МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ

КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И. РАЗЗАКОВА

ТОКМОКСКИЙ ТЕХНИЧЕСКИЙ ИНСТИТУТ

Кафедра «Программное обеспечение компьютерных систем»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО ЯЗЫКУ NEWLISP ПО ДИСЦИПЛИНЕ “ФУНКЦИОНАЛЬНОЕ И ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ ”

для студентов специальности 552801.04 – «Программное обеспечение вычислительной техники и автоматизированных систем»

Бишкек – 2011

«Рассмотрено»	**«Одобрено»**
на заседании кафедры ПОКС	на заседании УМС ТТИ
Прот. № 7 от 05.02.2011 г.	Прот. № 8 от 27.04.2011 г.

Составители: ЕШПУЛАТОВ С.Е,КЛЕМЕШЕВА., О.В.

Методические указания к выполнению лабораторных работ по языку NewLISP по дисциплине “Функциональное и логическое программирование. / ТТИ КГТУ им. И. Раззакова; сост.: С.Е. Ешпулатов, О.В. Клемешева. – Б.: ИЦ “Техник”, 2011. – 31 с.

Предлагаемая работа является развернутым руководством, содержащим рекомендации по выполнению лабораторных работ для студентов, обучающихся по специальности 552801.04 – “Программное обеспечение вычислительной техники и автоматизированных систем”. В данном руководстве приведены требования к подготовке и выполнению лабораторных работ, порядок защиты и примеры их выполения.

Рецензент к.т.н., доц. К.Дж. Боскебеев

Иллюстрации: 2.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

В данных методических указаниях рассматривается язык программирования NewLISP, предназначенный для символьной обработки данных. Язык NewLISP используется для создания экспертных систем, в задачах искусственного интеллекта, функционирующих на персональных компьютерах.

В данных указаниях рассматривается применение языка NewLISP для решения некоторых типичных простых задач и задач средней трудности невычислительного характера и символьной обработки, приводится также теоретический материал, описание всех используемых функций, конкретные примеры задач, их решения и результаты выполнения. Анализируется также применение языка NewLISP для обработки некоторых часто используемых структур данных для решения конкретных задач символьной обработки.

В связи с этим и было принято решение о разработке методических указаний для студентов, обучающихся по специальности 552801.04 – “Программное обеспечение вычислительной техники и автоматизированных систем”, позволяющих решать различные задачи в сфере функционального программирования на языке NewLISP.

Содержание

Введение

Лисп – интерпретируемый бестиповой язык символьной обработки. Сходство с машинным языком: единая форма представления данных и программ. Близость к естественному языку обеспечивается за счет декларативности ЛИСП – программ. Наряду с типичными для “декларативных” языков средств, Лисп допускает применение некоторых структур данных алгоритмических языков, не допустимых в языке логического программирования, в частности, массивов, а также имеет встроенные “ассемблерные” функции.

К основным преимуществам языков функционального программирования можно отнести следующее:

- ✓ краткость программы;
- ✓ функциональные программы поддаются формальному анализу легче своих аналогов на алгоритмических языках за счет использования математической функции в качестве основной конструкции.

Языки функционального программирования применяются в следующих сферах:

- ✓ системы автоматизированного проектирования;
- ✓ программирование;
- ✓ математическая лингвистика.

Функциональным называется программирование при помощи функций в математическом их понимании. Функциональное программирование основано на следующей идее: в результате каждого действия возникает значение, которое может быть аргументом следующего действия. Программы строятся из логически расчлененных определений функций. Каждое определение функции состоит из организующих вычисления управляющих структур и из вложенных, в том числе вызывающих самих себя (рекурсивных) вызовов функций.

NewLISP — это свободный скриптовый язык общего назначения для задач, обычно решающихся с помощью интерпретируемых языков, таких как, программирование для интернет, системное администрирование, обработка текста, соединение различных программ и модулей в единое приложение.

Возможности NewLISP:

- ✓ множество математических, статистических функций, работа с файловым вводом-выводом, датой и временем;
- ✓ поддержка Unicode наличие в комплекте IDE (newLISP-GS);
- ✓ вызов функций из динамических библиотек (.so, .dll и т.д.);
- ✓ реализация основных принципов ООП с помощью концепции контекстов (пространств имён).

NewLISP ближе к обычному языку обработки списков Лисп, но отличен в некоторых аспектах от диалектов языка обработки списков Лисп. В большинст-

ве случаев обозначение и работа в функциях подобны ANSI обычному языку обработки списков Лисп. Некоторые обозначения функций и их применение заимствованы от Scheme, а другие от языка C. NewLISP является сравнительно малым чем Scheme, но тем не менее в нем можно реализовать более 230 функций. Последние версии NewLISP для LINUX далеко ушли в развитии в отличие от первоначальной версии Windows. Небольшой размер приблизительно 100 КБ, выполнимый из NewLISP, использует только самая обычная система UNIX C-библиотеки, имеет очень быстрое время загрузки и маленькую память для печати. Это конкурирует в скорости с другими популярными языками описания и использует очень небольшие вычислительные ресурсы.

NewLISP – динамический, подобно первоначальному языку обработки списков Лисп. Для объекта сориентировавшего программирование и пакетирование локальных состояний и процедур NewLISP вводит переменные lambda - выражения. Функции (выражения Лямбды) в NewLISP реализовываются подобно любому другому выражению списка. Подобно Scheme NewLISP также оценивает элемент оператора выражения списка. Программы в NewLISP используют различные имена или контексты. Это облегчает разработку больших приложений в NewLISP, составленных из различных независимых развитых модулей с их собственными именами. Коррекция локальных переменных в выражениях лямбды рассматривают как часть операции, которая имеет место, если функция закончена без ошибок. Начиная с версии 6.3, содержание локальных переменных сохраняется непосредственно в списке лямбды. Это позволяет экономить лямбду и анонимные выражения лямбды, поскольку постоянные объекты к файлу посредством простого экономят построенные функции. NewLISP расширяется через разделенный интерфейс библиотеки. Функции от разделенных библиотек легко импортированы. Гнездо интерфейса TCP/IP NewLISP облегчает написание сетевых приложений. Простые примеры, при написании клиентов сети и серверов включены процессы текстового средства обслуживания NewLISP и включают правильное соответствие образца выражения и функции анализа текста. Построенный в интерфейсе XML делает NewLISP полезным инструментом для CGI, обрабатывающего на серверах сети. Пример для обработки форм HTML и простого сервера сети, написанного в NewLISP, также включен в это распределение.

Некоторые сведения о часто используемых функциях:

and	Синтаксис: (and *exp-1 exp-2* [*exp-3*...])	Вычисляет выражения слева направо, пока не происходит результат nil, при этом возвращает nil. Если ни одно из выражений не дает результат nil, то возвращается результат последнего выражения.
define	Синтаксис: (define (*sym-name* [*sym-param-1* ...]) [*body-1*]) Синтаксис: (define *sym-name exp*)	Определяет новую функцию *sym-name* с дополнительными параметрами *sym-param-1*... . Функция define эквивалентна связыванию выражения лямбды с символом. При запросе определенной функции, все параметры вычисляются и связываются с переменными *sym-param-1*..., затем вычисляется тело *body-1*... . При определении функции возвращается лямбда выражения, содержащегося в *sym-name*. Все определенные параметры являются дополнительными. При запросе определенной функции без указания значений параметров, эти параметры примут значения nil.
find	Синтаксис: (find exp-key list)	Ищет искомый элемент в списке.
for	Синтаксис: (for (sym *num-from num-to* [*num-step*]) *body*)	Повторяет выражение или выражения в *body* в количестве диапазона значений, указанных в *num-from* и *num-to*. Может быть определен размер шага *num-step*. Если размер шага не указан, то принимается 1. Возвращает последний результат вычисления выражений в *body*.
if	Синтаксис: (if *exp-condition exp-1* [*exp-2*])	В зависимости от результата вычисления *exp-condition*, вычисляется либо выражение *exp-1*, либо выражение *exp-2*, если оно определено, и возвращается результат вычисления одного из

		этих выражений. Если выражение *exp-condition* не является пустым списком () и не равно nil, то в этом случае вычисляется выражение *exp-1*, в противном же случае – выражение *exp-2*.
join	Синтаксис: (join list [string]).	Соединяет строки в списке.
last	Синтаксис: syntax: (last list)	Возвращает последний элемент списка.
length	Синтаксис: (length *list*)	Определяет длину списка.
nth	Синтаксис: (nth *num-offset list*)	Находит элемент в списке *list*, имеющий индекс *num-offset*. Первый элемент будет иметь индекс 0, второй – индекс 1 и т.д. Последний элемент списка возвращается в том случае, если индекс находится вне диапазона.
open	Синтаксис: (open str-path-file str-access-mode)	Предназначена для открытия конкретного файла
parse	Синтаксис: (parse str-data [str-break])	Функция parse принимает единственный аргумент типа string, содержащий алгебраическое выражение в естественной нотации, а возвращает это же выражение, разбитое на лексемы. Круглые скобки становятся структурными скобками результирующего S-выражения. Лексемами являются символические имена переменных, числовые константы и знаки алгебраических операций.
pop	Синтаксис: (pop *list* [*int-offset*])	Удаляет первый элемент из списка *list* и возвращает удаляемый элемент. Если указан второй параметр, то удаляется элемент, имеющий положение *int-offset* в списке. Если указанный индекс больше по величине, чем длина списка или равен длине списка, то удаляется последний элемент.
print	Синтаксис:	Выводит на текущее устройство

	(print exp-1 [*exp-2 ...*]), где exp-1 – выражение	ввода-вывода все результаты вычислений выражений *exp-1*.... Чтобы завершать печать переводом строки, используется функция println.
println	Синтаксис: (println *exp-1* [*exp-2 ...*])	Выводит на текущее устройство ввода-вывода все результаты вычислений выражений *exp-1*... В конец печатается перевод строки. Функция println в остальном работает аналогично функции print.
push	Синтаксис: (push *exp list* [*int-offset*])	Элемент *exp* вставляется в список *list*. Если определен параметр *int-offset*, то элемент вставляется в указанной позиции в списке, если параметр не определен, то элемент вставляется в начало списка. После вставки, возвращается вставленный элемент. Если значение *int-offset* больше или равно длине списка или отрицательное число, то элемент вставляется в конец списка.
read-line	Синтаксис: (read-line [*int-file*])	Считывает строку из текущего устройства ввода-вывода, строки разграничены символом перевода строки. Нет ограничения к длине строки. Символ перевода строки не является частью возвращаемой строки. По умолчанию текущее устройство – клавиатура (устройство 0). Также может быть определен дескриптор файла *int-file*, который получен при последней операции open.
replace-nth	Синтаксис: (replace-nth int-nth list exp-replacement)	Заменяет nth в int-nth-ом элементе в списке с оценкой exp-replacement и возвращает элемент, который был заменен.
reverse	Синтаксис: (reverse list)	Изменяет порядок элементов на

		обратный.
set	Синтаксис: (set *sym-variable exp*)	Вычисляет оба параметра, затем присваивают результат выражения *exp* символу *sym-variable*. Выражение возвращает результат присваивания. Старое содержание символа удаляется.
sort	Синтаксис: (sort list)	Это функция, которая производит сортировку элементов.
substring	Синтаксис: (substring str int-offset [int-length])	Определяет подстроку заданной строки.
while	Синтаксис: (while exp-condition body)	Если результат не является нулевым или пустой список ( ), выражения в *теле (органе)* оцениваются. Проверка повторяется, пока выражение не заканчивается нулем или пустым списком ( ).
write-line	Синтаксис: (write-file str-file-name str-buffer)	Пользуется преимуществом того факта, что результат последнего считывания сохраняется во внутреннем системном буфере. Использование write-line без параметров выводит содержимое буфера последнего считывания на экран.

Лабораторная работа № 1
Разработка программ с применением списков
Использование функций if, for, while

Цель работы: Ознакомиться со структурой программ, описываемых на языке NewLISP. Изучить особенности языка NewLISP и основные понятия данного языка. Освоить функции для работы со списками, приобретение навыков написания функций для работы со списками.

Материальное обеспечение: Компьютерное оборудование и программное обеспечение.

Краткие теоретические сведения:

Списки являются основными типами данных языка Лисп. В Лиспе список - это последовательность элементов (list). Под списком понимается конечная последовательность элементов списка, заключенная в круглые скобки. Элементом списка может быть либо атом, либо список. Первый элемент списка называется головой списка, а остаток списка (без первого элемента) - хвостом списка. Атомом называется произвольная последовательность букв и цифр, заключенная между двумя ограничителями языка Лисп. Литеральный атом часто называется символом. В языке Лисп определены два стандартных атома T и NIL, выполняющие роль понятий ИСТИНА и ЛОЖЬ. Строго говоря, числа в Лиспе также являются атомами. Пример списка: (A (B (C))). Данный список состоит из двух элементов: атома A и списка (B (C)), который в свою очередь состоит из атома A и списка (C). Пустой список - это список, не содержащий ни одного элемента. Он обозначается ( ) или атомом NIL. Пустой список в Лиспе относится к атомам.

Таким образом, список - это многоуровневая или иерархическая структура данных, в которой открывающиеся и закрывающиеся скобки находятся в строгом соответствии. Список, в котором нет ни одного элемента, называется пустым списком и обозначается символом NIL.

Вследствие способа, которым элементы списка связаны друг с другом в памяти, для них существуют возможности быть либо атомами, либо подсписками.

Подсписок — это просто список, который содержится в другом списке.

Апостроф — специальный символ, предотвращающий вычисление выражения, в результате чего получается обычный список. Апостроф — это просто короткая запись для функции quote. Следующие записи эквивалентны (переменной b присваивается список, а не квадратный корень из 25):

```
(setq b (quote (sqrt 25)))
(setq b '(sqrt 25))
```

Символ в функциональном программировании аналогичен переменной в традиционном языке программирования – это имя, состоящее из букв латиницы, цифр и некоторых специальных литер. Символ, как и переменная, может иметь какое-либо значение, то есть представлять какой-либо объект. Наряду с

символами, в ФП используются также: числа (целые и вещественные); специальные константы t и nil, обозначающие логические значения true (истина) и false (ложь); списки. Символы, числа и специальные константы представляют простейшие объекты, на основе которых строятся все прочие объекты данных. Поэтому для них используется обобщающее название – **атомы**.

Атомы - это простейшие объекты Лиспа, из которых строятся остальные структуры. Атомы бывают двух типов - символьные и числовые.

Символьные атомы - последовательность букв и цифр, при этом должен быть, по крайней мере, один символ, отличающий его от числа.

Числовые атомы - обыкновенные числа.

Атомы являются элементарными S-выражениями (последовательность из букв и цифр, начинающаяся с буквы). Атомы могут иметь вид имен, чисел или других объектов, неделимых базовыми средствами языка. Атомы, выглядящие как имена, могут обладать свойствами, задаваемыми системой или программой.

Кроме списков имеются более общие структуры данных – **символьные выражения** (**S-выражения**), реализуемые как двоичные деревья, а ЛИСП-системы поддерживают обработку различных специальных структур данных, таких как вектора, массивы, строки, хэш-таблицы, файлы, потоки ввода-вывода и др. Все вышеперечисленные объекты (атомы и списки) в совокупности называют символьными выражениями. Областью определения и областью значений для функций в функциональном программировании являются S-выражения. Виды S-выражений показаны на рисунке 1.1.

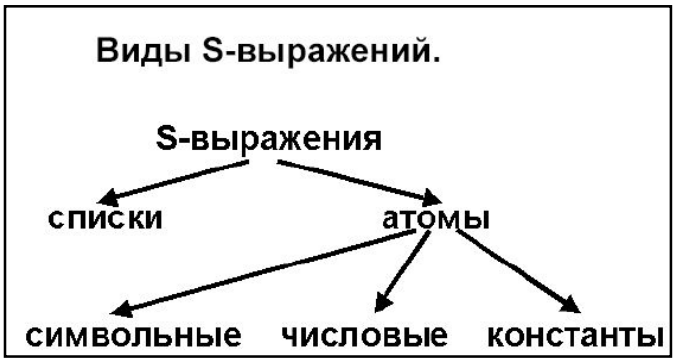

Рис 1.1 Виды S – выражений

В языках программирования, ориентированных на компиляцию, принято переменные классифицировать по типам данных, а значения в памяти хранить без информации о типе данных. Особым видом структур являются списки - последовательность термов, заключенная в скобки. В виду важности подобного вида структур, для их обработки был разработан специальный синтаксис.

ЛИСП является языком *функций.* Это означает, что каждая конструкция, которая может быть записана на ЛИСПе и которая является функцией в математическом смысле, выполняется в ЛИСПе как функция.

Функцией называется правило, по которому каждому значению одного или нескольких аргументов ставится в соответствие конкретное значение результата.

Используемые функции:

Функция set - вычисляет оба параметра, затем присваивает результат выражения *exp* символу *sym-variable*. Выражение возвращает результат присваивания. Старое содержание символа удаляется.
Синтаксис: (set *sym-variable exp*)

Функция print - выводит на текущее устройство ввода-вывода все результаты вычислений выражений *exp-1....* Чтобы завершать печать переводом строки, используется функция **println**.
Синтаксис: (print exp-1 [*exp-2* ...]), где exp-1 – выражение.

Функция println - выводит на текущее устройство ввода-вывода все результаты вычислений выражений *exp-1...* В конец печатается перевод строки. Функция println в остальном работает аналогично функции print.
Синтаксис: (println *exp-1* [*exp-2* ...])

Функция if - в зависимости от результата вычисления *exp-condition*, вычисляется либо выражение *exp-1*, либо выражение *exp-2*, если оно определено, и возвращается результат вычисления одного из этих выражений. Если выражение *exp-condition* не является пустым списком ( ) и не равно nil, то в этом случае вычисляется выражение *exp-1*, в противном же случае – выражение *exp-2*.
Синтаксис: (if *exp-condition exp-1* [*exp-2*])

Функция for - повторяет выражение или выражения в *body* в количестве диапазона значений, указанных в *num-from* и *num-to*. Может быть определен размер шага *num-step*. Если размер шага не указан, то принимается 1. Возвращает последний результат вычисления выражений в *body*.
Синтаксис: (for (sym *num-from num-to* [*num-step*]) *body*)

Функция while - если результат не является нулевым или пустой список (), выражения в *теле(органе)* оцениваются. Проверка повторяется, пока выражение не заканчивается нулем или пустым списком ( ).
Синтаксис: (while exp-condition body)

Функция read-line - считывает строку из текущего устройства ввода-вывода, строки разграничены символом перевода строки. Нет ограничения к длине строки. Символ перевода строки не является частью возвращаемой строки. По умолчанию текущее устройство – клавиатура. Также может быть определен дескриптор файла *int-file*, который получен при последней операции open.
Синтаксис: (read-line [*int-file*])

Функция write-line - пользуется преимуществом того факта, что результат последнего считывания сохраняется во внутреннем системном буфере. Использо-

вание write-line без параметров выводит содержимое буфера последнего считывания на экран.
Синтаксис: (write-file str-file-name str-buffer)

Функция length - определяет длину списка.
Синтаксис: syntax: (length *list*)

Функция open - предназначена для открытия конкретного файла.
Синтаксис: (open str-path-file str-access-mode)

Пример:
Вывести числа от 11 до 20 в файл afile.dat, затем вывести их на экран.

```
(set 'out-file (open "afile.dat" "write"))
(for (x 11 20 1) (write-line (string x) out-file))
(close out-file)
(set 'in-file (open "afile.dat" "read"))
(while (set 'x (read-line in-file))
(print x " "))
(println)
 (close in-file)
```

Результат работы программы:
11 12 13 14 15 16 17 18 19 20

Задания:
1. Определить значение заданного линейного списка по заданному индексу.
2. Определить индекс списка для заданного значения, если заданное значение находится вне диапазона, то вывести сообщение “Нет значения”.
3. Вывести числа от 11 до 20 в файл afile.dat, затем вывести их на экран.
4. Прочитать числа из файла afile.dat и записать их в список, найти среднее значение после записи.
5. Прочитать числа из файла afile.dat и записать их в список и найти максимум и минимум этих чисел
6. Решите квадратное уравнение по заданным коэффициентам.
7. Автоморфными называются числа, которые содержатся в последних разрядах их квадрата, например, 5*5=25, 25*25=625. Напишите программу для нахождения нескольких автоморфных чисел.

Порядок выполения работы:
1. Изучить теоретические сведения по данной лабораторной работе.
2. Составить программу для заданного варианта.
3. Записать программу на языке NewLisp.
4. Отладить программу.
5. Продемонстрировать работу программы на ПК для заданного варианта.

Содержание отчета:

1. Титульный лист.
2. Тему и цель работы.
3. Вариант задания.
4. Текст и результат программы.
5. Анализ результатов программы.
6. Ответы на контрольные вопросы.
7. Вывод.

Контрольные вопросы:

1. Что называется списком?
2. Предназначение и синтаксис написания функций print и println.
3. Предназначение и синтаксис написания функции if.
4. Предназначение и синтаксис написания функции for
5. Предназначение и синтаксис написания функции while.
6. Предназначение и синтаксис написания функций read-line, write-line.
7. Предназначение и синтаксис написания функции length.
8. Предназначение и синтаксис написания функции open.

Библиографический список:

1. Полещук Н., Лоскутов П. AutoLISP и Visual LISP в среде AutoCAD. М., 2006.
2. Хювёнен Э., Сеппянен И. Мир Лиспа. Методы и системы программирования. М, 1990.
3. Хювёнен Э., Сеппянен И. Мир Лиспа. Введение в язык Лисп и функциональное программирование. М., 1990.
4. Кичкайло Т.А., Тушев А.Н. Функциональное и логическое программирование: Учебное пособие – Алт.гос.тех.ун-т им.И.И.Ползунова. Центр дистанционного обучения. Барнаул, 1999. – 148с.
5. Маурер У. Введение в программирование на языке Лисп. - М: Мир, 1976.
6. Хоггер К. Введение в логическое программирование. – М.: Мир, 1988. – 348 с.
7. Лавров С., Силагадзе Г. Автоматическая обработка данных. Язык Лисп и его реализация. - М.: Наука, 1978.

Лабораторная работа № 2
Разработка программ с применением функций push, pop, sort

Цель работы: Ознакомиться с функциями push, pop, sort. Изучить особенности написания синтаксиса данных функций и области их применения в языке NewLISP, приобрести навыки работы используемых функций.

Материальное обеспечение: Компьютерное оборудование и программное обеспечение.

Краткие теоретические сведения:

Функции в языке программирования NewLISP составляют функциональный базис данного языка. В первую очередь это функции работы со списками. Логически функциональный базис
можно разделить на следующие категории:

- ✓ работа со списками;
- ✓ предикаты;
- ✓ ветвление;
- ✓ определение функции и лямбда - выражения;
- ✓ преобразование данных и программ.

Диаграммы s-выражений

Базовый элемент этих диаграмм называется списочной ячейкой (cons-cell). Списочная ячейка состоит из 2-х частей: полей CAR и CDR, т.е. частей, содержащих голову и хвост списка.

Функция car возвращает голову переданного в качестве аргумента списка (первый элемент списка называется головой, а остаток списка – хвостом).

Функция cdr возвращает хвостовую часть списка.

Каждое из полей содержит указатель. Указатель может ссылаться на другую списочную ячейку или некоторый другой Лисп-объект, например, атом.

Для представления списка используют одну списочную ячейку для каждого элемента списка. Первая часть ячейки заполняется именем атома, если элемент является атомом, или указателем на другой список, если элемент является списком. Вторая часть списочной ячейки содержит указатель на следующий элемент списка или знак \ (слэш), если этот элемент является последним элементом списка. На рисунке 2.1 приведено несколько примеров представления конструкций языка в виде диаграмм списочных ячеек:

(+ 3 5)

(* (a b (c)))

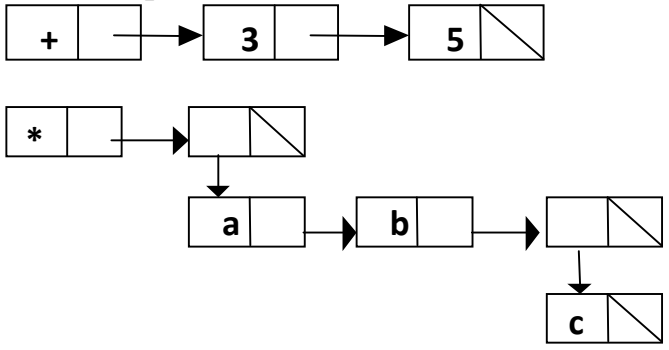

Рис. 2.1. Изображение списков в виде диаграмм списочных ячеек

Рекурсия и итерация

Функция является рекурсивной, если в ее определении содержится вызов этой функции. На рекурсии основана декомпозиция задачи на подзадачи, решение которых, насколько это, возможно, пытаются свести к уже решенным или к решаемой в настоящий момент задаче. Рекурсия близка к аппарату математической индукции, что определяет удобство реализации индуктивных определений. Для рекурсивных функций характерно наличие терминальной ветви. Как правило, рекурсивная функция реализует разбор случаев, некоторые из них влекут продолжение процесса вычисления через вызов рекурсивной функции, другие не прибегают к рекурсии, они называются терминальными ветвями.

Для решения многих видов вычислительных задач требуется неоднократно повторять одни и те же алгоритмические процессы. Такие процессы могут быть описаны с помощью рекурсивных или итеративных процедур. Рекурсивная процедура или функция – это подпрограмма, которая вызывает сама себя, т.е. выполнению рекурсивной подпрограммы предшествует выполнение ее собственной копии. При выполнении рекурсивной подпрограммы осуществляется многократный переход от некоторого текущего уровня организации алгоритма к нижнему уровню последовательно до тех пор, пока, не будет получено тривиальное решение поставленной задачи. Итерация обычно бывает более эффективной, чем рекурсия, поскольку для завершения шага итерации – в отличие от шага рекурсии – не требуется ожидать результатов выполнения последующих шагов. Использование итерации, следовательно, позволяет избежать расходов, связанных с организацией в период исполнения стека латентных вызовов (вызовов еще ожидающих активации), что невозможно при использовании рекурсии. Рекурсивная форма организации алгоритма обычно выглядит изящнее итерационной и дает более компактный текст программы, но при выполнении, как правило, медленнее и может вызвать переполнение стека латентных вызовов.

Рекурсия также используется для описания структур данных частично состоящих или определяемых с помощью самих себя. Примерами таких рекурсивных структур данных являются списки и деревья. Достоинство рекурсивного определения объекта состоит в том, что оно позволяет с помощью конечного высказывания определить бесконечное множество объектов.

Для рассмотрения примеров написания программ на языке Лисп, содержащих рекурсивные или итерационные функции, нам необходимо познакомиться с некоторыми функциями языка Лисп, которые могут понадобиться для написания этих программ.

Определение функций и их вычисление в Лиспе основано на лямбда-исчислении Черча. В лямбда-исчислении функция записывается в следующем виде:

lambda (x1, x2, .., xn).fn

В Лиспе лямбда-выражение имеет вид:
(LAMBDA (x1 x2 ... xn) fn)
где xi – формальные параметры,
fn – тело функции

Лямбда-выражение – это безымянная функция, которая пропадает тотчас после вычисления значения формы. Дать имя и определить новую функцию можно с помощью функции defun:

(defun имя_функции (список_аргументов) тело_функции)

Значением этой формы является имя новой функции.

Например, определим функцию, реализующую сложение двух чисел:

>(defun add (a b) (+ a b))
ADD

Заметьте, что интерпретатор вернул имя определенной пользователем функции в качестве значения s-выражения. Теперь мы можем использовать функцию add подобно любой встроенной лисповской функции.

Используемые функции:

Функция pop - удаляет первый элемент из списка *list* и возвращает удаляемый элемент. Если указан второй параметр, то удаляется элемент, имеющий положение *int-offset* в списке. Если указанный индекс больше по величине, чем длина списка или равен длине списка, то удаляется последний элемент.
Синтаксис: (pop *list* [*int-offset*])

Функция push - элемент *exp* вставляется в список *list*. Если определен параметр *int-offset*, то элемент вставляется в указанной позиции в списке, если параметр не определен, то элемент вставляется в начало списка. После вставки, возвращается вставленный элемент. Если значение *int-offset* больше или равно длине списка или отрицательное число, то элемент вставляется в конец списка.
Синтаксис: (push *exp list* [*int-offset*])

Функция sort - это функция, которая производит сортировку элементов.
Синтаксис: (sort list)

Пример:

Используя функции for и push записать (с выводом на экран) числа от 1 до 10 в пустой поначалу список, и вывести их на экран в обратном порядке, используя функции while и pop.

```
(set 'pList '())
(for (i 1 10 1)
(print i " ")
(push i pList))
(println)
(while (!= pList '())
(print (pop pList) " "))
(println)
```

Результат работы программы:
1 2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1

Задания:
1. Используя функции for и push записать (с выводом на экран) числа от 1 до 10 в пустой поначалу список, и вывести их на экран в обратном порядке, используя функции while и pop.
2. Предварительно записать 10 чисел в файл afile.dat произвольном порядке, прочитать их из файла и втолкнуть в список, выталкивать из списка и выводить их в возрастающем порядке.
3. Вывести на экран, сколько раз во введенном числе повторяется каждый из его простых делителей.
4. Совершенным числом называется целое число, которое равно сумме всех его сомножителей, за исключением самого этого числа. Напишите программу получения трех совершенных чисел. Например 28=1+2+4+7+14.
5. Напишите программу, которая генерирует всех k-элементных подмножеств множества {1,...,n} в лексикографическом порядке.
6. Составьте программу получения простых чисел от 1 до 100.

Порядок выполения работы:
1. Изучить теоретические сведения по данной лабораторной работе.
2. Составить программу для заданного варианта.
3. Записать программу на языке NewLisp.
4. Отладить программу.
5. Продемонстрировать работу программы на ПК для заданного варианта.

Содержание отчета:
1. Титульный лист.
2. Тему и цель работы.
3. Вариант задания.
4. Текст и результат программы.
5. Анализ результатов программы.
6. Ответы на контрольные вопросы.
7. Вывод.

Контрольные вопросы:
1. Что называется функцией?
2. Предназначение и синтаксис написания функции pop.
3. Предназначение и синтаксис написания функции push.
4. Предназначение и синтаксис написания функции sort.

Библиографический список:
1. Полещук Н., Лоскутов П. AutoLISP и Visual LISP в среде AutoCAD. М., 2006.
2. Хювёнен Э., Сеппянен И. Мир Лиспа. Методы и системы программирования. М., 1990.
3. Хювёнен Э., Сеппянен И. Мир Лиспа. Введение в язык Лисп и функциональное программирование. М., 1990.
4. Кичкайло Т.А., Тушев А.Н. Функциональное и логическое программирование: Учебное пособие – Алт.гос.тех.ун-т им. И.И.Ползунова. Центр дистанционного обучения. Барнаул, 1999. – 148с.
5. Маурер У.. Введение в программирование на языке Лисп. - М: Мир, 1976.

Лабораторная работа № 3
Разработка программ с применением функций nth, first, last, rest

Цель работы: Ознакомиться с функциями nth, first, last, rest. Изучить особенности написания синтаксиса данных функций и области их применения в языке NewLISP, приобрести навыки работы используемых функций.

Материальное обеспечение: Компьютерное оборудование и программное обеспечение.

Краткие теоретические сведения:

Более общая функция NTH принимает два аргумента: индекс и список, и возвращает n-ый (начиная с нуля) элемент списка. Также существует функция NTHCDR, принимающая индекс и список и возвращающая результат n-кратного применения REST к списку.

Таким образом, (nthcdr 0 ...) просто возвращает исходный список, а (nthcdr 1 ...) эквивалентно вызову REST. Имейте в виду, что ни одна из этих функций не является более производительной по сравнению с эквивалентной комбинацией FIRST и REST, т.к. нет иного способа получить n-ый элемент списка без n-кратного вызова CDR. Функция NTH - выделение (N + 1)-го элемента списка (счет элементов в списке начинается с N = 0):

Например:

```
>(NTH 3 '(1 1 2 3 5 8))
    3
```

Элементы нумеруются начиная с нуля. Ее можно определить так:

```
(DEFUN NTH (I L)
   (COND ((ATOM L) NIL)
   ((ZEROP I) (CAR L))
   (T (NTH (- I 1) (CDR L)))))
```

Примеры:

```
(NTH 1 '(1 2 3)) ==> 2
(NTH 3 '(A B C (D E))) ==> (D E)
```

(NTH 0 '(A B C)) ==> A

Функция LAST:

(LAST x). Обычная функция. Ее значением является последнее звено списка **x**, т.е. список из последнего элемента списка x. Если список x пустой, то значением функции будет **NIL**. Ее можно опеределить так:

```
(DEFUN LAST (L)
   (COND  ((ATOM L) L)
   ((ATOM (CDR L)) L)
   (T (LAST (CDR L)))))
```

Примеры:

(LAST '(A1 B2 C3)) ==> (C3)
(LAST '(A1 B2 . C3)) ==> (B2 . C3)

Функция AND:

(AND e1 e2 ... en). Это — особая функция. Значением является конъюнкция аргументов. Эта функция последовательно вычисляет аргументы (слева направо) до тех пор, пока не появится значение, равное **NIL**; в этом случае функция прекращает вычисление аргументов и заканчивает свое выполнение со значением **NIL**. Если же значения всех аргументов отличны от **NIL**, то значением функции будет значение последнего аргумента. Таким образом, функция **AND** может использоваться как следующее условное выражение: "если e1, и e2,..., и en-1, то выполнить e ".

Функция OR:

(OR e1 e2 ... en). Это — особая функция. Значением ее является дизъюнкция аргументов.

Эта функция последовательно вычисляет аргументы до тех пор, пока не появится значение, отличное от **NIL**; в этом случае функция прекращает вычисление аргументов и выдает последнее вычисленное значение. Если же значения всех аргументов равны **NIL**, то и значением функции будет **NIL**. Таким образом, функция может использоваться для задания такого условного выражения: "если не e1, не e2, ..., не en-1 то выполнить en"

Примеры:

(OR 'A 'B 'C) ==> A
(OR 'NIL 'B 'C) ==> B
(OR (ATOM '(A)) (NUMBERP 'A)) ==> NIL

Рассмотрим теперь функцию **COND**, которая служит для задания условных выражений произвольного вида.

Примеры:

(AND 'A () (5 B C)) ==> NIL
(AND (ATOM 5) (NUMBERP 5)) ==> T
(AND (+ 5 6) 3 7) ==> T

Используемые функции:
Функция nth - находит элемент в списке *list*, имеющий индекс *num-offset*. Первый элемент будет иметь индекс 0, второй – индекс 1 и т.д. Последний элемент списка возвращается в том случае, если индекс находится вне диапазона.
Синтаксис: (nth *num-offset list*)

Функция first - возвращение первого элемента списка.
Синтаксис: (first list).

Функция last - возвращение последнего элемента списка.
Синтаксис: (last list).

Функция rest – оценивает значение списка, затем возвращает копию всего списка за исключением первого элемента.
Синтаксис: (rest list).

Функция and - вычисляет выражения слева направо, пока не происходит результат nil, при этом возвращает nil. Если ни одно из выражений не дает результат nil, то возвращается результат последнего выражения.
Синтаксис: (and *exp-1 exp-2* [*exp-3*...])
Пример:
Запишите числа от 1 до 10 в список, прочитайте их в прямом и обратном порядке без разрушения списка с использованием функции nth.

```
(set 'pLisp '())
(for (x 1 10 1)
  (push  x pLisp)

 (print x " "))
 (println)
(for (x 1 10 1)
  (set 'i (nth (- x 1) pLisp))
  (print i " "))
  (println)
(for (x 1 10 1)
  (set 'i (nth (- 10 x) pLisp))
  (print i " "))
  (println)
```

Результат программы:
1 2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1
1 2 3 4 5 6 7 8 9 10

Задания:

1. Запишите числа от 1 до 10 в список, прочитайте их в прямом и обратном порядке без разрушения списка с использованием функции nth.
2. Запишите числа от 1 до 10 в список, прочитайте их в прямом порядке с использованием функций first и rest.
3. Напечатать все трехзначные числа, в десятичной записи которых нет двух одинаковых цифр.
4. Даны натуральные числа m и n. Найти такие натуральные числа m1 и n1, не имеющие общих делителей, что m1/n1=m/n. (Сокращение дроби.)
5. Составить программу вывода всех трехзначных десятичных чисел, сумма цифр которых равна данному натуральному числу.
6. Число из n цифр является числом Армстронга, если сумма цифр, возведенных в n-ю степень, равна самому числу. Напишите программу, которая находит числа Армстронга из трех и четырех цифр.

Порядок выполения работы:

1. Изучить теоретические сведения по данной лабораторной работе.
2. Составить программу для заданного варианта.
3. Записать программу на языке NewLisp.
4. Отладить программу.
5. Продемонстрировать работу программы на ПК для заданного варианта.

Содержание отчета:

1. Титульный лист.
2. Тему и цель работы.
3. Вариант задания.
4. Текст и результат программы.
5. Анализ результатов программы.
6. Ответы на контрольные вопросы.
7. Вывод.

Контрольные вопросы:

1. Предназначение и синтаксис написания функции nth.
2. Предназначение и синтаксис написания функции first.
3. Предназначение и синтаксис написания функции last.
4. Предназначение и синтаксис написания функции rest.
5. Предназначение и синтаксис написания функции and.

Библиографический список:

1. Полещук Н., Лоскутов П. AutoLISP и Visual LISP в среде AutoCAD. М., 2006.
2. Хювёнен Э., Сеппянен И. Мир Лиспа. Методы и системы программирования. М., 1990.
3. Хювёнен Э., Сеппянен И. Мир Лиспа. Введение в язык Лисп и функциональное программирование. М., 1990.

4. Кичкайло Т.А., Тушев А.Н. Функциональное и логическое программирование: Учебное пособие – Алт.гос.тех.ун-т им. И.И.Ползунова. Центр дистанционного обучения. Барнаул, 1999. – 148 с.
5. Маурер У. Введение в программирование на языке Лисп. - М: Мир, 1976.

Лабораторная работа № 4
Разработка программ с применением функций find, substring, replace-nth

Цель работы: Ознакомиться с функциями find, substring, replace-nth. Изучить особенности написания синтаксиса данных функций и области их применения в языке NewLISP, приобрести навыки работы используемых функций.

Материальное обеспечение: Компьютерное оборудование и программное обеспечение.

Краткие теоретические сведения:

Лямбда – исчесления как основа определения функций

Определение функций и их вычисление в Лиспе основано на лямбда-исчислении. В предложенном исчислении функция записывается в следующем виде:
lambda(x1, x2, …,xn).fn. В Лиспе исчисление заимствовано для определения вычислений и описания параметров функций:
(lambda (x1 x2 …xn) fn)

Символ lambda означает, что мы имеем дело с определением производимых функцией действий. Символы xi являются формальными параметрами определения, которые именуют аргументы в описывающем вычисления теле функции fn. Телом функции является произвольная форма, значение которой может вычислить интерпретатор Лиспа. Формальность параметров означает, что их можно заменить на любые другие символы, и это не отразится на определяемых функцией действиях.

Неименованные функции Лиспа.

Лямбда-выражение – это определение вычислений и параметров функций в чистом виде без фактических параметров :
(lambda (<список формальных параметров>) < тело функции>).

Для применения описанной таким образом функции к некоторым аргументам a1, …, an необходимо в
вызове функции поставить лямбда-выражение на место имени функции :
(< лямбда-выражение > a1 a2 … an).

Такую форму вызова называют лямбда-вызовом. Если вычислять значения аргументов не нужно, то такую функцию нужно определять с помощью nlambda – выражения (от англ. No-spread lambda) :
(nlambda (<список формальных параметров>) <тело функции>)nlambda – функции применяются при разработке новых синтаксических форм для

расширения языка, а также при реализации интерпретаторов проблемно-ориентированных языков с лиспоподобной структурой.

Используемые функции:
Функция find - ищет искомый элемент в списке.
Синтаксис: (find exp-key list)
Пример:
(find "world" '("hello" "world")) => 1
(find "hi" '("hello" "world")) => nil
(find '(1 2) '((3 4) 5 6 (1 2) (8 9))) => 3

(find "world" "Hello world") => 6
(find "WORLD" "Hello woRLd") => nil
(find "WorlD" "Hello woRLd" nil) => 6
(find "hi" "hello world") => nil
(find "Hello" "Hello world") => 0

Функция substring - определяет подстроку заданной строки.
Синтаксис: (substring str int-offset [int-length])
Примеры:
(substring "Hello World" 6 2) = "Wo"
(substring "Hello World" 0 5) = "Hello"
(substring "Hello World" 6) = "World"

Функция replace-nth - заменяет nth в int-nth-ом элементе в списке с оценкой exp-replacement и возвращает элемент, который был заменен.
Синтаксис: (replace-nth int-nth list exp-replacement)

Пример:
Вывести на экран, сколько раз во введенном слове повторяется каждая буква.

```
(println (set 's "nobody"))
(println (set 'l (length s)))
(set 'b '())
(set 'd '())
(for (x 0 (- l 1) 1)
  (if (= (set 'i (find (substring s x 1) b)) nil)
    (begin
      (push (substring s x 1) b)
      (push 1 d)
    )
    (replace-nth i d (+ (nth i d) 1))
  )
)
(while (!= b '())
```

```
  (print (pop b) " ")
  (print (pop d)" "))
(println)
```

Результат программы:

```
nobody
6
y 1 d 1 b 1 o 2 n 1
```

Задания:

1. Вывести на экран, сколько раз во введенном слове повторяется каждая буква.
2. Определить, является ли заданное слово палиндромом.
3. Составьте программу получения простых чисел от 1 до 100, используя метод Эратосфена.
4. Числа Пифагора определяются соотношением c*c=a*a+b*b, где a, b и c целые числа. Напишите программу для нахождения нескольких таких чисел.
5. Напишите программу, которая находит целые числа, при возведении в квадрат которых получаются палиндромы. Палиндром - это сочетания символов, которые читаются одинаково слева и справа.
6. Простое число Мерсенна - это число, которое может быть представлено 2p-1, где p - то же простое число. Напишите программу для нахождения ряда таких чисел.

Порядок выполения работы:

1. Изучить теоретические сведения по данной лабораторной работе.
2. Составить программу для заданного варианта.
3. Записать программу на языке NewLisp.
4. Отладить программу.
5. Продемонстрировать работу программы на ПК для заданного варианта.

Содержание отчета:

1. Титульный лист.
2. Тему и цель работы.
3. Вариант задания.
4. Текст и результат программы.
5. Анализ результатов программы.
6. Ответы на контрольные вопросы.
7. Вывод.

Контрольные вопросы:

1. Предназначение и синтаксис написания функции find.
2. Предназначение и синтаксис написания функции substring.
3. Предназначение и синтаксис написания функции replace-nth.

Библиографический список:
1. Полещук Н., Лоскутов П. AutoLISP и Visual LISP в среде AutoCAD. М., 2006.
2. Хювёнен Э., Сеппянен И. Мир Лиспа. Методы и системы программирования. М., 1990.
3. Хювёнен Э., Сеппянен И. Мир Лиспа. Введение в язык Лисп и функциональное программирование. М., 1990.

Лабораторная работа № 5
Разработка программ с применением функций parse, reverse, join

Цель работы: Ознакомиться с функциями parse, reverse, join. Изучить особенности написания синтаксиса данных функций и области их применения в языке NewLISP, приобрести навыки работы используемых функций.

Материальное обеспечение: Компьютерное оборудование и программное обеспечение.

Краткие теоретические сведения:

Используемые функции:

Функция parse - функция PARSE принимает единственный аргумент типа STRING, содержащий алгебраическое выражение в естественной нотации, а возвращает это же выражение, разбитое на
лексемы. Круглые скобки становятся структурными скобками результирующего S-выражения. Лексемами являются символические имена переменных, числовые константы и знаки алгебраических операций.

Пример:

```
(parse "hello how are you") => ("hello" "how" "are" "you")
(parse "one:two:three" ":") => ("one" "two" "three")
(parse "one--two--three" "--" => ("one" "two" "three")
```

Функция reverse - изменяет порядок элементов на обратный.

Синтаксис: (reverse list)

Пример:

```
(set 'l '(1 2 3 4 5 6 7 8 9))
(reverse l) => (9 8 7 6 5 4 3 2 1)
l => (9 8 7 6 5 4 3 2 1)
```

Функция join - соединяет строки в списке.

Синтаксис: (join list [string]).

Пример:

Напишите программу, которая читает текст и печатает частоту появления в нем различных слов.

```
(println (set 's "I received a letter from you, the letter was fine"))
```

```
(set 'lst (parse s))
(set 'l (length lst))
(set 'b '())
(set 'd '())
(for (x 0 (- l 1) 1)
  (if (= (set 'i (find (nth x lst) b)) nil)
    (begin
      (push (nth x lst) b)
      (push 1 d)
    )
    (replace-nth i d (+ (nth i d) 1))
  )
)
(while (!= b '())
  (print (pop b) " ")
  (println (pop d))
)
```

Результат программы:

```
I received a letter from you, the letter was fine
fine 1
was 1
the 1
, 1
you 1
from 1
letter 2
a 1
received 1
I 1
```

Задания:

1. Напишите программу, которая читает текст и печатает частоту появления в нем различных слов.
2. Используя функции for и push записать (с выводом на экран) числа от 1 до 10 в пустой поначалу список, и вывести их на экран в прямом порядке, используя функции while и pop.
3. Запишите числа от 1 до 20 в 2 списка, четные в один список, нечетные - в другой, затем выдайте содержимое списков в возрастающем порядке с использованием функции pop.
4. Перевернуть заданное слово (т.е. получить анаграмму) и выдать на экран.
5. Можно ли заданное натуральное число представить в виде суммы двух квадратов натуральных чисел? Написать программу решения этой задачи.

Порядок выполения работы:

1. Изучить теоретические сведения по данной лабораторной работе.
2. Составить программу для заданного варианта.
3. Записать программу на языке NewLisp.
4. Отладить программу.
5. Продемонстрировать работу программы на ПК для заданного варианта.

Содержание отчета:

1. Титульный лист.
2. Тема и цель работы.
3. Вариант задания.
4. Текст и результат программы.
5. Анализ результатов программы.
6. Ответы на контрольные вопросы.
7. Вывод.

Контрольные вопросы:

1. Предназначение и синтаксис написания функции parse.
2. Предназначение и синтаксис написания функции reverse.
3. Предназначение и синтаксис написания функции join

Библиографический список:

1. Полещук Н., Лоскутов П. AutoLISP и Visual LISP в среде AutoCAD. М., 2006.
2. Хювёнен Э., Сеппянен И. Мир Лиспа. Методы и системы программирования. М., 1990.
3. Хювёнен Э., Сеппянен И. Мир Лиспа. Введение в язык Лисп и функциональное программирование. М., 1990.
4. Кичкайло Т.А., Тушев А.Н. Функциональное и логическое программирование: Учебное пособие – Алт.гос.тех.ун-т им. И.И.Ползунова. Центр дистанционного обучения. Барнаул, 1999. – 148 с.
5. Маурер У. Введение в программирование на языке Лисп. – М.: Мир, 1976.

Лабораторная работа № 6
Разработка программ с применением функций print, define

Цель работы: Ознакомиться с функциями print, define. Изучить особенности написания синтаксиса данных функций и области их применения в языке NewLISP, приобрести навыки работы используемых функций.

Материальное обеспечение: Компьютерное оборудование и программное обеспечение.

Краткие теоретические сведения:

Используемые функции:

Функция print - выводит на текущее устройство ввода-вывода все результаты вычислений выражений *exp-1*.... Чтобы завершать печать переводом строки, используется функция **println**.

Синтаксис: (print exp-1 [*exp-2* ...]), где exp-1 – выражение.

Пример:
```
(print (set 'res (+ 1 2 3)))
 (print "the result is" res "\n")
```

Функция define - определяет новую функцию *sym-name* с дополнительными параметрами *sym-param-1*.... Функция define эквивалентна связыванию выражения лямбды с символом. При запросе определенной функции, все параметры вычисляются и связываются с переменными *sym-param-1*..., затем вычисляется тело *body-1*.... При определении функции возвращается лямбда выражения, содержащегося в *sym-name*. Все определенные параметры являются дополнительными. При запросе определенной функции без указания значений параметров, эти параметры примут значения nil.

Синтаксис: (define *sym-name exp*)

Пример:

Написать программу вычисления факториала.

```
(define (fact i)
  (set 'f 1)
  (if (> i 1)
     (while (> i 1)
      (set 'f (* i f))
      (set 'i (- i 1))
     )
   )
 )
  (fact 5)
 (println f)
```

Результат программы: 120

Задания:

1. Написать программу вычисления факториала.
2. Напишите программу, которая переводит заданное десятичное число в двоичную, восьмеричную и шестнадцатеричную системы исчисления.

3. Сколькими способами заданное натуральное число N можно представить в виде суммы двух кубов натуральных чисел N=i*i*i+j*j*j? Перестановка слагаемых нового способа не дает. Операцией возведения в степень пользоваться нельзя.
4. Напишите программу, которая считала бы два целых числа и определяла бы, являются ли они взаимно-простыми. Два числа называют взаимно-простыми, если они не имеют общих делителей.
5. Напечатать все представления натурального числа N суммой натуральных чисел. Перестановка слагаемых нового способа не дает. Например, при N=4 получается:

4; 3+1; 2+1+1; 2+2; 1+1+1+1.

6. Напечатать все взаимно-простые по отношению к заданному числу и не превышающее это число числа в порядке возрастания.

Порядок выполения работы:

1. Изучить теоретические сведения по данной лабораторной работе.
2. Составить программу для заданного варианта.
3. Записать программу на языке NewLisp.
4. Отладить программу.
5. Продемонстрировать работу программы на ПК для заданного варианта.

Содержание отчета:

1. Титульный лист.
2. Тема и цель работы.
3. Вариант задания.
4. Текст и результат программы.
5. Анализ результатов программы.
6. Ответы на контрольные вопросы.
7. Вывод.

Контрольные вопросы:

1. Предназначение и синтаксис написания функции print.
2. Предназначение и синтаксис написания функции define.

Библиографический список:

1. Полещук Н., Лоскутов П. AutoLISP и Visual LISP в среде AutoCAD. М., 2006.
2. Хювёнен Э., Сеппянен И. Мир Лиспа. Методы и системы программирования. М., 1990.
3. Хювёнен Э., Сеппянен И. Мир Лиспа. Введение в язык Лисп и функциональное программирование. М., 1990.
4. Кичкайло Т.А., Тушев А.Н. Функциональное и логическое программирование: Учебное пособие – Алт.гос.тех.ун-т им. И.И.Ползунова. Центр дистанционного обучения. Барнаул, 1999. – 148 с.
5. Маурер У. Введение в программирование на языке Лисп. - М: Мир, 1976.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к выполнению лабораторных работ по языку NewLISP
по дисциплине “Функциональное и логическое программирование ”

Составители: ***Ешпулатов С.Е., Клемешева О.В.***

Тех. редактор ***Бейшеналиева А.И.***

Подписано к печати 30.05.2011 г. Формат бумаги 60x84 1/16.
Бумага офс. Печать офс. Объем 2 п.л. Тираж 50 экз. Заказ 231. Цена 36 с.

Бишкек, ул. Сухомлинова, 20. ИЦ “Текник” КГТУ им. И. Раззакова, т.: 54-29-43
e-mail: beknur@mail.ru