Національний технічний університет «Харківський політехнічний інститут»
Машинобудівний факультет
Кафедра «Інтегровані технології машинобудування» ім. М.Ф. Семка

Кобець О.В.

ЗАВДАННЯ ДЛЯ ЛАБОРАТОРНИХ РОБІТ

з дисципліни «…Основи програмування інженерних систем графіки.»

Харків

ПРОГРАММИРОВАНИЕ на AutoLISP

Цель этих методических указаний - познакомить студентов с языком программирования AutoLISP, на котором были написаны многие встроенные функции AutoCAD. Задача - обучить программированию на новом языке в сжатые сроки. В методических указаниях присутствует описание нескольких тренировочных заданий, которые должны помочь приобрести опыт программирования.

1. Назначение и возможности языка AutoLISP

AutoLISP — это язык программирования, способный существенно повысить производительность проектирования объектов за счет автоматизации часто используемых или повторяющихся задач. AutoLISP позволяет взглянуть на работу AutoCAD изнутри и может служить отличным средством для изучения самых современных методов автоматизации. Программы AutoLISP обеспечивают полный контроль над работой и взаимодействием с пользователем.

Кажущаяся сложность AutoLISP необоснованно отпугивает многих пользователей AutoCAD. Однако время, затраченное на изучение этого языка, окупается теми преимуществами, которые дает его использование.

В составе системы AutoCAD поставляется интерпретатор языка AutoLISP. Если при генерации AutoCADa интерпретатор AutoLISPa был подключен, то он загружается в оперативную память после запуска графического редактора ACAD и доступен в течение всего сеанса работы с ACAD. Таким образом, графический редактор ACAD и интерпретатор языка AutoLISP представляют собой единую систему: любая функция AutoLISPa может быть вызвана из графического редактора, и любая команда редактора может быть использована в программе на AutoLISPe.

Наиболее характерны следующие классы применений: программирование чертежей с параметризацией, создание и использование графических баз данных, анализ и (или) автоматическое преобразование изображений.

- ***Программирование чертежей с параметризацией.*** Создаётся программа, позволяющая при каждом обращении к ней формировать новый чертёж, отличающийся от предыдущих чертежей, построенных этой же программой, размерами, а также, возможно, и топологией: могут появиться новые элементы, сечения, измениться текстовая часть чертежа и т.д. Время получения чертежа с помощью такой программы может быть в десятки раз меньше времени, необходимого для его создания с помощью редактора ACAD, и, что не менее важно, получить чертёж сможет любой конструктор, даже мало знакомый с командами ACAD.
- ***Создание и использование графических баз данных.*** Если накоплено большое количество чертёжных файлов, программ на AutoLISPe, соответствующих чертёжным фрагментам, деталям, узлам, то их можно в некотором смысле считать графической базой данных. Программы на AutoLISPe в сочетании с пользовательскими меню могут организовывать просмотр, поиск, подключение к объектам их частей и т.п. Хранение графических данных в виде набора программ на AutoLISPe даёт возможность в десятки и сотни раз сократить требуемую память по сравнению с памятью, необходимой для хранения чертёжных файлов ACAD, так как, во-первых, одна программа позволяет получить не один, а множество чертежей, во-вторых, текст программы на AutoLISPe занимает на порядок меньше памяти, чем файл, который может быть получен в результате работы этой программы.
- ***Анализ и (или) автоматическое преобразование изображений.*** Программа на AutoLISPe может воспринимать чертёж на экране, построенный с помощью графического редактора и обсчитывать его. Про-

грамма также может быстро осуществить преобразование изображения, на которое при работе в графическом редакторе пришлось бы затратить значительное время. Например: заменить все вставки одного типа на вставки другого типа из какого-либо чертёжного файла; перенести все объекты с одного слоя на другой; повернуть все блоки на заданный угол - каждый относительно своей базовой точки и т.п.

AutoLISP обеспечивает возможность формирования чертежа за считанные минуты. С его помощью можно расширить системы команд графического редактора ACAD и строить на основе универсального редактора специализированные САПР, имеющие гораздо более простой и естественный для пользователей язык, ориентированный на конкретную предметную область.

Название LISP образовано от **LISt Processing**, что в переводе означает "**обработка списков**", а понятие списков является базовым для программирования на этом языке.

Как правило, программа AutoLISP определяет имя команды, которую затем можно ввести в командой строке, чтобы выполнить программу. Некоторые программы содержат диалоговые окна, облегчающие выбор опций и параметров.

2. Основные этапы программирования на AutoLISP

Написание программы на AutoLISP требует выполнения ряда основных действий: четкой формулировки (постановки) задачи; выявление основных особенностей; взаимосвязей и количественных закономерностей; формирование графического изображения объекта; разработка программы; реализация программы.

Постановка задачи. На этом этапе определятся: графический объект, который может формироваться программой при различной входной информации, входные данные, выходные данные (указания размеров, текстовые пояснения, местоположение объекта на чертеже и т.д.).

Выявление основных особенностей, взаимосвязей и количественных закономерностей. На этом этапе устанавливаются взаимосвязи объектов чертежа с точки зрения эффективного их построения, определяются постоянные и переменные параметры, формируются запросы и сообщения, программно выдаваемые конструктору. Определяются способы построения отдельных элементов графического объекта и условия, которые влияют на выбор способа построения элемента.

Разработка последовательности действий создания графического изображения объекта. На этом этапе подробно излагаются все действия, связанные с построением графического изображения объекта, возможность использования циклических процедур.

Разработка программы (функций) AutoLISP. Основной этап, который включает в себя непосредственно написание программы и ее отладку в интегрированной среде разработки Visual LISP

Выполнение программы на AutoLISP. Заключительный этап реализации разработанной программы на ЭВМ.

Лабораторна робота №1

3. Интерпретатор AutoLISP

В AutoCAD входит интерпретатор AutoLISP, начинающий работать, если вы в командной строке в ответ на стандартную подсказку Автокада **Команда:** (Command:) введете открывающую скобку (. В этот момент AutoCAD настраивается на прием и вычисление выражения AutoLISP, которое должно иметь форму списка, т. е. ***начинаться открывающей скобкой и заканчиваться закрывающей, а элементы списка должны отделяться друг от друга пробелами***. Например, для получения суммы трех чисел вы можете ввести:

(+ 224.75 16.002 89.899)

Как только вы завершите ввод своего выражения нажатием клавиши <Enter>, AutoCAD вычислит указанное выражение и выдаст результат (рис. 1): 329.651 (десятичная точка в вещественных числах играет роль разделителя между целой и дробной частями).

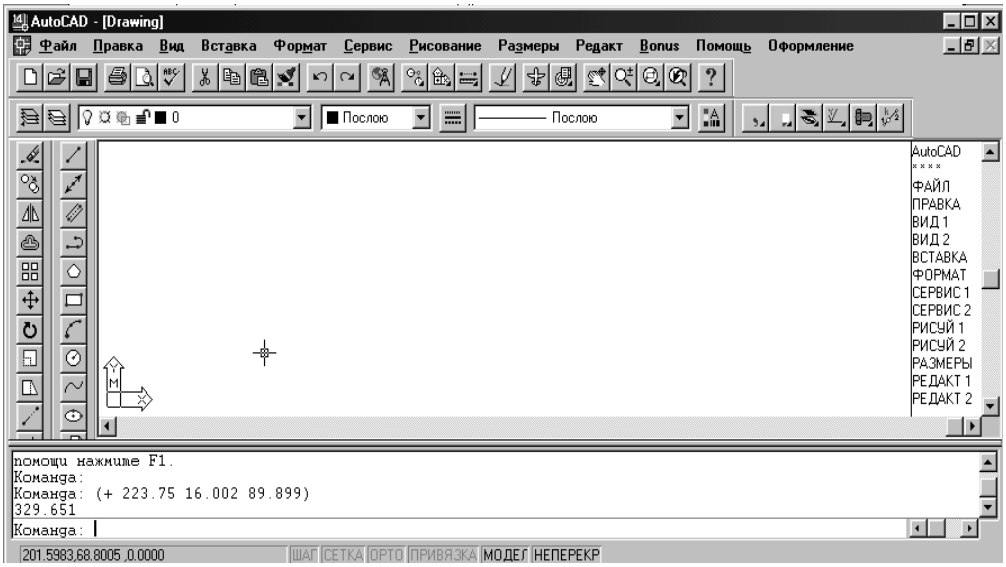


Рис. 1. Использование вычислителя выражений интерпретатора AutoLISP

3.1 Введение в Visual LISP

Visual LISP (VLISP) представляет собой интегрированную среду разработки (integrated development environment — IDE), обеспечивающую удобный и простой в использовании интерфейс, который помогает создавать код, отлаживать его и тестировать программы.

К числу полезных возможностей Visual LISP можно отнести следующие:

- *средство проверки синтаксиса* Syntax Checker и выделение синтаксических ошибок, облегчающее их исправление;
- *компилятор* File Compilation, обеспечивающий более быстрое выполнение программ;
- *отладчик*, поддерживающий пошаговое выполнение программы с целью локализации ошибок;
- *окна Inspect и Watch*, в которых можно просматривать значения переменных в процессе выполнения программы;
- контекстно-зависимая *справка*;

3.2 Открытие Visual LISP

Чтобы запустить Visual LISP нужно:

- либо ввести в командной строке AutoCAD команду **VLIDE,**
- либо выбрать в падающем меню Tools ⇒ Application (Инструменты ⇒Приложение) выбрать пункт Файл и загрузить файл **VLIDE.ARX**, который находится в папке **AutoCAD_R14\ Vlisp**.

С помощью одного из этих методов можно переключиться в Visual LISP в любой момент времени. На рис. 2 показано окно Visual LISP.

Открытие и загрузка файла AutoLISP с помощью Visual LISP

Войдя в среду Visual LISP, можно создать новый файл AutoLISP или открыть уже существующий.

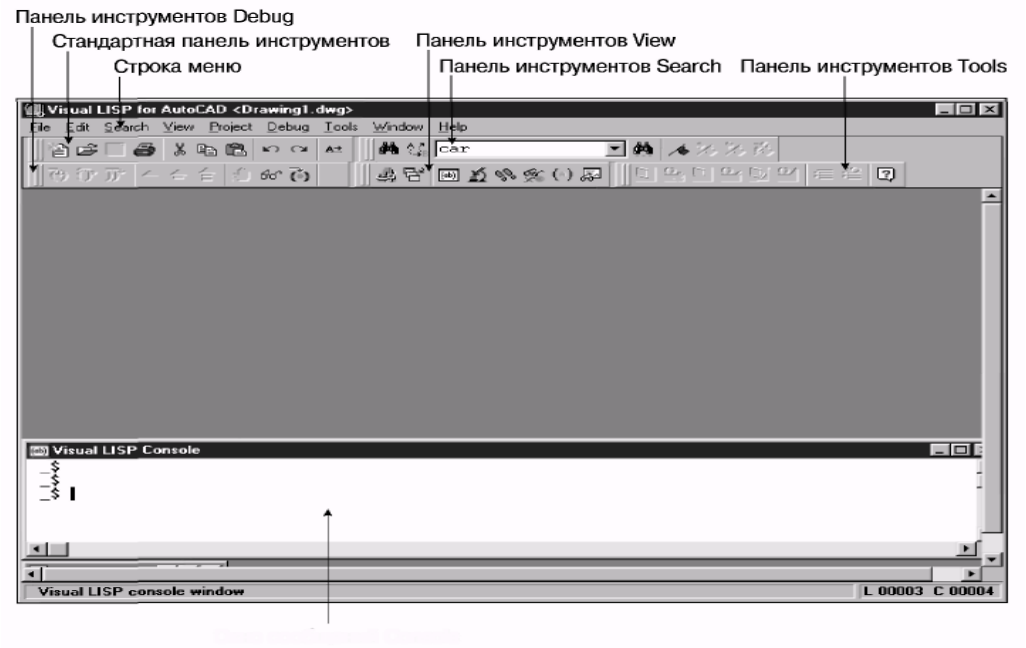


Рис. 2. Окно Visual LISP

Чтобы ***открыть новый*** файл ***в редакторе текста*** Visual LISP:

1. Выберите из меню Visual LISP команду File⇒New file (Файл⇒Новый).
2. Либо щелкните на кнопке New (Новый). Visual LISP откроет новый файл в отдельном окне.

Чтобы ***отредактировать*** или просмотреть файл в редакторе текста Visual LISP, выполните приведенные ниже действия.

1. Выберите из меню Visual LISP команду File⇒Open (Файл⇒Открыть).
2. В диалоговом окне Open File to Edit/View (Открытие файла для редактирования) найдите и выберите файл, который хотите открыть.
3. Щелкните на кнопке Open (Открыть). Visual LISP откроет выбранный вами файл в отдельном окне, как показано на рис. 3.

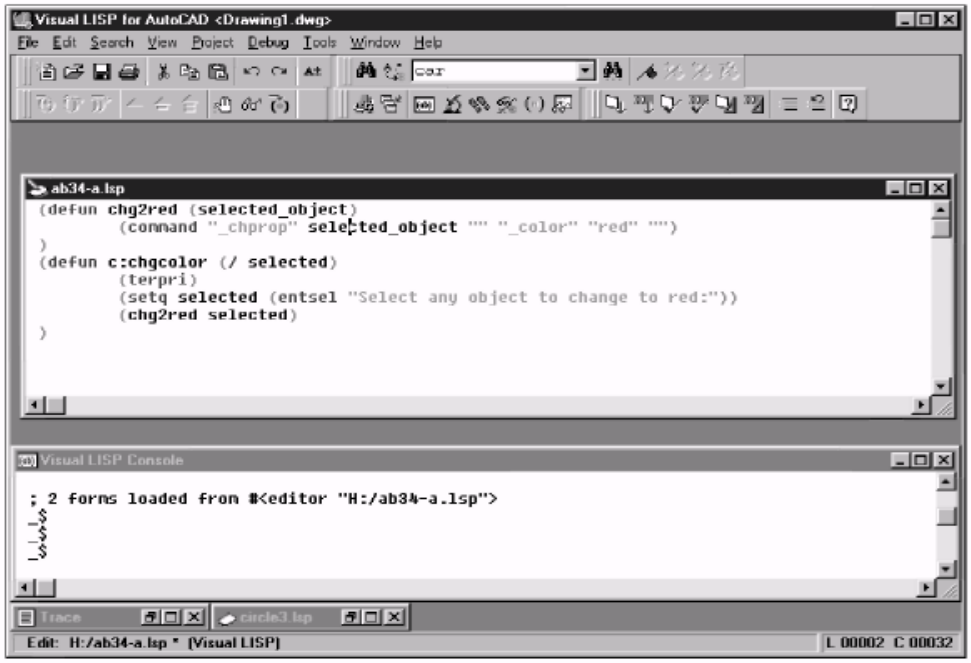


Рис 3. Файл AutoLISP, открытый в отдельном окне Visual LISP

После открытия файла в строке заголовка его окна отображается пиктограмма с изображением чистого листа бумаги, свидетельствующая о том, что файл еще не был изменен. Если в файл внести какие-либо изменения, то к пиктограмме добавится изображение карандаша, означающее, что файл был изменен.

В Visual LISP можно открыть любое количество файлов.

3.3 Использование интерфейса Visual LISP

В окне Visual LISP содержится множество средств, облегчающих процесс программирования. Эти средства и обеспечивают разительное отличие между созданием кода AutoLISP в новой и более ранних версиях AutoCAD. Если раньше код AutoLISP приходилось вводить в обычном текстовом редакторе, то теперь специальные средства редактора помогут вам отформатировать и создать код. Ниже перечислены наиболее полезные инструменты Visual LISP.

 **Форматирование программы**. Щелкните на пиктограмме Format Edit Window (Форматировать содержимое окна редактирования) панели инструментов Tools (или выберите в меню редактора команду Tools ⇒ Format Code (Сервис⇒Форматировать код)), чтобы структурировать код (и комментарии), сделав его более легким для восприятия. Структурированный код позволяет яснее увидеть уровни вложенности скобок. Конечно же, структурировать код можно и вручную, но возможность автоматического структурирования оценили все программисты AutoLISP.

 **Форматирование выделенного фрагмента программы**. Выделите фрагмент кода, который вы хотите отформатировать. Затем щелкните на пиктограмме Format Selection (Формат выделенного фрагмента) панели инструментов Tools (или выберите команду Tools⇒Format Code in Selection (Сервис⇒Форматировать код выделенного фрагмента)), чтобы структурировать только выделенный фрагмент кода.

 **Синтаксический контроль текста программы**. Щелкните на пиктограмме Check Edit Window (Проверить содержимое окна редактирования) панели инструментов Tools (или выберите в редакторе команду Tools⇒Check Text (Сервис⇒Проверить текст)), чтобы выполнить предварительную проверку файла до его загрузки Visual LISP. В процессе этой проверки в файле можно выявить несоответствие скобок, некорректные определения функций (попытки переопределить встроенные или защищенные функции), а также множество других типичных ошибок. Результаты проверки Visual LISP отобразит в окне Build Output.

 **Синтаксический контроль выделенного фрагмента текста программы.** Выделите фрагмент кода, который вы хотите проверить. Щелкните на пиктограмме Check Selection (Проверка выделенного фрагмента) панели инструментов Tools (или выберите в редакторе команду Tools⇒Check Selection (Сервис⇒Проверить выделенный фрагмент)), чтобы выполнить предварительную проверку лишь выбранного фрагмента кода. Результаты проверки Visual LISP отобразит в окне Build Output.

Поиск парной скобки. Наиболее типичной ошибкой при написании программ AutoLISP является неправильное соответствие скобок. Visual LISP позволяет перемещаться между соответствующими друг другу скобками и быстро проверять текущий уровень вложенности скобок в процессе разработки приложения. Для выполнения этой операции можно пользоваться командами меню Edit ⇒ Parentheses Match-ingoMatch Forward (Правка ⇒ Соответствие скобок ⇒ Вперед) и Edit ⇒ Parentheses Matching ⇒ Match Backward (Правка⇒Соответствие скобок⇒Назад). Однако гораздо удобнее использовать клавиши ускоренного доступа, чтобы не отрываться от кода, с которым вы работаете. Чтобы найти соответствующую закрывающую скобку, нажмите комбинацию клавиш **<Ctrl+]>.** Для нахождения соответствующей открывающей скобки используйте комбинацию клавиш **<Ctrl+[>.**

Вы можете выделить весь фрагмент кода, расположенный в соответствующих скобках. Чтобы выделить весь фрагмент **слева направо**, поместите указатель мыши **рядом** с открывающей скобкой и нажмите комбинацию клавиш **<Ctrl+Shift+]>** или просто дважды щелкните мышью. Чтобы выделить фрагмент справа налево (в обратном направлении), поместите курсор после закрывающей скобки и нажмите комбинацию клавиш <Ctrl+Shift+[> или дважды щелкните мышью.

 **Загрузка выделенного фрагмента текста программы**. Вы уже знаете, как загрузить код в активное окно. Но можно также загрузить выделенный фрагмент кода. Для этого выделите нужный фрагмент кода и щелкните на пиктограмме Load Selection (Загрузка выделенного фрагмента) панели инструментов Tools или выберите команду ToolsoLoad Selection.

 **Преобразование блока программы в комментарий**. Visual LISP поддерживает несколько стилей комментариев. В Visual LISP значительно упростился процесс добавления комментария с тройным символом "точка с запятой" (;). Чтобы создать такой комментарий, выделите текст, который должен служить комментарием и щелкните на пиктограмме Comment Block (Закомментировать блок) панели инструментов Tools.

 **Преобразование комментария в блок программы**. Чтобы удалить комментарий, отмеченный тройным символом "точка с запятой", щелкните на пиктограмме Uncomment Block (Раскомментировать блок) панели инструментов Tools.

Еще одним преимуществом Visual LISP является окно Console. Оно обычно располагается в нижней части окна Visual LISP и напоминает командную строку AutoCAD. Подобно тому, как выражения AutoLISP вводятся в командной строке (эта операция будет описана ниже в этой главе), их можно вводить и в окне Console, чтобы наблюдать результат выполнения. В каждой строке окна содержится приглашение _$.

Замечательное свойство окна Console заключается в том, что Visual LISP помнит все введенные вами выражения. Нажимая клавишу <Tab> вы можете повторить их последовательность, а с помощью комбинации клавиш <Shift+Tab> можно воспроизвести их в обратном порядке.

3.4 Закрытие файла и Visual LISP

Чтобы **закрыть файл** в Visual LISP, щелкните на пиктограмме Close. (Для этого можно также воспользоваться комбинацией клавиш **<Ctrl+F4>**.)

Чтобы **завершить работу** Visual LISP, щелкните на кнопке Close самого приложения или нажмите комбинацию клавиш **<Alt+F4>.** Как и при выходе из AutoCAD, вам будет предложено сохранить все внесенные изменения.

3.5 Панели инструментов Visual Lisp

Вывести на экран панели инструментов можно с помощью пункта **Toolbars** (Панели инструментов) падающего меню **View** (Просмотр). С его помощью на экране отображаются следующие панели инструментов: **Standard** (Стандартная), **Search** (Поиск), **Tools** (Инструменты), **Debug** (Отладка), **View** (Просмотр).

Панель инструментов просмотра: View.

Панель инструментов просмотра включает восемь основных кнопок:

 **Activate AutoCAD** (Активизировать систему AutoCAD).

 **Select Windows (**Выбрать окно).

 **LISP console** (Выбрать консоль AutoLisp).

 **Inspect** (Проверить).

 **Trace** (Трассировка ошибки).

 **Symbol service** (Обслужить символы).

 **Apropos** (Найти слово по фрагменту).

 **Watch windows** (Открыть окно наблюдений).

Панель инструментов Tools (Инструменты)

Панель инструментов Tools включает девять основных кнопок:

Load activate edit windows – Загрузить программу из активного окна редактирования.

Load selection – загрузить выделенное выражение.

Check edit windows – Проверить синтаксис программы в активном окне редактирования.

Check selection – Проверить синтаксис выделенного.

Format edit windows – Отформатировать программу в активном окне редактирования.

Format selection – Отформатировать выделенный фрагмент.

Comment block – Закомментировать блок.

Uncomment block – Разкомментировать блок.

Панель инструментов отладки Debug

Панель инструментов отладки включает девять основных кнопок и один индикатор в виде кнопки:

Step into – Шагнуть внутрь выражения.

Step over – Шагнуть с обходом.

Step out – Шагнуть вне выражения.

Continue – Продолжить отладку.

Quit – Выйти из текущего уровня отладки.

Reset – Вернуться на верхний уровень.

Toggle breakpoint – Установить (удалить) точку прерывания).

Add watch – Добавить выражение для просмотра.

Last break – Последнее прерывание.

Step indicator (Индикатор шага). На индикаторе отображаются круглые скобки и положение курсора, показывающего, что процесс отладки остановлен перед или после выражения.

Панель инструментов поиска Search

Панель инструментов поиска включает семь основных кнопок и раскрывающийся список:

Find – Найти.

Replace – Заменить.

- Раскрывающийся список с текстовым полем для поиска символов в активном окне текстового редактора или в окне консоли.

Find toolbars string – Найти строку текстового поля.

Toggle bookmark – Установить (удалить) закладку.

Next bookmark –Перейти к следующей закладке.

Previous bookmark –Перейти к предыдущей закладке.

Clear all bookmark –Удалить все закладки.

3.6 Отладка программ в среде Visual LISP

Отладка – как правило, наиболее трудоемкая стадия в разработке любой программы. Visual LISP включает мощный отладчик, который позволяет:

- запустить программу **Trace** (трассировка программы);
- отслеживать значения переменных во время выполнения программы;
- просмотреть последовательность значений выполняемых выражений (функций);
- запустить программу прерываний;
- производить пошаговое выполнение программы;
- просмотреть содержимое стека.

Для решения этих задач Visual LISP имеет следующие средства:

- **Break loop mode** (Режим прерываний в цикле), который позволяет остановить выполнение программы в определенных точках, просматривать и изменять значения объектов (переменных, символов, функций и выражений) во время прерывания;
- **Inspectors** (Инспектора) – позволяют получать детализованную информацию об объектах в окне **Inspector dialog Windows** (Диалоговое окно просмотра);
- **Watch Window** (Окно наблюдений) – позволяет наблюдать значения переменных во время выполнения программы. Содержание окна модифицируется автоматически;
- **Trace Stack Facility** (Средство трассировки стека) – позволяет просматривать функциональный стек вызовов – механизм, с помощью которого Visual LISP записывает последовательность выполнения функций;
- **Trace Facility** (Средство трассировки) – позволяет регистрировать обращения и значения отслеживаемых функций в специальном окне **Trace** (Трассировка).

Отладчик может приостановить выполнение программы до или после выполнения любого выражения.

Наиболее распространенный способ отладки – ***установка точек прерывания***. Рассмотрим процесс установки точки прерывания и отладки программы на примере выполнения первой тестовой задачи в текстовом окне редактора (рис. 4). Эта программа позволяет переменной с именем ***imy*** присвоить имя, введенное с клавиатуры. Назначить стиль выводимого текста, вывести на экран заданный текст и провести отрезок, используя в качестве концов отрезка точку, введенную с клавиатуры, и точку, полученную заданием относительных координат. В программе используются две переменные – ***pt*** (для координатной пары точки конца отрезка) и ***imy***. Переменная ***pt*** описана как локальная переменная, поэтому она удалится из памяти сразу после выполнения программы, а переменная ***imy*** останется в памяти.

Текст программы:

```
(defun li (/ pt)
  (command "_undo" "_back" "_y")
 (setq imy (getstring "\n Ваше имя?   "))
 (command "_style" "" "" "60" "3" "" "" "" "")
 (command  "_text" "3,300" "10" "Добро пожаловать в Visual LISP")
 (command "_text" "20,150" "0" (strcat "Дорогой  " imy "!!!"))
 (command "_text" "250,50" "0" "Успеха Вам !!!"
                  "_zoom" "_a")
 (princ "Input Point:")
 (setq pt (getpoint))
 (command "_line" pt "@843,210" "")
        )
```

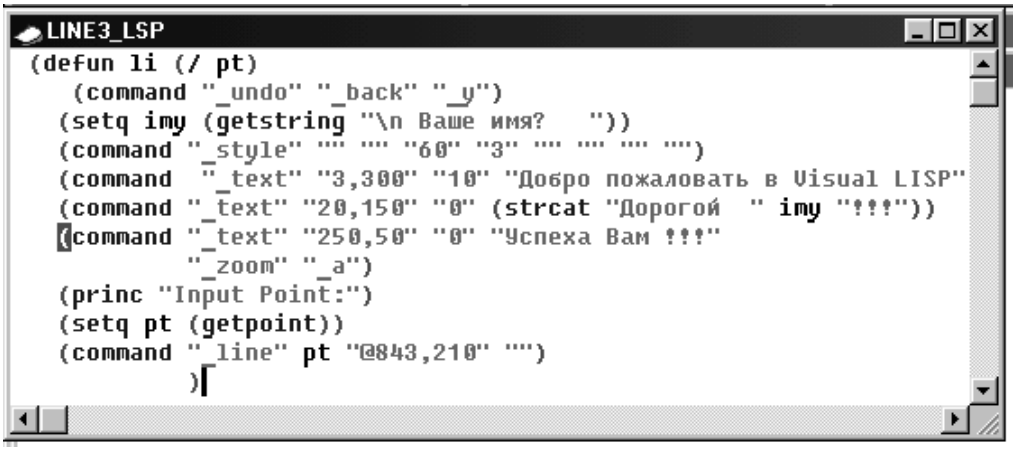


Рис. 4. Окно текстового редактора

Для установки точки прерывания необходимо установить курсор в нужную позицию открывающейся круглой скобки и щелкнуть по кнопке  **Toggle breakpoint** (Установить/ удалить точку прерывания). На экране появится точка прерывания (красный прямоугольник), если ее там не было. Точка прерывания устанавливается и убирается одной и той же клавишей. Курсор не может находиться внутри круглых скобок.

 Щелкните по кнопке **Load active edit window** (Загрузить текст активного окна редактирования). Появится окно консоли Visual LISP (рис. 5).

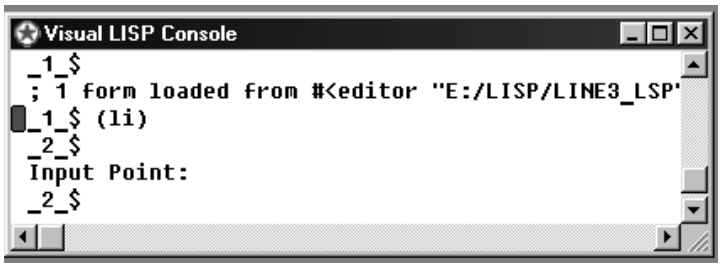


Рис. 5. Окно консоли Visual LISP

В окне консоли появится сообщение о загрузке файла. После этого его надо выполнить – ***набрать имя загруженной функции в круглых скобках – (li).*** Если функция описана как команда Автокада, то при работе из окна консоли в имя необходимо включить и префикс C:, например (C:LI). Результат работы функции показан на рис. 6.

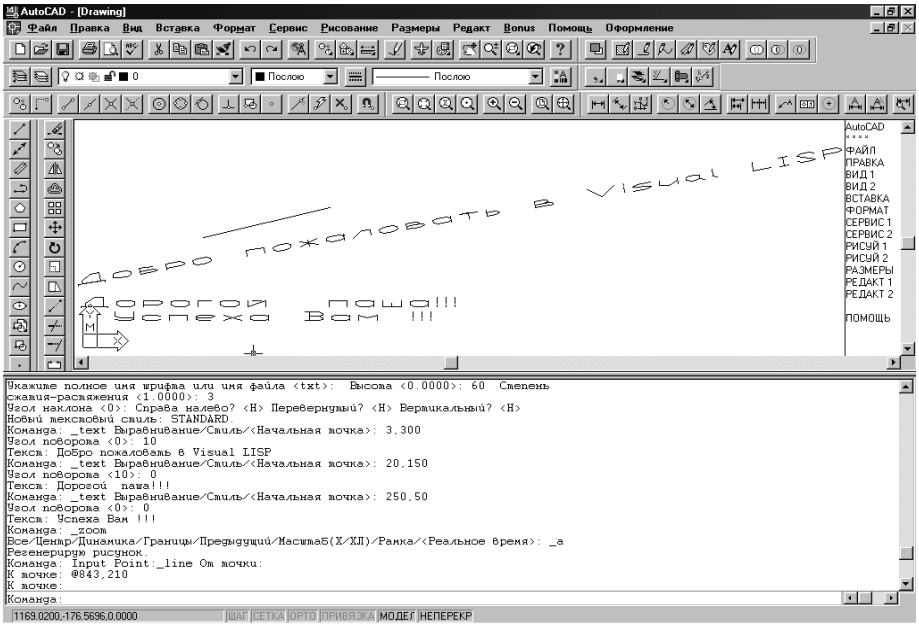
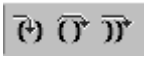


Рис. 6. Рабочее окно AutoCAD после выполнения пользовательской функции LI

Если работать не из окна консоли, а из текстового окна Автокада, то функцию лучше описать как команду Автокада, т.е. перед именем функции добавить ***c:***, в нашем случае первая строка будет выглядеть как

(defun c:li (/ pt), а вызов из командной строки Автокада необходимо производить без круглых скобок.

При отладке программы можно пользоваться командами меню Debug (Отладка)  . При пошаговой отладке основной командой являются  - **Step into** (Шагнуть внутрь выражения), **Step over** (Шагнуть с обходом), **Step out** (Шагнуть вне выражения). При просматривании программы по шагам кнопка  - **Step indicator** (Индикатор шага) активна.

Во время пошаговой отладки можно отследить значения каждого выражения. Для этого можно щелкнуть по кнопке  **Add watch** (Добавить выражение для просмотра) и в появившемся окне ввести имя переменной. Можно просто открыть окно наблюдений View⇒Watch Window (Просмотр⇒Окно наблюдений) или щелкнуть на клавише  . Результат показан на рис. 7.

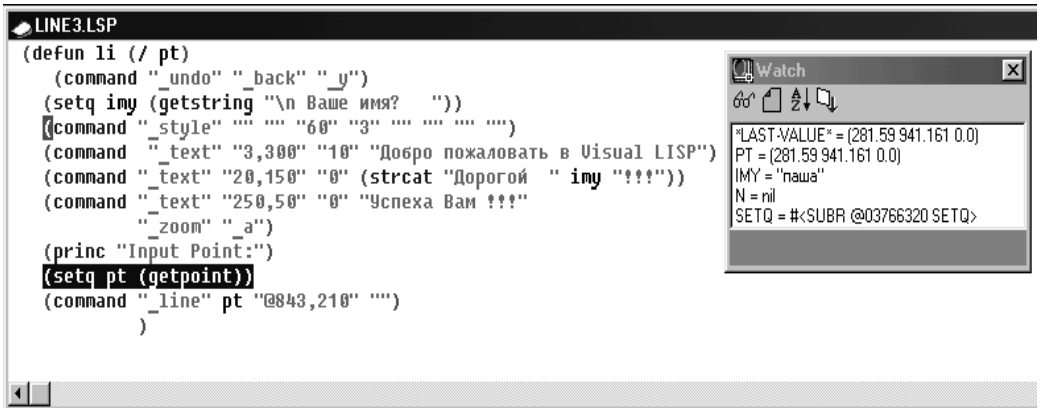


Рис. 7. Пошаговая отладка программы LI в текстовом окне Visual LISP с просмотром значений переменных в окне наблюдений

Клавиша  **Continue** (Продолжить отладку) позволяет выполнить программу до следующей точки прерывания, а если ее нет, то до конца. Клавиша  **Quit current level (**Выйти из текущего уровня отладки) завершает прерывание и возвращает на один уровень вверх, а клавиша  **Reset to the top level** (Вернуться на верхний уровень) завершает прерывание и возвращает на самый верхний уровень.

Кроме этого в падающем меню **Debug** (Отладка) можно установить следующие режимы управления выполнением программы:

- **Stop Once** (Остановка сразу) – заставляет остановиться при выполнении первого выражения;
- **Break on error** (Прерывание по ошибке) – активизирует автоматическое прерывание каждый раз, когда во время прерывания возникает ошибка;
- **Animate** (Оживлять) предполагает неоднократное использование команды **Step into** (Шагнуть внутрь выражения) с определенной задержкой. Окно редактора в режиме Animate отображает выполняемые выражения, а окно Watch (Окно наблюдений) постоянно модифицирует значения переменных.

Чтобы получить возможность просматривать значения переменных, необходимо добавить переменную в окно **Watch** (Окно наблюдений). Когда курсор находится на имени переменной, двойным щелчком мышки необходимо выделить его и щелкнуть на кнопке Add Watch (Добавить наблюдение). Можно также выбрать пункт Add Watch из контекстного меню по правой клавише мыши (курсор на имени переменной). На экране появится диалоговое окно Enter expression to watch (Введите выражение для наблюдений) с введенным именем переменной. Если окно Watch уже активно, то можно добавить дополнительные переменные к списку, щелкая по кнопке с очками на инструментальной панели в окне Watch.

Лабораторна робота №2.

4. Элементы языка AutoLISP

4.1 Вычисляемые выражения

Форма вычисляемых выражений языка AutoLISP:

(<функция> [<аргумент1> [<аргумент2> .. . [<аргументN>]...]])

Здесь <функция> - это имя функции, <аргумент1>, <аргумент2>, ..., <аргументN> - аргументы функции, разделяемые, по крайней мере, одним пробелом. Квадратные скобки в приведенной форме указывают на возможное отсутствие находящихся между ними аргументов. Количество аргументов зависит от синтаксиса функции. Существуют функции, которые вызываются без аргументов.

С точки зрения AutoLISP все, что заключено в круглых скобках, является списком, который следует обработать как вычисляемое выражение.

Основное правило AutoLISP - баланс скобок, т. е. количество открывающих скобок должно быть равно количеству закрывающих. Иначе AutoCAD, обнаружив дисбаланс (например, если пользователь забыл ввести конечную скобку), выдаст об этом сообщение.

4.2 Классификация функций языка AutoLISP

В языке AutoLISP определены более 150 различных операций, которые называются встроенными функциями. По назначению их можно подразделить на функции:

- ***для работы с числовыми данными***, реализующие арифметические операции, а также наиболее часто используемые математические функции. Эти функции позволяют вычислять координаты примитивов, рассчитывать длины, площади и т.п.
- ***операции сравнения*** для проверки выполнения различных условий, булевские функции ("и", "или", "не") и др., а также функции, организующие ветвления по условиям. С помощью этих функций можно, например, получать различные чертежи из одной программы.
- ***для работы со строками текста:*** формирование, сцепление, сравнение строк, выделение символов из строки и т.п. Эти функции позволяют, например, формировать технические требования на чертеже путём совмещения переменной и постоянной частей.
- ***для ввода с клавиатуры*** устройств указания, и вывода на экран и принтер, с помощью которых реализуется диалог пользователя с программой. Вывод на принтер позволяет получать из программы текстовые документы, например, спецификацию по сборочному чертежу.

- ***для создания и чтения текстовых файлов***, благодаря чему обеспечивается возможность связи по данным между различными программами на AutoLISPe.

- характерные для всех языков программирования и обеспечивающие компактное описание действий в программе за счёт таких конструкций, как ***циклы и подпрограммы***;

- характерные для языков типа LISP***: создание, анализ и преобразование списков.*** Поскольку данные о графических объектах-примитивах и блоках представляются в виде списков, то эти функции используются для обработки внутрипрограммных описаний графических объектов.

Специфику языка AutoLISP определяют функции, связанные с ***графикой*** и ***работой в среде графического редактора ACAD***:

- для внутрипрограммных геометрических построений, важнейшая из этих функций - ***определение точки***, заданной через другую точку, угол луча и расстояние по лучу. С помощью этой функции можно формировать из программы опорные точки примитивов чертежа, задавая их параметры с помощью переменных.

- для ***приёма геометрических данных***, т.е. данных, которые могут задаваться перемещением курсора на экране: точки, угла, расстояния;

- для ***выделения примитивов*** построенного на экране чертежа и наборов примитивов, выделения и изменения характеристик примитивов и блоков, анализа и изменения системных переменных и содержимого символьных таблиц ACAD;

- для включения в программу ***любой команды ACAD***. Причём аргументы и опции команды могут быть заданы не только из программы, но и в режиме графического диалога в точности так, как если бы эта команда выполнялась просто в редакторе ACAD.

4.3 Особенности языка

Скобки - основной управляющий символ языка LISP. Так как язык обрабатывает списки, то скобок там бесчисленное множество, и 50% ошибок программистов связано с подсчетом и неверным количеством скобок в программе.

Характерная черта языка - то, что выражения AutoLISP строятся тоже как списки. Это означает, что ***сначала*** пишется ***действие***, а потом - ***аргументы этого действия***.

Свойства выражений

- Каждая открывающая круглая скобка должна иметь закрывающую.

- Сразу после открывающей круглой скобки должен идти ***идентификатор операции*** (функции), выполняемой при вычислении выражения (имя функции).

- Следующие за именем функции аргументы функции должны быть отделены от имени функции и друг от друга по крайней мере одним ***пробелом*** (дополнительные пробелы и переводы строк игнорируются, так что выражение AutoLISP может занимать несколько строк, что в действительности и происходит).

- Каждое выражение вычисляется (выполняется) и ***результат возвращается***. Результатом может быть нуль (nil) или результат вычисления последнего подвыражения.

- С логической точки зрения любое возвращаемое значение либо истинно, либо ложно. Если значение выражения вычислено быть не может и возвращается нуль, то оно считается ложным. Если выражение вычисляется, то оно считается истинным - не-нуль (non-nil).

Выражение AutoLISP имеет вид:

(функция аргумент1 аргумент2 . .. аргументN)

Функция - имя операции (в том числе и арифметической), которая должна быть выполнена. ***Число аргументов*** может быть больше 3.

Произведение трех чисел: (* 2 3 4)

Вложенные выражения: (* 4.4 (- 4.3 (+ 3.2 1.1)))

Выражение анализируется AutoLISP ***слева направо***, пока не встретится скобка. Если встречается закрывающая скобка, то завершается анализ выражения, выполняется функция и вычисленное значение передается на более старший уровень вложенности или в AutoCAD. Если же встречается открывающаяся скобка, AutoLISP переходит к анализу выражения более младшего уровня вложенности и, пока не завершит его анализ, не перейдет к дальнейшему анализу выражения предыдущего уровня. Предел вложенности выражений - 100.

4.4 Типы данных

- ***Строковые переменные*** - совокупности букв и констант, заключенных в кавычки.
- ***Целые переменные*** - положительные или отрицательные целые числа (без дробей и десятичной точки). Целые числа представлены в машине двумя байтами и поэтому не могут выходить за диапазон (-32 768, 32 767).
- ***Действительные переменные*** - положительные или отрицательные числа с десятичной точкой. Особенность: если значение меньше 1, то нужно явно указывать 0 перед десятичной точкой, иначе будет выдаваться сообщение об ошибке.

Переменные любого из простых типов называют атомами. Списком называется набор разделенных пробелами атомов и/или списков, заключенных в круглые скобки.

Комментарии обозначаются "**;**" в начале строки. Все последующие выражения на данной строке интерпретатором игнорируются.

4.5 Арифметические операторы

(+ число1 число2 ...)

Возвращает сумму всех аргументов. Если все числа целые, то результат тоже будет целым; если же хотя бы одно число вещественное, все числа будут преобразованы в действительные и результат будет вещественным.

```
(+ 1 2)          возвращает      3
(+ 1 2 3 4.5)    возвращает          10.500000
(+ 1 2 3 4.0)    возвращает          10.000000
```

(- число1 [число2 ...])

Вычитает число1 из числа2, если более двух аргументов, то из первого аргумента вычитается сумма всех остальных, если задан один аргумент, то он вычитается из нуля (инвертируется его знак).

```
(- 50 40)              возвращает          10
(- 50 40.0 2)    возвращает          8.000000
(- 50 40.0 2.5) возвращает           7.500000
```

(- 8) возвращает -8

(* число1 [число2 ...])

Возвращает ***произведение*** всех чисел. Если задано только одно число, оно умножается на 1 и возвращается результат.

(* 2 3) возвращает 6
(* 2 3 4.0) возвращает 24.000000
(* 3 -4.5) возвращает -13.500000

(/ число1 [число2 ...])

Делит *число1* на *число2*, а если аргументов более двух, то первое число делится на произведение всех остальных.

(/ 100 2) возвращает 50
(/ 100 2.0) возвращает 50.000000
(/ 100 20 2.0) возвращает 2.500000
(/ 100 20.0 2) возвращает 2.500000
(/ 100 20 2) возвращает 2
(/ 135 360) возвращает 0
(/ 135 360.0) возвращает 0.375000
(/ 4) возвращает 4

(1+ число)

Увеличение целого или действительного аргумента на единицу.

(1+ 5) возвращает 6
(1+ -17.5) возвращает -16.500000

(1- число)

Уменьшение целого или действительного аргумента на единицу.

(1- 5) возвращает 4
(1- -17.5) возвращает -18.500000

(abs число)

Вычисление ***абсолютного значения*** действительного или целого числа.

(abs 100) возвращает 100
(abs -100) возвращает 100
(abs -99.25) возвращает 99.250000

(angle тчк1 тчк2)

Данная функция возвращает угол (в радианах), образованный лучом, направленным из точки *тчк1* в точку *тчк2*, и осью X, причем угол измеряется против часовой стрелки.

(angle '(1.0 1.0) '(1.0 4.0)) возвращает 1.570796
(angle '(5.0 1.33) '(2.4 1.33)) возвращает 3.141593

(angtos угол [представление] [точность])

Данная функция берет ***угол*** (вещественное число в радианах) и возвращает его ***преобразованным в строковую константу*** в соответствии с *представлением, точностью* и значением системной переменной Автокада DIMZIN. Аргумент *точность* – целое число, указывающее число цифр после запятой. Аргумент *представление* – целое число: 0 – градусы, 1 – градусы/минуты/ секунды, 2 – грады, 3 – радианы, 4 – топографические единицы.

```
(setq pt1 '(5.0 1.33))
(setq pt2 '(2.4 1.33))
(setq a (angle pt1 pt2))
```

тогда

(angtos a 0 0)	возвращает	“180”
(angtos a 0 4)	возвращает	“180.0000”
(angtos a 1 4)	возвращает	“180d0’0” ”

(atan число1 [число2])

Данная функция вычисляет ***арктангенс*** переменной *число1* в радианах, если не задано *число2*, *число1* может быть отрицательным; область допустимых значений от -π/2 до +π/2 радиан.

(atan 0.5)	возвращает	0.463647
(atan 1.0)	возвращает	0.785398
(atan -1.0)	возвращает	-0.785398
(angtos (atan -1.0) 0 4)	возвращает	“315.0000”

Если заданы оба числа, то возвращает арктангенс переменной число1/число2 в радианах. Если число2 - нуль, то в зависимости от знака переменной число1 возвращается + или - 1.570796 радиан

(atan 2.0 3.0)	возвращает	0.588003
(angtos (atan 2.0 3.0) 0 4)	возвращает	“33.6901”

(cos число)

Возвращает значение ***косинуса*** *угла*, заданного аргументом в радианах.

(cos 0)	возвращает	1.0
(cos pi)	возвращает	-1.0

(exp число)

Вычисляет значение ***экспоненциальной функции с основанием e*** в степени *число* и возвращает вещественное число.

(exp 1.0)	возвращает	2.718282
(exp 2.2)	возвращает	9.025013
(exp -0.4)	возвращает	0.670320

(expt основание степень)

Вычисляет ***значение экспоненциальной функции с указанным основанием*** и *степенью*. Если оба аргумента целые – результат целый, в любом другом случае – вещественный.

(exp 2 4)	возвращает	16
(exp 3.0 2.0)	возвращает	9.000000

(gcd число1 число2)

Возвращает наибольший ***общий делитель*** *числа1* и *числа2*, которые должны быть целыми.

(gcd 81 57)	возвращает	3
(gcd 12 20)	возвращает	4

(log число)

Возвращает ***натуральный логарифм*** *числа* как вещественное число.

(log 4.5)	возвращает	1.504077
(log 1.22)	возвращает	0.198850

(max число1 число2 ...)

Возвращает ***наибольшее*** из заданных *чисел*. Каждое число может быть вещественным или целым. Если все аргументы целые – результат целый, в любом другом случае – вещественный.

(max 4.07 -144)	возвращает	4.07
(max -88 19 5 2)	возвращает	19
(max 2.1 4 8)	возвращает	8.0

(min число1 число2 ...)

Возвращает ***наименьшее*** из заданных *чисел*. Каждое число может быть вещественным или целым. Если все аргументы целые – результат целый, в любом другом случае – вещественный.

(min 4.07 -144)	возвращает	-144.00
(min -88 19 5 2)	возвращает	-88
(min 2.1 4 8)	возвращает	2.1

(minusp число1)

Возвращает ***T***, если *число1* отрицательное действительное или целое, в противном случае – ***nil***.

(minusp -1)	возвращает	T
(minusp -4.293)	возвращает	T
(minusp 830.2)	возвращает	nil

(rem число1 число2 ...)

Данная функция делит *число1* на *число2* и возвращает ***остаток от деления***.

(rem 42 12)	возвращает	6
(rem 12.0 16)	возвращает	12.0
(rem 60 3)	возвращает	0

(sin угол)

Возвращает значение ***синуса*** *угла* как вещественное число, где угол выражен в радианах.

(sin 1.0)	возвращает	0.841471
(sin 0.0)	возвращает	0.0

(sqrt число)

Возвращает ***квадратный корень*** *числа* как вещественное число.

(sqrt 4)	возвращает	2.0
(sqrt 2.0)	возвращает	1.414214

Лабораторна робота №3

4.6 Операторы присваивания

(setq <переменная1><выражение1> [<переменная2> <выражение2>]…)

Данная функция ***устанавливает*** в *символ1* значение *выражения1*, в *символ2* значение *выражения2* и т.д.

Запись B=200 эквивалентна следующей:

(setq B 200)

(setq a 5.0) возвращает 5.0 и присваивает значение 5.0 переменной A. Когда бы ни использовалось A, оно будет равным вещественному числу 5.0.

```
(setq b 10          ;присвоение переменной B значения 10
      c "ABG"       ; присвоение C текстовой константы "ABG"
      D (COS 0.6)) ; присвоение D значения (COS 0.6)
```

Функция setq может использоваться с любым количеством аргументов, которое должно быть обязательно четным и не менее двух.

Функция setq - основное средство для сохранения значений, возвращаемых другими выражениями.

Пример:

(setq myd (- (+ 56.022 78.11) (+ 124.77 78.0)))

Значение выражения будет сохранено в переменной myd.

В дальнейшем прочитать значение переменной myd можно с помощью операции ! (восклицательный знак). Если в командной строке AutoCAD в ответ на стандартную подсказку **Команда:** (Command:) ввести !myd, то AutoCAD выдаст текущее значение переменной myd.

(set <'символ> <выражение>)

Данная функция ***присваивает*** *символу* с апострофом значение *выражения*.

(set 'B 200) присваивает символу 'B значение 200

4.7 Операторы ввода данных различного типа

Для получения данных с устройств ввода созданы специальные функции ряда **GET.** Их структура типична, сначала идет сам оператор, затем с вариациями - текст вопроса-подсказки. Этот текст будет выводиться в командную строку для того, чтобы пользователь понимал, что ему необходимо ввести в данный момент.

(getangle [точка] [текст запроса-подсказки])

Данная функция создает паузу для того, чтобы пользователь ***ввел угол***, и возвращает значение этого угла в радианах. Угол между заданным пользователем вектором и положительным направлением оси X, установленным в системной переменной ANGBASE, и против часовой стрелки (в пользовательской системе координат). Всегда в радианах, измерение угла происходит относительно текущего направления измерения углов.

(setq ang (getangle "В каком направлении? "))

(getcorner точка [текст запроса-подсказки])

Ввод координат. Возвращает координаты указанной пользователем точки и строит «резиновый» прямоугольник от *точки* до расположения курсора.

(getdist [точка] [текст запроса-подсказки])

Ввод ***расстояния между точками***. Данная функция создает паузу для того, чтобы пользователь ввел расстояние, *одну или две точки*. Функция всегда возвращает вещественное число, которое является расстоянием между двумя точками. Можно указать две точки на экране, можно первую точку указать внутри функции факультативно. При любых текущих единицах измерения эта функция всегда возвращает действительное число.

```
(setq dist (getdist ))
(setq dist (getdist '(1.0 3.5)))
(setq dist (getdist «Как далеко?» ))
(setq dist (getdist '(1.0 3.5) «Как далеко?» ))
```

(getenv имя переменной)

Возвращает строковое значение, присвоенное ***переменной среды*** AutoCAD.

(getint [текст запроса-подсказки])

Ввод ***целого числа***. Только с клавиатуры.

```
(setq num (getint))
```

(setq num (getint «Введите число»))

(getkword [подсказка])

Данная функция запрашивает у пользователя ключевое слово и ***возвращает*** ключевое слово ***как строковую константу***. Список имеющих смысл ключевых слов задается с помощью функции INITGET до вызова функции GETKWORD. Если введенное слово не является одним из заданых ключевых слов, Автокад повторит запрос.

(initget 1 "Да Нет")
(setq x (getkword «Вы уверены? (Да или Нет) »))

(getorient [точка] [текст запроса-подсказки])

То же самое, что и GETANGLE, за исключением того, что ***значение угла***, которое возвращает функция GETORIENT, не зависит от значений системных переменных ANGBASE и ANGDIR. Функция GETORIENT измеряет углы от нулевого направления вправо (восток), и углы увеличиваются в направлении против часовой стрелки. Выражает угол в радианах.

(getpoint [точка] [текст запроса-подсказки])

Позволяет ***ввести точку***, имеющую координаты текущей ПСК.

(setq p (getpoint))
(setq p (getpoint "Где?"))
(setq p (getpoint '(1.5 2.0) «Вторая точка»))

(getreal [текст запроса-подсказки])

Позволяет вводить ***действительное число***. Только с клавиатуры.

(setq p (getreal))
(setq p (getreal «Масштабный коэффициент: »))

(getstring [флаг пробела] [текст запроса-подсказки])

Запрашивает ввод ***текстовой константы***. Если флаг пробела указан и не равен нулю, то строковая константа может содержать пробелы, и завершением ввода считается нажатие клавиши ENTER, в противном случае строка не может содержать пробелы и клавиша ПРОБЕЛ работает как символ ввода.

(setq s (getstring «Ваше имя?»))
(setq s (getstring T «Ваше имя и фамилия?»))

(getpvar имя переменной)

Возвращает ***значение системной переменной*** AutoCAD.

Лабораторна робота №4

4.8 Работа со строками

(strcase <строка> [<признак>])

Данная функция берет строковую константу, указанную аргументом *строка* и возвращает ее копию, ***переведя все символы*** алфавита ***в верхний или нижний регистр*** в зависимости от аргумента *признак*. Если признак опущен или равен nil, то все символы алфавита в строке будут переведены в нижний регистр.

(strcase "Пример") возвращает «ПРИМЕР»
(strcase "Пример" T) возвращает «пример»

(strcat <строка1> <строка2>...)

Эта функция возвращает строку, которая является результатом ***сцепления*** *строки1*, *строки2* и т.д.

(strcat «с» «лово»)	возвращает	«слово»
(strcat «а» «б» «в»)	возвращает	«абв»
(strcat «а» «» «в»)	возвращает	«ав»

(strlen <строка>)

Эта функция возвращает ***длину в символах*** строковой константы *строка* как целую величину.

(strlen «слово»)	возвращает	5
(strlen «один» «два» «три»)	возвращает	10
(strlen «»)	возвращает	0

4.9 Списки

Простые списки

(list (<элемент1> [<элемент2> ... [<элементN>] ...]])

Функция **list** — это основная функция, позволяющая ***создать список.***

Данная функция берет любое число выражений и организует из них строку, возвращая список. В качестве аргументов *элементы,* из которых образуется список, могут выступать любые объекты, которыми оперирует AutoLISP. Самый распространенный список — это список из двух или трех вещественных чисел, представляющий точку. В качестве элементов списка могут выступать другие списки или точечные пары.

(list ‘a ‘b ‘c)	возвращает	(A B C)
(list ‘a ‘(b d) ‘c)	возвращает	(A (B D) C)
(list 3.9 6.7)	***возвращает пару***	***(3.9 6.7)***

Эта функция часто используется для определения ***значений координат*** дву- или трехмерных ***точек*** (список из двух или трех вещественных чисел).

Например: Определим значения координат для двух точек и используем их для отрисовки отрезка командой Автокада LINE.

Задаем координаты X и Y точек:

```
(SETQ   X1  ( -   S   H   20)
            Y1   ( -   L   40)
(SETQ   X2  ( + H     H    L)
            Y2   ( -   L   H)
```

Присваиваем переменным T1 и T2 значения координат точек:

```
(SETQ       T1   (LIST   X1    Y1)
                 T2    ( LIST    X2    Y2)
```

Обращаемся к команде ОТРЕЗОК

```
(COMMAND   “LINE“   T1   T2   “”)
```

Ассоциативные списки

Функция **assoc** применяется к списку, в котором элементами являются списки или точечные пары, и ***выбирает*** из этих элементов тот, у которого первый элемент имеет ***заданное значение***:

(assoc <код> <список>)

Если в аргументе *список* имеется несколько элементов, удовлетворяющих требуемому условию, то в качестве возвращаемого значения выбирается первый из них. Функция assoc — основной инструмент в операциях выборки из списка с характеристиками примитива AutoCAD того элемента, который содержит точечную пару с нужным DXF-кодом свойства (цвета, типа линии, веса и т. д.).

Допустим, что “al” определен как

```
((name   box)  (width   3)  (size   4.7263)  (depth   5))
```

тогда

(assoc ‘size al)	возвращает	(SIZE 4.7363)
(assoc ‘weight al)	возвращает	nil

Лабораторна робота №5

Работа со списками

AutoCAD представляет данные об объекте в виде списка, который содержит много других списков меньшего размера. Хотя на первый взгляд такая структура выглядит сложной (а правильнее сказать — непривычной), обработка списков предельно проста и понятна.

(append [<список1> [<список2> .. . [<списокN>] ...]])

***слияние** списков* в один.

(append ‘(a b) ‘(c d))	возвращает	(A B C D)
(append ‘((a) (b)) ‘((c) (d)))	возвращает	((A) (B) (C) (D))

(nth <номер> <список>)

извлечение из списка элемента по порядковому *номеру* (нумерация элементов списка выполняется слева направо и начинается с нуля).

(nth 3 ‘(a b c d e))	возвращает	D
(nth 0 ‘(a b c d e))	возвращает	A

(reverse <список>)

***переворот** списка*; возвращает список с его элементами, расставленными в обратном порядке.

(reverse ‘((a) b c))	возвращает	(C B (A))

(length <список>)

***длина** списка*, возвращает целое число, означающее число элементов в списке.

(length ‘(a b c d))	возвращает	4
(length ‘(a b (c d)))	возвращает	3

Точки

Список из трех вещественных величин является точкою. Использование такого списка в командах AutoCAD позволяет указывать точки для отображения примитивов.

Применение списков для задания координат

Список всегда заключается в круглые скобки. Одна из простейших и широко применяемых в AutoCAD списочных структур — набор координат точки, например:

```
(1.0 3.5 2.0)
```

Этот список задает точку с координатами `1.0,3.5,2.0` в прямоугольной системе координат X,Y,Z. Так как список представляет собой группу элементов, может потребоваться извлечь из него один или несколько элементов. В таблице приведен перечень функций извлечения элементов на примере списка `(1.0,3.5,2.0)`.

Для большей гибкости можно использовать функцию NTH, которая позволяет получить доступ к произвольному элементу списка и извлекает элемент указанного списка с нужным номером. Для этого ей нужно передать два аргумента: первый задает номер элемента списка (заметим, что нумерация элементов списка начинается с 0), а второй указывает на сам список.

Основные функции извлечения элементов списка

Функция	Результат	Описание

CAR	1.0	Возвращает первый элемент списка
CDR	(3.5 2.0)	Возвращает все элементы списка, кроме первого
CADR	3.5	Возвращает второй элемент списка
CADDR	2.0	Возвращает третий элемент списка

Как правило, имя списка— переменная, значение которой присваивается оператором setq, Например:

(setq corner (list 1.0 3.5 2.0))

(nth 0 corner)

В этом примере функция (nth 0 corner) возвращает значение 1.0, так как 1.0 является первым элементом списка corner.

Список создается функцией LIST. Если элементы списка являются ***константами (а не переменными)***, для его формирования можно использовать функцию QUOTE. Для ускоренного вызова функции QUOTE можно использовать одиночную кавычку (') (она задается той же клавишей, что и апостроф). Следующие две функции эквивалентны:

(setq corner (list 1.0 3.5 2.0))

(setq corner '(1.0 3.5 2.0))

Лабораторна робота №6

4.10 Условия и логические операции

Необходимость выполнить какую-либо процедуру при наличии некоторых условий возникает довольно часто. Условное выражение проще всего создать при помощи оператора IF, который выполняет одно действие, если выражение-операнд истинно, и другое, если оно ложно. Иными словами, результат операции зависит от истинности некоторого выражения.

Функции проверки выполнения условий

Функции проверки выполнения условий записываются так же, как и арифметические операции. Знаки условий: **>, <, =, >=, <=, /=.**

(> a b) ; эквивалентно записи a>b

Логический оператор - это функция, сравнивающая между собой два или более аргументов. Результат может быть либо истина (non-nil), либо ложь (nil).

(and выражение1 выражение2)

Возвращает результат выполнения ***логического И*** над списком выражений. Если любое из выражений имеет значение nil, то возвращается nil, в противном случае – T.

Пусть

(setq a 103) (setq b nil) (setq c “строка”)

тогда

(and 1.4 a c) возвращает T

(and 1.4 a b c) возвращает nil

(not аргумент)

Возвращает результат выполнения ***логического НЕ*** над своим аргументом.

Пусть

(setq a 123) (setq b “строка”) (setq c nil)

тогда

(not a) возвращает nil

(not b) возвращает nil

(not c) возвращает T
(not ‘()) возвращает T

(or выражение1 выражение2)

Возвращает результат выполнения ***логического ИЛИ*** над списком выражений. OR оценивает выражения слева направо до тех пор, пока не встретит выражение, результатом которого не является nil. Если такое найдено, OR прекращает дальнейшую оценку и возвращает T. Если все выражения равны nil, OR возвращает nil.

(OR nil 45 ‘()) возвращает T
(OR nil ‘()) возвращает nil

(= атом атом)

Данная функция является функцией сравнения ***равно***. Если все атомы эквивалентны, возвращается T, в противном случае – nil. В качестве аргумента могут использоваться как числа, так и строковые константы.

(= 4 4.0) возвращает T
(= 20 388) возвращает nil
(= 2.4 2.4 2.4) возвращает T
(= 499 499 500) возвращает nil
(= “я” “я”) возвращает T
(= “я” “ты”) возвращает nil

(/= атом атом)

Данная функция является функцией сравнения ***не равно***. Если первый атом не эквивалентен второму атому, возвращается T, в противном случае – nil. Функция не определена для числа аргументов более двух.

(/= 10 20) возвращает T
(/= 5.43 5.44) возвращает T
(/= “ты” “ты”) возвращает nil

(< атом атом)

Данная функция является функцией сравнения ***меньше чем***. Если первый атом меньше, чем второй, возвращается T, в противном случае – nil. При наличии более двух атомов, если каждый предыдущий атом меньше последующего, возвращается T.

(< 10 20) возвращает T
(< “b” “c”) возвращает T
(< 2 3 88) возвращает T
(< 2 3 4 4) возвращает nil

(<= атом атом)

Данная функция является функцией сравнения ***меньше или равно***. Если первый атом меньше или равен второму, возвращается T, в противном случае – nil. При наличии более двух атомов, если каждый предыдущий атом меньше или равен последующего, возвращается T.

(<= 10 20) возвращает T
(<= “b” “b”) возвращает T
(<= 2 3 1) возвращает nil
(<= 2 3 4 4) возвращает T

(> атом атом)

Данная функция является функцией сравнения ***больше***. Если первый атом больше второго, возвращается T, в противном случае – nil. При наличии более двух атомов, если каждый предыдущий атом больше последующего, возвращается T.

(> 120 20) возвращает T

(< “b” “c”)	возвращает	nil
(< 7 4 2)	возвращает	T
(< 77 4 4)	возвращает	nil

(>= атом атом)

Данная функция является функцией сравнения ***больше или равно***. Если первый атом больше или равен второму, возвращается T, в противном случае – nil. При наличии более двух атомов, если каждый предыдущий атом больше или равен последующего, возвращается T.

(>= 120 20)	возвращает	T
(<= “b” “b”)	возвращает	T
(>= 77 4 9)	возвращает	nil
(>= 77 4 4)	возвращает	T

Условное ветвление программ

Условные выражения позволяют управлять ходом выполнения программы на основе анализа некоторых данных. Результатом условного выражения будет либо T (истина), либо nil (ложь). Например, для оператора

(< 3 5) AutoLISP возвратит T (истина), поскольку 3 меньше, чем 5. Для оператора

(> 3 5) AutoLISP возвратит nil (ложь), поскольку 3 не больше 5.

Каждая программа имеет свою логическую структуру. Ветвление - это способ управления ходом выполнения программы. Условные операторы - средство управления ветвлением программ. Условные конструкции и селективные позволяют управлять ветвлением программы:

(IF тест-выражение выражение-тогда [выражение-иначе])

Данная функция исполняет выражение по условию. Если тест-выражение не nil, то исполняется выражение-тогда, в противном случае исполняется выражение-иначе, причем, последнее не обязательно.

(IF (= 1 3) “ДА!!!” “НЕТ…”)	возвращает	НЕТ…
(IF (= 2 (+ 1 1)) “ДА!!!” “НЕТ…”)	возвращает	ДА!!!
(IF (= 2 (+ 3 4)) “ДА!!!”)	возвращает	nil

Иногда по условию требуется выполнить не одно, а несколько выражений. Для этого используют функцию **PROGN**. Последовательность выражений, объединенных функцией PROGN, считается одним выражением.

```
(if ( = a b)      (progn
                              (setq   a    ( +    a    10))
                              (setq   b    ( -    b     10))
                        )
)
```

Предположим, требуется выбрать окружности с радиусом меньше 0.25. Вот пример оператора IF, в котором radius — переменная с присвоенным ранее значением:

```
(if  (<  radius  .25)
(princ "Радиус меньше .25")
(princ "Радиус не меньше .25") )
```

Здесь используется условное выражение (< radius . 25). ВыражениеЕсли_True — это (princ "Радиус меньше . 25"), а выражением Если_False— (princ "Радиус не меньше .25"). Этот условный оператор эквивалентен следующему утверждению: Если радиус меньше, чем .25, печатать "Радиус меньше .25", если нет— "Радиус не меньше .25".

Выражение Если_False можно опустить. Тогда AutoLISP выполнит выражение Если_Тгие, если условное выражение истинно, а если оно ложно, сразу перейдет к выполнению остальной части программы. В следующем упражнении вы встретите оба типа операторов IF.

```
(defun c:prim ( / entered_num)
    (setq entered_num (getint "\nВведите число:"))
    (if (< entered_num 3)
        (princ "\nВведенное число меньше 3")
        (if (= entered_num 3)
          (princ "\nВведенное число равно 3")
          (princ "\nВведенное число больше 3")
        )
    )
(princ)
)
```

Обычно функция IF при условии истинности логического выражения выполняет один оператор. Однако вам может понадобиться при выполнении этого условия осуществить несколько операций. Если условный оператор состоит из двух частей — Если_Тгue и Если_False, то нужно каким-то образом отделить эти группы операторов выполнения функции друг от друга. Для этого можно использовать функцию PROGN. AutoLISP, встретив эту команду, будет рассматривать все, что включено в *конструкцию* PROGN, как один оператор.

В следующем примере вы увидите условные операторы IF. Во втором операторе, если введено число 3, будет выведена не одна, а две строки. Это достигается объединением соответствующих строк кода оператором PROGN.

```
(defun c:prim1 (/ entered_num)
        (setq entered_num (getint "\nВведите число: "))
                (if (< entered_num    3)
                        (princ "\nВведенное число меньше 3.")
                (if (= entered_num    3)
                    (progn
                        (princ "\n Введенное число равно 3.") (terpri)
                        (princ "\nЭто и есть результат эксперимента.")
                     )
                     (princ "\n Введенное число больше 3.")
                 )
                 )
     (princ)
)
```

Лабораторна робота №7

4.11 Селективные конструкции

(cond (тест1 результат1 ...) ...)

Воспринимает в качестве аргументов любое число списков. Оценивает по очереди первые элементы списков, пока не встретится элемент, отличный от nil. Затем вычисляется то выражение, которое следует за тестом, и возвращается значение последнего выражения в субсписке. Если в субсписке только одно выражение (например, результат отсутствует), то возвращается значение выражения тест.

Пример: Использование функции COND для вычисления абсолютного значения числа:

```
(COND   ( ( minusp   a)  ( - a))
              (t  a)
)
```

Если в «а» было установлено значение –10, будет возвращено 10. Как видно, COND может использоваться как функция типа “CASE” (в случае…). Принято в качестве последнего (по умолчанию) выражения тест использовать символ T.

Пример: Присваивая ответ пользователя переменной “s”, функция COND проверяет его и возвращает 1, если ответ “y”, либо 0, иначе nil, если ответ “n”.

```
(COND  ( (=  s  “y”)  1)
         ( (=  s  “n”)  0)
         ( t  nil)
)
```

Лабораторна робота №8

4.12 Циклические конструкции

В любом языке программирования одной из важнейших управляющих структур являются циклы, которые позволяют повторять определенную процедуру несколько раз, пока она не будет выполнена над всеми объектами или элементами. Оператор цикла устанавливает условия начала выполнения операций, количество объектов, над которыми выполняются операции, и условия выхода из цикла.

(repeat число выражение1 выражение2 ...)

Функция повторяет каждое выражение указанное число раз и возвращает значение последнего выражения. В этой функции число представляется любой положительной целой величиной.

```
Пусть
        (setq  a  10)
        (setq  b  100)
тогда
        (REPEAT  4
                 (setq  a  (+  a  10))
                 (setq  b  (+  b  100))
        )
```

устанавливает A в 50, B в 500 и возвращает 500.

(while тест-выражение выражение)

Выход из цикла осуществляется по условию. Данная функция вычисляет тест-выражение и, если оно не равно nil, вычисляет другие выражения, затем вновь проверяет тест-выражение. Это продолжается до тех пор, пока тест-выражение не станет равным nil. Затем WHILE возвращает самое позднее значение последнего выражения.

```
Пусть
        (setq   a   1)
тогда
        (while  (=  a  10)
                (some-func  a)
                (setq  a  (1+  a))
        )
```

будет вызвана функция пользователя SOME-FUNC десять раз с A равным от 1 до 10. После этого будет возвращено 11, что является значением последнего выражения.

(foreach имя список выражение)

Эта функция, проходя по списку, присваивает каждому элементу имя и вычисляет каждое выражение для каждого элемента списка. Может быть задано любое число выражений. FOREACH выдает результат последнего вычисленного выражения.

```
(foreach  n  ‘(a  b  c)  (print  n)
```

эквивалентно

```
(print  a)        (print  b)        (print  c)
```

за исключением того, что FOREACH возвращает результат только последнего вычисленного выражения.

Общий подход к установке условий состоит во включении счетчика в оператор WHILE. Счетчик подсчитывает количество выполненных операций. Условием выхода из цикла может служить достижение счетчиком некоторого значения. Чтобы сформировать счетчик, присвойте переменной значение, с которого вы хотите начать выполнение операции. Затем напишите код (точнее, его фрагмент) для одного прохода. После этого увеличьте значение счетчика, используя выражение типа (setq counter (+ 1 counter))

Программа будет повторять выполнение операции до тех пор, пока счетчик (переменная counter) не достигнет установленного вами значения. Вот простой пример:

```
(defun c:process (/ counter)
      (setq counter 1)
      (while (< counter 6)
            (princ "Номер цикла обработки ")
            (princ counter)
            (terpri)
            (setq counter (+ 1 counter))
      )     )
```

В этом примере функция PROCESS присваивает переменной counter значение 1. Затем начинается цикл WHILE, условием выполнения которого является то, что значение переменной counter должно быть меньше 6. Внутри оператора WHILE печатается строка "Номер цикла обработки " и затем значение переменной counter. Функция (terpri) обеспечивает печать каждой следующей строковой константы с новой строки. Затем для переменной counter вычисляется новое, большее значение. Каждый раз после выполнения операторов тела цикла значение счетчика увеличивается на 1. Без приращения значения переменной counter условие в строке 3 всегда будет возвращать T и цикл никогда не закончится.

Если вы случайно создали программу с таким бесконечным циклом, то можете ***прервать выполнение*** AutoLISP-программы, нажав <Esc>, <Ctrl+Break> или выбрав в меню Visual LISP команду DebugoAbort Evaluation.

В приведенном примере цикл WHILE продолжается до тех пор, пока переменная counter меньше 6. Когда значение счетчика станет равным 6, программа остановится. Оператор WHILE возвращает последнее значение программы, так что AutoCAD в последней строке выведет 6.

Лабораторна робота №8

4.13 Работа с файлами

(open имяфайла режим)

Открыть файл - значит подготовить дескриптор файла к использованию его функциями AutoLISP. Поэтому возвращаемое функцией open значение должно присваиваться некоторой символьной переменной.

(setq a (open "file.ext" "r"))

Здесь a - дескриптор файла file.ext, открытого для чтения.

(close дескриптор файла)

Закрытие файла.

(read-line дескриптор файла)

Считывает строку символов с клавиатуры или из открытого файла. Возвращается считываемая строка.

(write-line строка дескриптор файла)

Записывает строковую константу строка на экран или в открытый файл. Возвращает строку, взятую в кавычки, и опускает кавычки при записи в файл.

Лабораторна робота №9

4.14 Работа с чертежом, доступ к командам AutoCAD

(command аргумент1 аргумент2 ...)

Аргументы - это команды AutoCAD и их опции. Каждый аргумент вычисляется и посылается в Автокад как ответ на соответствующий запрос. Имена команд воспринимаются Автокадом только после подсказки «Команда:».

```
(setq  pt1  '(1.45  3.23))
(setq  pt2  (getpoint  «Введите точку»))
(command  "ОТРЕЗОК"  pt1  pt2)
(command  "")
```

Эти выражения устанавливают значения для точки pt1, запрашивают точку pt2 и выполняют команду Автокада ОТРЕЗОК по двум заданным точкам. Аргументы функции COMMAND могут быть строковыми константами, действительными числами, целыми числами, точками, т.е. такими, какими ожидает их получить выполняемая команда в ответ на ее запросы. Пустая строка ("") равносильна нажатию клавиши пробела на клавиатуре. Вызов COMMAND без аргументов равносилен нажатию клавиш <CTRL>/<C> на клавиатуре и прерывает большинство команд Автокада.

4.15 Определение подпрограмм и функций

(defun <символ> <список аргументов> <выражение>...)

DEFUN - ***определяет функцию*** с именем <символ>. За функцией следует список аргументов (возможно пустой), за которым следует (факультативно) косая черта (slach) и имена одного или более локальных переменных функции. Косая черта должна отстоять от первого локальной переменной и последнего аргумента хотя бы на один пробел. Если нет аргументов и локальных символов, которые следует объявлять, за именем функции следует поставить пустые скобки.

Использование функций всегда начинается с оператора DEFUN. Можно определить три основных типа функций:

К первому типу относятся функции, определяемые оператором DEFUN, именам которых предшествует **C:.** Это позволяет использовать имя функции в командной строке AutoCAD. Такую функцию можно использовать, как любую команду AutoCAD и обращаться к ней без круглых скобок.

Определение функции можно формировать и без префикса C:. Этот тип функций наиболее удобен, если функция вызывается другими AutoLISP-выражениями. Для обращения к ней из командной строки необходимо заключить имя функции в круглые скобки.

Третий тип функций— S::STARTUP. Если функция определена с именем S: : STARTUP (обычно именно такое определение вы встретите в файлах acad-doc.lsp), она будет автоматически выполнена при инициализации нового чертежа или запуске AutoCAD.

```
(defun myfunc (x y) ... ) ;(функция берет два аргумента)
(defun myfunc (/ a d) ... ) ; (функция имеет две локальных переменных)
(defun myfunc (x / temp) ... ); (один аргумент и одна локальная переменая)
(defun myfunc () ... ) ;(без аргументов и локальных переменных)
```

За списком аргументов и локальных символов следует одно или более выражений, составляющих тело функции. Когда функция, определенная таким образом вызывается, ее аргументы вычисляются и связываются в список аргументов.

Локальные переменные: это переменные, которые используются внутри данной функции без изменения их связи на внешних уровнях. По умолчанию все переменные - глобальные.

Возвращение значения: функция будет возвращать результат последнего вычисленного выражения. Все предыдущие выражения будут иметь только побочный эффект.

```
(defun add10 (x)
    (+ 10 x);                    возвращает ADD10
     (add10 5) ;                 возвращает 15
     (add10 -7.4) ;      возвращает 3.60000
(defun dots (x y / temp)
              (setq temp (strcat x "..."))
              (strcat temp y)
              ) ;                 возвращает DOTS
(dots "a" "b")                    возвращает "a...b"
              (dots "from" "to") возвращает "from...to"
```

Никогда не используйте имена встроенных функций в качестве переменных, так как это сделает недоступными встроенные функции.

При разработке функций можно использовать метод рекурсии, то есть при определении функции эта функция может ***вызывать саму себя***.

Например: Вычислим показательную функцию с целым положительным коэффициентом – Y(A,N) = A^N.

```
(DEFUN POC (A N)    ; ОПРЕДЕЛЕНИЕ ПОКАЗАТЕЛЬНОЙ ФУНКЦИИ
        (IF (=  N  1)  A
                (* A  (POC  A  (-  N  1))))
)
```

Вызов функции (POC 5 4).

Результат 625.

Лабораторна робота №10

Локальные и глобальные переменные

Имя переменной — это символ, которым можно оперировать в данной программе. Важным свойством переменных является возможность присваивать им значения. Существует два типа переменных — глобальные и локальные.

Глобальная переменная доступна всем функциям AutoLISP, которые загружены в чертеж. К ней можно обратиться и получить сохраненное значение и после завершения работы программы, в которой она определена. Глобальные переменные следует использовать в том случае, если их значения должны быть доступны не только во время выполнения функции, но и в процессе всего сеанса работы над проектом. Они могут хранить фиксированное значение, значение, используемое различными функциями, или промежуточное значение, необходимое для отладки программы. ***Любая переменная, которая специально не определена как локальная, является глобальной.***

Локальная переменная может принимать некоторое значение только в процессе выполнения функции. Как только функция завершает работу, значение локальной переменной становится недоступным и память,

занимаемая этой переменной, освобождается. Локальные переменные следует использовать лишь в том случае, если вы твердо уверены, что их значения не понадобятся в других функциях. Локальные переменные облегчают процесс отладки программы, поскольку их действие распространяется лишь на тело самой функции. Как правило, большинство переменных должны быть локальными. Локальные переменные создаются и объявляются в операторе DEFUN после косой черты, как показано в следующем примере: **(defun list-objects (/ counter sset)...**

Глобальные переменные могут преподносить неожиданные сюрпризы. Они могут вызвать сбои в работе программы, поскольку их значения существуют постоянно, но их поиск, а значит, и отладка часто бывают затруднены. Имена глобальных переменных принято отмечать слева и справа звездочками, например: *aGlobal*. Если придерживаться такого синтаксиса, то глобальные переменные будет легко обнаружить в теле программы.

5. Пример выполнения программы параметрического изображения многоступенчатого объекта в диалоговом режиме.

1. Откройте AutoCAD и создайте новый чертеж, используя опцию Start from Scratch (Без шаблона).

2. Запустите Visual LISP, набрав в командной строке команду VLIDE. Откройте уже созданный файл или создайте новую программу, например, тестовую задачу 2.

```
;функция создания параметрического изображения
;многоступенчатого объекта в диалоговом режиме
(defun c:mn ()      ;начало создания функции
 ; (setvar "_blipmode" 0)  ;отключение изображения маркера
  ; установка границ чертежа и границ отображения
  (command "_limits" "0,0" "297,210" "_zoom" "_A")
 ; (textscr)
  (setq N (getint "\n Введите число ступеней N= ")
        xt (getreal "\n Введите координату X базовой точки XT= ")
        yt (getreal "\n Введите координату y базовой точки YT= ")
        BT (LIST XT YT))
  (REPEAT N
   (SETQ D (GETREAL "\n Введите диаметр ступени D= ")
         L (GETREAL "\n Введите длину ступени L= ")
         R (* D 0.5))
   ; определение характерных точек ступени объекта
   (SETQ T1 (POLAR BT (/ PI 2) R)
         T2 (POLAR T1 0 L)
         T4 (POLAR BT (* 1.5 PI) R)
         T3 (POLAR T4 0 L) )
   (COMMAND "_PLINE" BT "_W" 1 1 T1 T2 T3 T4 "_C")
   (SETQ BT (POLAR BT 0 L)))
  )
```

3.  Сохраните созданную программу на диске C: в студенческом разделе, в папке своей группы в файле с именем MNOG.LSP.

4.  Щелкните на пиктограмме Load Active Edit Window панели инструментов. В окне Console (Консоль) Visual LISP Вы увидите подтверждение загрузки файла MNOG.LSP:

```
; 1 form loaded from #<editor "E:/LISP/MNOG.LSP">
_$
```

5.  Щелкните на пиктограмме Activate AutoCAD (Активизировать AutoCAD) панели инструментов View (Просмотр) Visual LISP, и вы снова перейдете в AutoCAD.

6. Теперь, после загрузки программы необходимо выполнить функцию, для чего введите в командную строку имя функции MN.

Созданную функцию можно выполнить не покидая редактор Visual LISP. Для этого необходимо перейти в окно Консоли и набрать имя функции в круглых скобках. Если имени функции предшествовал префикс C:, то его также надо включить. При необходимости редактор сам передаст управление в AutoCAD и вернет Вас обратно в Visual LISP. На рис. 8 показан запуск созданной функции из окна Консоли.

7. В ответ на приглашение Введите число ступеней N=, количество ступеней создаваемого объекта, координаты базовой точки и по каждой ступени диаметр ступени D= и длину ступени L=.

8. На экране появится многоступенчатый вал (рис. 9).

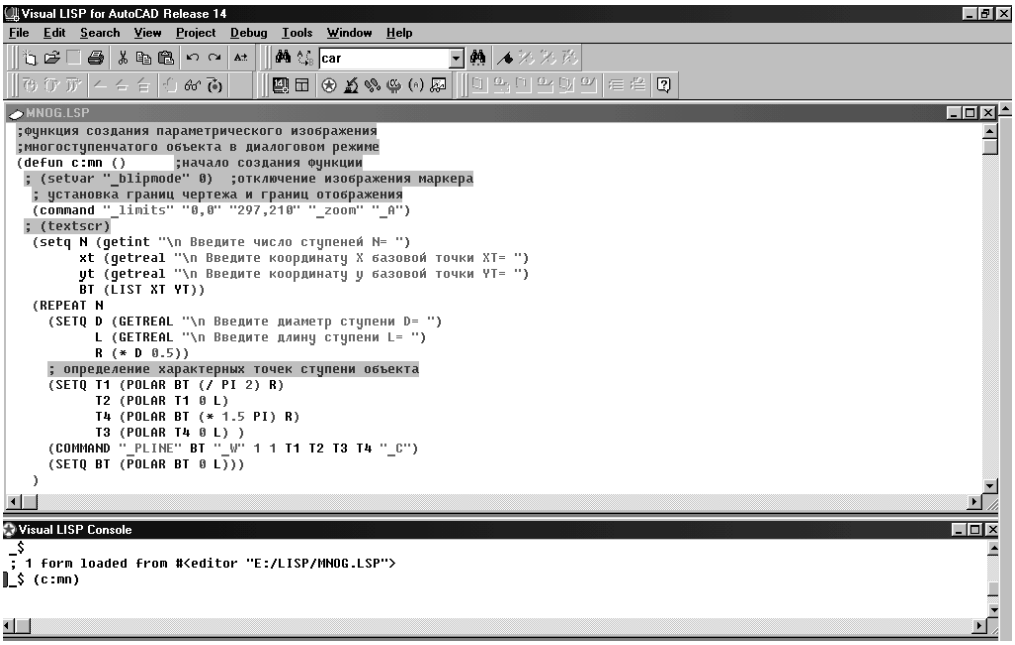


Рис. 8. Запуск созданной функции MN из окна Консоли