реализовать его интерфейс в классах clsCell и clsInfCell. При разработке функций все параметры, задающие объекты Лиспа, имеют тип aCell.

Важнейшими полями информационной ячейки атома являются:

- поле имени (строка);
- поле стандартных флагов (4 байта);
- ссылка на определяющее выражение (для функций и констант);
- однобайтовый тэг (биты которого используются для служебных целей).

Для хранения атомов в HomeLisp используется коллекция – встроенный в Visual Basic тип данных. В качестве ключа поиска используется имя атома (поле класса clsInfCell). Данные всех типов HomeLisp, описанные выше, хранятся в виде символьных строк, задающих имена соответствующих атомов. Некоторое неудобство возникает при хранении данных строкового типа. Дело в том, что при сравнении ключей в коллекции Visual Basic не различаются строчные и заглавные буквы. Это приводит к тому, что атом a и A есть один и тот же атом (что соответствует принятым стандартам). Однако строки «a» и «A» не должны являться одинаковыми. Для этого при занесении строки (цепочки символов, заключенной в двойные кавычки) в информационную ячейку атома перед заглавными буквами помещаются специальные маркеры и полученная новая строка используется в качестве ключа при занесении атома в коллекцию.

Для создания нового атома или для получения ссылки на существующий атом имеется функция creaAtom, которая берет на вход строку, задающую атом и возвращает ссылку на информационную ячейку этого атома. Функция проверяет наличие в списке объектов данного атома и, при наличии, возвращает ссылку на этот атом. Если атом отсутствует в списке объектов, то атом создается, заносится в список объектов, а ссылка на атом возвращается функцией как результат.

Поскольку менеджер памяти Visual Basic автоматически удаляет неиспользуемые объекты, то большая часть списочных ячеек утилизируется автоматически (когда уничтожаются все ссылки на эти ячейки). Исключение составляют только неиспользуемые циклические структуры.

С атомами все обстоит сложнее. Поскольку, как следует из сказанного выше, на каждый атом имеется по крайней мере одна ссылка (из списка объектов), менеджер памяти Visual Basic никогда не удаляет неиспользуемые атомы – это должна делать функция сборки мусора (GC).

Алгоритм сборки мусора в HomeLisp достаточно традиционный и состоит из двух этапов – разметки и удаления неотмеченных ячеек и атомов. Для разметки используется младший бит поля тэга (как информационной, так и списочной ячеек).

Чтобы утилизировать память, занятую циклическими структурами, необходим способ обхода всех списочных ячеек. В ранних реализациях Лиспа списочные ячейки хранились в линейных массивах, и проблемы обхода не существовало. Совокупность объектов Visual Basic не является массивом, что создает трудности при обходе всех созданных объектов. Если собрать все списочные ячейки в коллекцию, то это запретит автоматическую утилизацию неиспользованных линейных структур (поскольку на каждую ячейку будет существовать по крайней мере одна ссылка). Чтобы обойти эту проблему, был применен искусственный прием: в коллекцию собираются не ссылки на списочные ячейки, а их реальные адреса в оперативной памяти, полученные вызовом недокументированной функции ObjPtr.

Списки свойств атомов в HomeLisp реализованы следующим образом. Стандартные свойства (EXPR, FEXPR, MACRO, SUBR, FSUBR, FIXED, FLOAT, STRING, BIT и др.) моделируются битовыми флагами информационной ячейки атома, для чего выделено четырехбайтовое поле. Нестандартные свойства реализованы как обычные списки. Встроенная в ядро функция PROPLIST возвращает традиционный список, в котором сначала идут стандартные свойства, а затем (при наличии) – нестандартные.

Любую списочную конструкцию можно защитить от модификации, объявив константой (для чего служит функция CSETQ). Защита реализована с использованием второго бита поля тэга (первый используется программой GC).

Переменные HomeLisp (символы, имеющие значения) хранятся в ассоциативном списке. В соответствии с рекомендациями [1] ассоциативный список реализован как стек пар «атом – значение». Перед вызовом функции Лиспа связанные переменные со своими значениями помещаются в ассоциативный список, а после выхода удаляются из него. Аналогичные действия выполняются перед входом в PROG-конструкцию, но в ассоциативный список заносятся пары «атом – Nil».

Функция EVATOM, берущая на вход атом, просматривает ассоциативный список от конца к началу и ищет первую пару, у которой атом-имя совпадает с атомом, поданным на вход. Возвращается соответствующее значение. Такая программная логика позволяет одновременно существовать нескольким связям одного атома (но активной связью является, естественно, самая последняя связь).

Основной цикл работы Лисп системы состоит в чтении из какого-либо источника очередного S-выражения, перевода его во внутреннюю списочную структуру и передаче этой структуры на вход функции EVAL. Чтение S-выражения выполняет программа-парсер. Она устроена достаточно традиционно: входное выражение разбивается на лексемы (атомы и круглые скобки), а затем из атомов и скобок создается список Лиспа.

Функция EVAL анализирует S-выражение, поданное на вход, и в зависимости от типа выражения выполняет следующее:

- если S-выражение есть атом, то вызывается функция EVATOM;
- если S-выражение есть список, то голова списка рассматривается как имя функции и вызывается функция APPLY, на вход которой передаются два параметра – голова исходного списка и его остаток (который рассматривается как список аргументов);
- в остальных случаях возбуждается состояние ошибки.

Функция EVATOM проверяет, является ли ее аргумент:

- самоопределенной константой (типа FIXED, BIT, FLOAT, STRING). В этом случае функция возвращает свой аргумент в качестве результата;
- константой, определенной пользователем, тогда функция возвращает определяющее выражение константы из соответствующего поля информационной ячейки аргумента;
- атомом, не являющимся константой. Выполняется поиск последней связи атома в ассоциативном списке. Если связь найдена – возвращается значение, связанное с атомом. В противном случае возбуждается состояние ошибки.

Функция APPLY проверяет первый аргумент. Он может быть либо атомом, либо лямбда-выражением. Если первый аргумент APPLY есть атом, то оба аргумента APPLY передаются функции EVFUN. Если первый аргумент APPLY есть лямбда-выражение, то строится обращение к функции EVLAM. При этом первый аргумент (само лямбда-выражение) остается неизменным, а каждый элемент списка второго аргумента заменяется значением (т. е. к каждому элементу списка рекурсивно применяется функция EVAL).

Функция EVFUN определяет тип вызываемой функции (по наличию у первого аргумента одного из стандартных флагов SUBR, FSUBR, EXPR, FEXPR или MACRO). Для случаев SUBR и EXPR каждый элемент списка, составляющего второй аргумент заменяется своим значением (как описано выше). Для случаев FSUBR, FEXPR, MACRO замены значениями не происходит. Затем для флагов SUBR, FSUBR вызывается соответствующая встроенная функция (через функцию-посредник CALL_FSUBR, которой передается имя функции и список параметров). Для флагов EXPR, FEXPR, MACRO функция EVFUN извлекает из поля информационной ячейки определяющее лямбда-выражение функции и вызывает функцию EVLAM. При этом случай MACRO имеет особенность – результат вычисления EVLAM передается еще раз на вход EVAL.

Очень серьезной проблемой при реализации Лиспа является обработка ошибок и восстановление состояния после возникновения ошибки. К сожалению, авторы работы [3] не уделили этой проблеме должного внимания. В HomeLisp у каждой функции, входящей в ядро, имеется стандартный пролог, в котором анализируется значение глобального флага flgError. Если значение flgError есть TRUE, то происходит выход из функции с возвратом в качестве результата предопределенного атома ErrState (состояние ошибки). При возникновении ошибки любая функция обязана установить глобальный флаг flgError и вернуть в качестве результата атом ErrState. Такой подход приводит к тому, что любая ошибка (на любом уровне вложенности) вызывает мгновенный выход из самой «внешней» EVAL.

Аналогично решается проблема зацикливания – имеется глобальный флаг flgBreak, который поднимается, если заданное в конфигурации время вычислений истекло. Если флаг при

входе в функцию поднят, функция обязана немедленно вернуть предопределенный атом BrkState (состояние прерывания). При работе в среде разработки имеется кнопка, нажатие на которую вызывает установку флага flgBreak, что дает возможность прервать выполнение вычислений.

В HomeLisp встроена функция STOP*, позволяющая программно остановить вычисления. Поскольку эта функция может быть вызвана на любой глубине вложенности, имеется глобальный флаг flgStop и соответствующий атом StopState. Любая функция, входящая в ядро, в прологе анализирует флаг flgStop; если флаг поднят, функция обязана завершиться, вернув в качестве результата атом StopState.

Еще одной функцией стандартного пролога является опрос флага дампирования – flgDump. Если этот флаг установлен, то в заранее открытый текстовый файл (dump-файл) выводятся значения входных параметров и состояние ассоциативного списка.

Кроме пролога каждая функция имеет стандартный эпилог. В эпилоге происходит вывод в dump-файл результата, который возвратит функция (при установке флага дампирования).

Для расширенной обработки ошибок служит функция TRY (отсутствующая в реализации, описанной в [3]). Эта функция позволяет выполнить критический участок кода и, при возникновении ошибки, сбросить состояние ошибки и выполнить код-обработчик ошибки.

Арифметика целых чисел в HomeLisp реализована на символьных строках, что позволяет работать с целыми числами практически неограниченной разрядности и диапазоном, значительно более широким, чем стандартный DOUBLE.

Заключение

HomeLisp был представлен на школе-семинаре «Искусство программирования», проведенного Новосибирским государственным университетом 05.11.2010.

Автор планирует в ближайшее время выпустить следующую версию HomeLisp, в которой будут реализованы лексические переменные, замыкания, обратная блокировка в макросах, необязательные и ключевые параметры у функций пользователя и ряд других возможностей CommonLisp.

В заключение автор выражает искреннюю благодарность Л. В. Городней и Н. В. Шилову за интерес к работе и доброжелательную критику.

Список литературы

1. *Морозов М. Н.* Курс лекций по функциональному программированию/ URL: http:// www.mari.ru/mmlab/home/lisp

2. *Тужилов И. В.* Язык программирования XLISP. Пенза, 1994.

3. *Лавров С. С., Силагадзе Г. С.* Автоматическая обработка списков – язык лисп и его реализация. М.: Наука, 1978.

4. *Дацун Н. Н., Хован А. П.* Internet-учебный курс для обучения языку Лисп // Образование и виртуальность-2001: Сб. науч. тр. Харьков; Ялта, 2001. С. 153–161.

Материал поступил в редколлегию 23.04.2012

B. L. Fayfel

HOMELISP – THE SIMPLE IMPLEMENTATION OF LISP 1.5 FOR EDUCATION

HomeLisp is a simple implementation of Lisp 1.5, designed for language learning. Variables in HomeLisp have dynamic scope. Core language includes features of standard Lisp, as well as features that provide simple graphics, dialogues, work with binary data, work with the system registry, working with files and COM-objects. Integrated development environment includes a display designer to develop a dialogue. The package also includes HomeLisp ActiveX-library that allows to call functions from anywhere in the Lisp programming environment that supports COM, as well as WEB-component to run Lisp on the WEB-server IIS. In article features of realization HomeLisp are described.

Keywords: HomeLisp, Lisp, education.