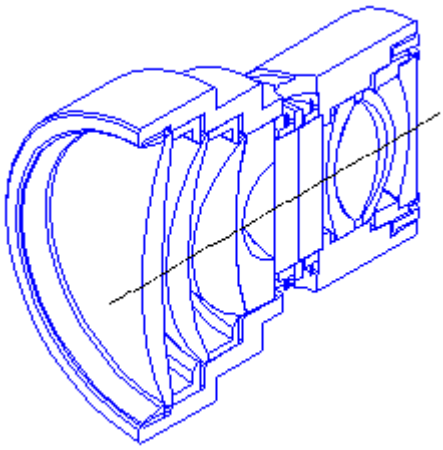
МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ИНСТИТУТ
ТОЧНОЙ МЕХАНИКИ И ОПТИКИ
(ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)

Н.Д. Толстоба

СИСТЕМЫ АВТОМАТИЗИРОВАННОГО КОНСТРУИРОВАНИЯ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ



Санкт-Петербург
2002

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ИНСТИТУТ
ТОЧНОЙ МЕХАНИКИ И ОПТИКИ
(ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ)

Н.Д. Толстоба

СИСТЕМЫ АВТОМАТИЗИРОВАННОГО КОНСТРУИРОВАНИЯ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Санкт-Петербург

2002

Толстоба Н.Д. Системы автоматизированного конструирования. Методические указания. - СПб, 2002. - 54 с.

Приводятся варианты индивидуальных заданий и даются методические указания к лабораторным работам по разделам курса: программирование на AutoLISP, программирование в кодах.

Для студентов оптических и приборостроительных направлений и специальностей.

Пособие подготовлено на кафедре Прикладной и компьютерной оптики Санкт-Петербургского государственного института точной механики и оптики (технического университета).

Рецензент: д.т.н., проф., Н.Б. Вознесенский

Одобрено на заседании кафедры Прикладной и компьютерной оптики ___ _____ 2002 г., протокол № ___.

Введение

Данное методическое указание представляет собой справочный материал по программированию в среде автоматизированного проектирования AutoCAD.

Рассмотрены примеры программ, тестовые задания, вопросы составления параметрических чертежей.

Пособие предполагается использовать в лабораторных циклах курсов «Компьютерное проектирование оптических приборов» и в процессе выполнения курсовых, комбинированных курсовых проектов, бакалаврских работ и дипломного проектирования в рамках специализации «Оптические приборы».

1. ПРОГРАММИРОВАНИЕ на AutoLISP

Введение

Цель этой главы - познакомить читателя с языком программирования AutoLISP, на котором были написаны многие встроенные функции AutoCAD. В лекции в доступной форме описываются особенности языка, без особых углублений в подробное описание функций, которое Вы можете найти в любом справочнике.

1.1. Назначение и возможности языка AutoLISP

Графический язык программирования AutoLISP является расширением языка программирования LISP. LISP- это язык высокого уровня, ориентированный на обработку списков, который выбран в качестве базового потому, что графические примитивы (начиная с точки), блоки, наборы примитивов и блоков удобно представляются в виде списков.

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. Возможности применения AutoLISPa весьма широки и разнообразны.

Наиболее характерны следующие классы применений

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 обогащения, 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. Тогда работа конструктора в системе AutoCAD будет сводиться к поиску нужных объектов (сборочных единиц, деталей) или частей чертежа, обращению к

соответствующим LISP-программам и ответам на вопросы этих программ.

Есть ещё одно очень важное обстоятельство: хранение графических данных в виде набора программ на AutoLISPe даёт возможность в десятки и сотни раз сократить требуемую память на магнитном диске 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. Программа 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.

Расширение системы команд графического редактора ACAD и построение на основе универсального редактора специализированных САПР, имеющих гораздо более простой и естественный для пользователей язык, ориентированный на конкретную предметную область. В этом случае хорошим дополнением к AutoLISPу является возможность создания пользовательских меню.

1.2. 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.
- для проверки выполнения различных условий **операции сравнения**, булевские функции ("и", "или", "не") и др., а также функции, организующие ветвления по условиям. С помощью этих функций можно, например, получать топологически различные чертежи из одной программы.
- для работы **со строками текстов**: 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.

1.3. Особенности языка

Скобки - основной управляющий символ языка LISP. Так как язык обрабатывает списки, то скобок в программе бесчисленное множество, и 50% ошибок программистов связано с невнимательностью к учету скобок в программе. Необходимо сосредоточиться.

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 ... аргументN)

Функция - имя операции (в том числе и арифметической), которая должна быть выполнена. Число **аргументов** может быть больше 2.

Произведение трех чисел: (* 2 3 4)

Вложенные выражения: (* 4.4 (- 3.3 (+ 2.2 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.

1.4. 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

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

1.5. Операторы

Арифметические операторы

(+ число1 число2 ...) (* число1 число2 ...)

(- 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).

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

(atan число1 [число2]) Если не задано **число2**, то возвращает арктангенс переменной **число1** в радианах, область допустимых значений - **[-pi,pi]** радиан. 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 (+90o или -90o).

(abs число) Вычисление абсолютного значения действительного или целого числа.

(cos число) Возвращает значение косинуса угла, заданного аргументом в радианах.

(exp степень) Вычисляет значение экспоненциальной функции с основанием e и аргументом, равным степени.

(expt основание степень) Вычисляет значение экспоненциальной функции с указанным основанием и степенью.

(gcd число1 число2 ...) Возвращает наибольший общий делитель.

(log число) Натуральный логарифм аргумента.

(max число1 число2 ...) Наибольший аргумент.

(min число1 число2 ...) Наименьший аргумент.

(rem число1 число2 ...) Остаток от деления переменной **число1** на переменную **число2**.

(sin число) Возвращает значение синуса угла.

(sqrt число) Извлекает квадратный корень из аргумента.

Операторы присваивания

Из несложных операторов используется функция **setq**. Запись **B=200** эквивалентна следующей:

(setq B 200)

Операторы ввода

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") Возвращает угол между заданным пользователем вектором и положительным направлением оси X. Всегда в радианах.

(getcorner точка "текст запроса-подсказки") Возвращает координаты указанной пользователем точки.

(getdist точка "текст запроса-подсказки") При любых текущих единицах измерения эта функция всегда возвращает действительное число.

(getenv имя переменной) Возвращает строковое значение, присвоенное переменной среды DOS.

(getint "текст запроса-подсказки") Ввод целого числа. Только с клавиатуры.

(getorient точка "текст запроса-подсказки") То же самое, что и getangle, но измерение угла происходит относительно текущего направления измерения углов.

(getpoint точка "текст запроса-подсказки") Позволяет ввести точку.

(getreal "текст запроса-подсказки") Позволяет вводить действительное число. Только с клавиатуры.

(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 терминатор ввода.

(getpvar имя переменной) Возвращает значение системной переменной AutoCAD.

Операторы вывода

(prin1 выражение дескриптор_файла) Выражение выводится на экран и возвращается в AutoLISP. Если указан дескриптор файла и файл открыт, то запись идет сразу на два устройства: на экран и в файл. Печатается только указанное выражение; переход на новую строку не осуществляется, и никакие пробелы не печатаются.

(princ выражение дескриптор_файла) То же, что и prin1, но управляющие символы не расшифровываются, а выводятся на экран.

(print выражение дескриптор_файла) То же, что и prin1, но перед печатью осуществляется переход на другую строку.

(prompt строка_символов) Выводит сообщение в командную строку и возвращает nil.

(alert строка_символов) Выводит сообщение в диалоговое окно, и ожидает нажатия на "ОК" от пользователя.

Работа со строками

(strcase <строка> [<признак>]) - 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.

(strcat <строка1> <строка2>...)Эта функция возвращает строку, которая является результатом сцепления <строки1>, <строки2> и т.д.

(strlen <строка>)Эта функция возвращает длину в символах строковой константы <строка> как целую величину.

Списки

Простые списки

Функция list — это основная функция, позволяющая создать список: **(list (<элемент1> [<элемент2> ... [<элементN>] ...]])**

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.

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.).

Работа со списками

(append [<список1> [<список2> ... [<списокN>] ...]]) - слияние списков в один;

(nth <номер> <список>) - извлечение из списка элемента по порядковому номеру (нумерация элементов списка выполняется слева направо и начинается с нуля);

(reverse <список>) - переворот списка;

(length <список>) - длина списка;

Точки

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

Условия и логические операции

Условия - 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.

(not аргумент) Возвращает результат выполнения логического НЕ над своим аргументом.

(or выражение1 выражение2) Возвращает результат выполнения логического И над списком выражений.

Условное ветвление программ

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)**

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))
  )
)
```

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.

Циклические конструкции

(repeat число выражение1 выражение2 ...) Функция повторяет операторы указанное число раз.

(while тест-выражение выражение1 выражение2 ...) Выход из цикла осуществляется по условию.

(foreach имя список выражение) Эта функция, проходя по списку, присваивает каждому элементу имя и вычисляет каждое выражение для каждого элемента списка.

1.6. 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.

Подробно работа с файлами рассмотрена в Приложении В.

1.7. 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.

```
(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 переменная)
```

```
(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 2.60000
(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 или символов как <**символ**>, так как это сделает недоступными встроенные функции.

1.8. Создание и загрузка файла программы

Необходимо в файле с расширением lsp записать текст программы. Это обычные текстовые файлы, поэтому можно использовать обычный текстовый редактор, но при условии, что выбранный текстовый редактор не добавлет в файл служебные символы. Можете использовать Aditor или Notepad.

Загрузка программы

Простая:

Простая загрузка - это загрузка одного файла. Для такой загрузки не требуется особых усилий ни в одном из представленных способов загрузки.

Меню **`Tools-Load Application`**, выводится окно:

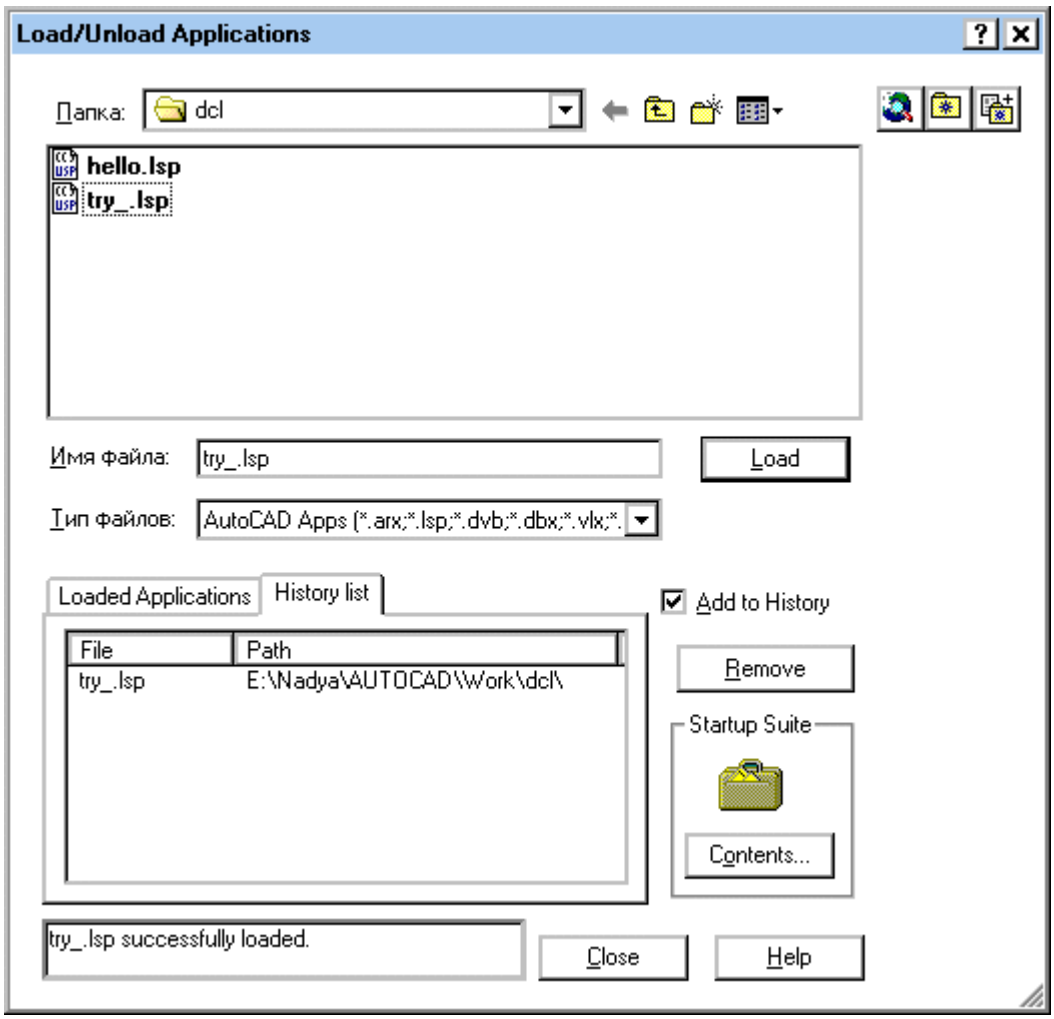


Рис 1. Диалоговое окно для загрузки программы

Тот же результат (то же окно диалога) Вы получите на экране при вводе команды appload из командной строки AutoCAD.

Также при фиксированном расположении Вашего загружаемого модуля, Вы можете просто записать строку для его загрузки в командной строке AutoCAD:

```
(load "диск:\\каталоги\\файл.lsp")
```

Комплексная загрузка

Комплексная загрузка - загрузка сразу нескольких файлов или библиотеки файлов. В таком случае рекомендуется создать файл-загрузчик, в котором будут перечисляться все файлы, необходимые Вам для работы. Можно составить файл из строк типа:

```
(load "диск:\\каталоги\\файл1.lsp")
(load "диск:\\каталоги\\файл2.lsp")
...
(load "диск:\\каталоги\\файлN.lsp")
```

Такой вариант наиболее прост в обращении и легко корректируется. А загрузку самого файла-списка можно "повесить" на кнопку.

1.9. Анализ ошибок

- При загрузке файла с текстом программы, интерпретатор AutoCAD почти ничего не пишет, поэтому необходимо себе помочь и сделать вывод надписей для ориентирования в процессе загрузки файла. После каждого блока (defun имя...) необходимо вставить (prompt "имя"). Тогда при отладке будут выводиться надписи с именами загруженных программ.
- Большинство ошибок - в подсчете и правильной расстановке скобок. И надпись об этом будет содержать "Malformed list".
- Malformed string - ошибка в расстановке кавычек.
- Syntax error – несколько вариантов ошибок – или в (setq) нечетное количество аргументов, или в (if) аргументов больше трех, или неверное использование (defun)

1.10. Ошибки, проявляющиеся после запуска программы:

error: Too many arguments - слишком много аргументов. Обычно появляется в блоке IF, когда позабыл автор об использовании (progn) и много операторов пытается запустить в теле IF.

error: Too few arguments - мало аргументов. Проявляется, когда мало аргументов указано для функции.

error: bad argument type - неверный тип агрумента, подаваемого в функцию.

Схематика вывода информации об ошибках:

Мы запускаем команду:

```
Command: ($getval "Введите что-нибудь!" "ку")
```

С заведомо неверным типом данных №2. Второй аргумент должен быть типа REAL, а не строка.

В результате сейчас получим ошибку:

```
error: bad argument type
(RTOS DEFLT)
(STRCAT "\n" STRING " <" (RTOS DEFLT) ">: ")
(SETQ QUESTION (STRCAT "\n" STRING " <" (RTOS DEFLT) ">: "))
($GETVAL "Введите что-нибудь!" "ку")
*Cancel*
```

Эта надпись обозначает следующее:

- сперва выводится тип ошибки - error: bad argument type
- потом пишется выражение, в котором произошла ошибка - **(RTOS DEFLT)**

- затем идет последовательное восстановление вложенности выражений, содержащих ошибку, до тех пор, пока мы не достигнем вложенности выражения **DEFUN**, тогда последней строкой выводится выражение, из которого произошел запуск функции, содержащей ошибку. Таким образом можно проследить расположение ошибочной команды.

Программирование как таковое

Хотелось бы обратить внимание Читателя на то, что называется стилем программирования. Несложные правила изложены в ниже, прошу придерживаться их при написании кода программы. Также хочу обратить Ваше внимание на алгоритмирование своей задачи. Продумайте все хорошенько. Да, предусмотреть всего не получится, но глобальных ошибок, влекущих переписывание всей программы, Вам удастся избежать.

1.11. Руководство по стилю программирования на AutoLISP

Условные обозначения:

 – ОБЯЗАТЕЛЬНО,  – ДОПУСТИМО,  – ЖЕЛАТЕЛЬНО,

 – НЕЖЕЛАТЕЛЬНО,  – ЗАПРЕЩЕНО.

Правила компоновки

 Использование глобальных данных и функций крайне нежелательно.

 Файл библиотеки должен начинаться с комментированного заголовка, имеющего следующую структуру:

```
;; filename.lsp
;; ФИО
;;
;; Имя библиотеки
;; Краткое описание назначения библиотеки


(prompt “/n filename.lsp /n”)
```

Далее следуют описания функций библиотеки

 Для структурирования текста программы обязательно использовать табуляцию. Величина табуляции должна составлять 4 пробела. Необходимо следить за тем чтобы в качестве табуляции использовался именно символ табуляции, а не 4 пробельных символа.

Комментарии

 Комментарии должны составлять примерно 50% текста. Желательно комментировать каждый логический блок операторов (условное ветвление программ, циклические конструкции).

 Обязательно комментировать объявление списков, функций и их аргументов, переменных.

 Предпочтительно использование комментария со сдвигом влево на величину табуляции следующим образом:

```
;Обмен значениями между переменными x и y
    (setq
        temp x
        x y
        y temp
    )
```

 Использование однострочного комментария заканчивающего строку допускается при объявлении переменных.

Идентификаторы

 Имена идентификаторов должны состоять из не более 6 латинских символов. Превышение этого предела требует большее количество памяти.

 Для имен идентификаторов обязательно использовать символы нижнего регистра. Для имен идентификаторов желательно использовать как можно более короткие англоязычные слова, например: count, row, area, top и т.п. Для составных имен использовать символ ‘_’.

Выражения и операторы

 Автоинкремент и автодекремент использовать в арифметических выражениях запрещается (1+ <число>).

Условные конструкции

 В конструкции `if` желательно структурирование даже в случае однооператорного `if`:

```
;однооператорный if:
(if (= a 4)
    (setq num_file "Table_1.txt")
    (setq num_file "Table_2.txt")
```

```
)

    ;многооператорный if:
(if (= num_file 1)
        (progn
                (setq f_name "Table_3.txt")
    ;создаем список
                (setq spisok (list a b c))
        )
;иначе
        (progn
                (setq f_name "Table_4.txt")
    ;создаем список
                (setq spisok (list b c a))
        )
)
```

Циклические конструкции

Для организации цикла с предусловием используется следующая конструкция:

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

Для организации пошагового цикла с предусловием используется следующая конструкция:

```
(repeat 4
   (setq a (+ a 10))
   (setq b (+ b 10))
)
```

Селективные конструкции

Когда необходимо предусмотреть большое количество разнообразных вариантов, то можно использовать конструкцию `cond` строго соблюдая пунктуацию, показанную ниже:

```
(cond
   ((= s "Y") 1)
```

```
   ((= s "y") 1)
   ((= s "N") 0)
   ((= s "n") 0)
   (t nil)
)
```

Функции

Имена функций могут состоять не менее чем из трех символов, которые должны быть латинскими буквами, и **при особой необходимости** арабскими цифрами.

Следует стараться избегать превышения предела в 6 символов для имен функций. Превышение этого предела требует большее количество памяти. Правда, с современной техникой это не принципиально.

Имена функций должны составляться из одного, двух или трех как можно более коротких англоязычных слов, каждое из которых записывается с большой буквы, например: `GetCount, SetTop` и т.п. (Для AutoLISP не имеет значения регистр, но такая запись дает возможность ориентироваться в назначении функций.)

Для реализации функции используется следующая конструкция:

```
;-----------------------------------------------
; назначение
;-----------------------------------------------
; входные параметры
;-------------------------
; выходные параметры
;-------------------------
 (defun имя (параметры / локальные переменные)
   тело функции
)
(prompt “имя, “)
```

В списке локальных переменных должны быть перечислены все переменные, используемые в данной функции.

Использовать в функциях аргументы по умолчанию нежелательно.

Желательно использовать передачу данных списком.

Передача данных должна производится как результат последнего действия.

Когда есть выбор, то предпочтительнее использовать функций стандартной библиотеки функций AutoLISP.

2. Создание примитивов. Работа с чертежом

2.1. Использование командной строки

Большинство команд Автокада могут быть выполнены из программы на AutoLISPe при помощи функции **command**:

```
(command fun par1 .. parn )
```

где fun - имя вызываемой команды; par1 ... parn - параметры вызываемой команды. Из программы на AutoLISP принципиально невозможно вызвать следующие команды: **DTEXT, SKETCH, PRINT, PLOT, SCRIPT,** а также команды, определенные пользователем при помощи (DEFUN C:).

Есть два особых вида выражений, которые могут быть аргументами функции **command**: **PAUSE** позволяет пользователю ввести соответствующий параметр вручную; "" (две кавычки) или отсутствие параметров вообще [(**command**)] равносильно прерыванию команды.

Пример:

Нарисуем из программы на AutoLISPe квадрат с левым нижним углом в точке (1,1) и стороной 21мм. Если бы мы пользовались только клавиатурой, то диалог выглядел бы следующим образом:

```
Command: PLINE
Specify start point: 1,1
Current line-width is 0.000
Specify next point or [Arc/Halfwidth/Length/Undo/Width]: @21,0
Specify next point or [Arc/Close/Halfwidth/Length/Undo/Width]: @0,21
Specify next point or [Arc/Close/Halfwidth/Length/Undo/Width]: @-21,0
Specify next point or [Arc/Close/Halfwidth/Length/Undo/Width]: Close
```

На AutoLISPe это будет выглядеть так:

```
(command "pline" "1,1" "@21,0" "@0,21" "@-21,0" "close" )
```

Все константы, являющиеся параметрами функции **command**, задаются как текстовые строки, даже если они являются числами или координатами точек.

Однако главное свойство функции **command** - возможность подстановки в качестве параметров результатов выполнения программ.

Любой параметр функции **command** можно заменить на имя переменной или выражение AutoLISPa. Данный параметр примет значение, равное значению переменной или результату вычисления выражения.

Ограничение: внутри функции **command** нельзя вызывать функции ввода данных

(GETREAL, GETSTRING и т.д.)

Как представить элементы в программе:

- Числа - использовать функции преобразования типов (itoa, rtos, atof, atoi).
- Строки - преобразование строк - strcat, substr.
- Точка - ввод заранее с помощью функции (getpoint) или формирование точки с помощью списка.
- Координаты точек являются списками из двух или трех вещественных чисел - координат по осям X, Y и Z соответственно. Таким образом, точка с координатами 10,10 может быть задана как текстовой строкой "10,10", так и списком: (**list** 10 10). Также можно использовать переменные и выражения AutoLISPa для указания координат.

В принципе имеющихся в AutoLISPe математических функций достаточно, чтобы выполнять геометрические построения.

2.2. Описание вызова команд AutoCAD из AutoLISP

Рисование линии

```
(command "_.line" p1 p2 ... pn "") ;; p1 и пр. - точки, через который проходит прямая. Для окончания ломаной - пустой ввод - "".
```

Рисование полилинии

```
(command "_.pline" точки или опции) ;; ломаная
(command "_.pline" p1 p2 p3 "") ;; ломаная
(command "_.pline" p1 "_a" "_s" p2 p3 "") ;; дуга
```

Кольцо

```
(command "_.donut" радиус_внутренний радиус_внешний точка_установки1
точка_установки2 ... до завершения - то есть до знака "")
;;Кольца с параметрами: внешний диаметр=5, внутренний=2, ставим в точки (0,0) и (23,0).
(command "_.donut" "2.0" "5.0" "0,0" "23,0" "")
```

Рисование прямоугольника

```
(command "_.rectangle" p1 p2)
```

Вывод штриховки - hatch

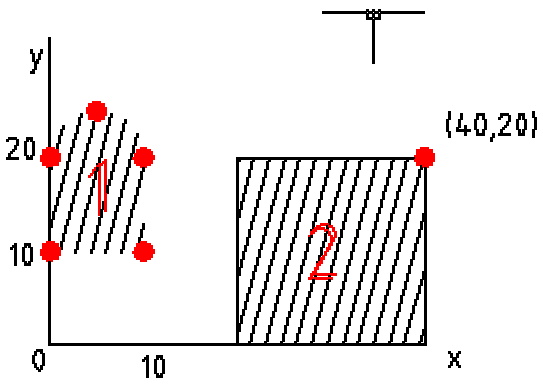


Рис 2. Штриховка.

(command "_.hatch" "название штриховки" "шаг" "угол" ... разные опции)

;; вывод штриховки для области, ограниченной точками (как полилиния) - случай № 1

(command "_.hatch" "ansi31" "0.5" "30" "" "" "10,10" "10,20" "_a" "_s" "5,25" "0,20" "_l" "0,10" "10,10" "" "")

;; вывод штриховки на область, ограниченную примитивом - прямоугольником - случай № 2

(command "_.hatch" "ansi31" "0.5" "30" "40,10" "")

bhatch

(command "_.bhatch" разные опции)

(command "_.bhatch" "p" "ansi31" "0.5" "30" "12,12" "") ;; вывод штриховки для области, указанной внутренней точкой

На рисунке 3 показан результат. Указана точка внутри квадрата. На втором примере (б) иллюстрируется указание сразу нескольких областей.

(command "_.bhatch" "p" "ansi31" "0.5" "30" "7,13" "21,16" "")

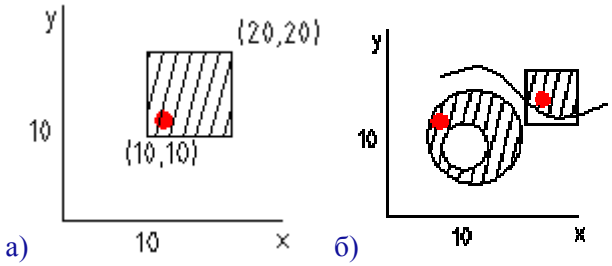


Рис 3. Штриховка. Область.

Изменение свойств объекта

;;Меняем цвет примитивов, находящихся внутри и пересекающих прямоугольник (10,10)-(200,200)

(command "_.change" "_box" "200,200" "10,10" "" "_p" "_c" "bylayer" "")

Работа с цветом. Изменение текущего рабочего цвета

Цвет может быть задан как названием цвета (yellow), так и цифрой (1-256). Так как мы имитируем ввод с клавиатуры, не забываем, что данные надо задавать в кавычках.

(command "_.color" "_bylayer") ;; задаем цвет, соответствующий основному цвету слоя

(command "_.color" (itoa col)) ;; цвет, рассчитанный по счетчику

Размер линейный

Простановка линейного размера между точками p1 и p2 расположение размерного текста в точке p3.

```
(command "_.dimlinear" p1 p2 p3)
```

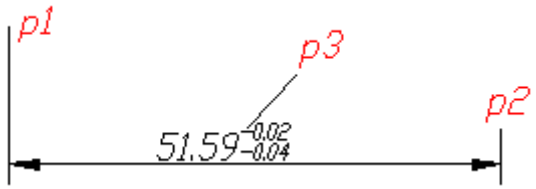


Рис 4. Размер

Радиус

```
(command "_.dimradius" p1 p3)
```

Точка на дуге - p1 определяет и расположение размерной линии по направлению к центру окружности. Случай а) - обозначение радиуса внутри окружности - в этом случае размерная линия до центра не доходит. Случай б) - размер ставится вне окружности. Тогда размер идет от центра окружности через точку p1 к указанной точке p2.

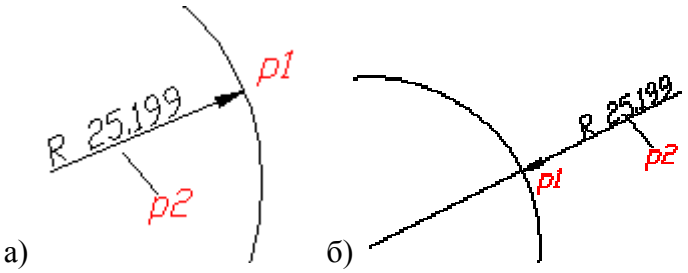


Рис 5. Радиус

Стиль размеров - управляет сохранением, удалением, активизацией стилей

Сохранение стиля - это сохранение опций системных переменных DIM* в единую структуру и присвоение имени. Поэтому для сохранения необходимо предварительно установить все переменные ряда *DIM в необходимое значение. Список переменных и их назначение – в приложении.

```
(command "_.dimstyle" "_r") ;;Активизация стиля:
(command "_.dimstyle" "_s" "rus_diam_sym" "_y") ;;Сохранение созданного стиля размеров
```

Опция "_y" означает, что такой стиль уже есть и его надо заменить. Если такого стиля нет, то можно записать сохранение так:

```
(command "_.dimstyle" "_s" "rus_diam_sym" "_y")
```

Разъединение группы примитивов, полилинии

```
(command "_.explode" p "");;P- точка на объекте или выбор объектов.
```

Сетка визуальная - управление отображением

```
(command "_.grid" "off")
```

Вставка блока

```
(command "_.insert" "имя" "точка_вставки" "масштаб по X" "по У" "угол разворота")
(command "_.insert" blockf "0,0" "1" "1" "0")  ; Вставка блока рамки
```

Управление слоями

```
;; Создание слоя "sizes", придание ему имени и цвета. Активизация созданного слоя.
(command "_.layer" "_n" "sizes" "_c" "3" "sizes" "s_" "sizes" "")
(command "_.layer" "_s" "0" "") ;;Переход на слой "0"
```

Стрелочка - выноска

```
(command "_leader" p1 p2 "" "" "n")
```

Этот набор параметров позволяет нарисовать стрелку с от p1 до p2 без подписей и прочей ерунды.

Смена типа линии

```
(command "_.linetype" "_s" "bylayer" "") ;; ставим тип линии в соответствии со слоем
(command "_.linetype" "_s" "dashdot" "") ;; ставим тип линии штрихпунктирный
```

Установка масштаба линй

```
(command "_.ltscale" "масштаб") ;;Иногда для верного отображения линий необходимо
корректировать масштаб отображения линий.
```

Установка привязок к элементам чертежа

```
(command "_.osnap" "привязки")
```

Совет: Использовать привязки необходимо только для организации ввода данных пользователем. В процессе рисования - привязки только мешают. Их лучше отключать.

```
 (command "_.osnap" "_none") ;;отключение привязок
(command "_.osnap" "int,nea,end,per,cen");; подключение нескольких привязок
```

Удаление информации из памяти чертежа

```
(command "_.purge" "опции")
;; Удалить все неиспользуемые слои
(command "_.purge" "la" "" "n")
```

Регенерировать изображение = обновить

```
(command "_.regen")
```

Создание стиля текста и активизация стиля

```
(command "_.style" "имя стиля" "шрифт" "высота" "коэф. ширины" "угол" "Backwards? <N>"
"Upside-down? <N>" "Vertical? <N>")
```

Если кроме имени стиля нет параметров, он становится текущим стилем текста.

Вывод текста

```
(command "_.text" "опции" "текст")
```

Вывод производится в определенную точку. Для текста эта точка является по умолчанию нижней левой точкой (BL). Изменить это можно с помощью опций команды из цикла Justify: Align/Fit/Center/Middle/Right/TL/TC/TR/ML/MC/MR/BL/BC/BR:

```
;; вывод текста с уплотнением, уместить текст между двумя точками. опция Fit. Обратите внимание на задание второй точки в относительных полярных координатах.
(command "_.text" "_fit" p1 "@17<0" txt)
(command "_.text" "c" "189,148" "0" txt) ;; вывод текста с указанием центральной точки
```

Рисование допуска

```
(command "_tolerance" p1)
```

Ставит обозначение допуска в точку. Параметры допуска - задаются пользователем в диалоговом окне.

Отмена предыдущего действия

```
(command "_.undo" "1")
```

Создание блока примитивов и сохранение их в отдельном файле

```
(command "_.wblock" "имя_файла" "имя_блока" "точка_вставки"  "выделение элементов" "")
```

Пример: создание блока с именем ff, и таким же именем файла-хранителя. Точка вставки - в начале координат. Выбор элементов - рамкой. Берем все элементы, попадающие в прямоугольник формата А4, + элементы, которые рамкой пересекаются.

```
(command "_.wblock" ff "" "0,0" "_c" "0,0" "210,297" "")
```

Увеличение, вид

```
(command "_.zoom" "опции")
 (command "_.zoom" "a") ;; Увеличение, при котором все элементы чертежа видны на рабочем поле
 (command "_.zoom" "w" p1 p2) ;; Увеличение окном, с указанием точек - границ окна
```

3. Работа с графической базой данных

Графическая база данных

Чуть ли не самой главной особенностью Автолиспа является то, что он позволяет осуществлять доступ к графической базе данных (ГБД) Автокада, многократно умножая возможности адаптации последнего к какому-либо типу задач. Попытаемся вкратце показать, как можно использовать возможности Автолиспа для работы непосредственно с объектами чертежа.

Любой создаваемый в Автокаде чертеж состоит из примитивов, геометрическое описание которых хранится в специальном формате (формате Автокада) в файле чертежа (расширение .dwg). При загрузке чертежа Автокад заполняет графическую базу данных: записывает системные настройки, создает список объектов и вносит в ГБД геометрическое описание этих объектов, присваивая каждому примитиву уникальное имя. В сеансе редактирования каждый примитив (отрезок, дуга, окружность и т.п.) имеет свое имя, по которому его распознает сам Автокад. Для оперирования примитивами необходимо в программе на Автолиспе сначала найти имя примитива в базе данных Автокада, чтобы потом изменять геометрические характеристики примитива. Попробуем извлечь это имя из ГБД при помощи Автолиспа.

3.1. Получение информации о примитивах

Нарисуйте отрезок. Для того, чтобы указать на примитив, используются различные способы. В частности, можно указать последний нарисованный элемент.

```
(entlast)
```

Введите с командной строки Автокада строку:

```
Command: (setq ENAME (entlast))
Автокад возвращает: <Entity name: 60000018>
```

Тем самым мы присвоили переменной **ENAME** имя последнего примитива (в данном случае отрезка). Имена примитивов в Автокаде - шестнадцатеричные величины; имя примитива может быть, например, таким: 60000A14. Используя это имя, вы можете при помощи функции **ENTGET** получить доступ к данным, связанным с примитивом:

```
(entget имя)
(setq EDATA (ENTGET ENAME))
```

Имя примитива - это *новый для нас тип данных Автолиспа*, и если функции Автолиспа **ENTGET** требуется имя примитива, то бесполезно указывать число 60000018 - надо передать переменную **ENAME**, в которой это имя хранится.

В результате выполнения команды вы получите малопонятное сообщение:

```
((-1 . <Имя примитива: 60000020>) (O . "LINE") (8 . "O") (10 1.0 2.0 0.0) (11 6.0 6.0 0.0))
```

DXF-коды

Все данные, описывающие примитив, - это список, состоящий из подсписков, в которых сгруппирована по функциональному назначению вся информация о примитиве, как геометрическая, так и общая; слой, цвет и т.п. Подсписки отличаются по специальным кодам формата **DXF** (Drawing exchange Format - формат обмена рисунками), позволяющим определить, какой тип данных хранится в подсписке. Каждый подсписок имеет две части. Первая - код **DXF**, вторая - данные. Целое число **0**, например, представляет собой код типа примитива. Код **8** говорит о том, что следующее за ним число - номер слоя. Код **10** - начальная точка примитива, код **11** - конечная и т.п. Отметим, что набор кодов **DXF** различен для примитивов разных типов. Однако сами коды относятся ко всем примитивам - имя примитива, например, всегда хранится в подсписке с кодом **DXF -1.**

Представим полученный список **EDATA** в более понятном виде:

```
( (-1 . <Имя примитива: 60000020>)
(0 . "LINE") - Тип примитива
(8 . "0") - Слой
(101.02.00.0) - Начальная точка
(11 6.0 6.0 0.0) ) - Конечная точка
```

Пользуясь кодами **DXF**, можно извлечь из списка **EDATA** любую информацию о примитиве. Такой доступ к рисунку более сложен, но позволяет изменять практически все свойства примитивов. При работе со списками данных о примитивах необходимо использовать кодировку кодов **DXF**.

3.2. Наборы примитивов

Автокад имеет стандартное средство работы с несколькими примитивами. Практически все команды редактирования работают не с отдельными примитивами, а с их **группой**. Из Автолиспа также можно работать с наборами примитивов: предоставлять пользователю возможность заносить примитивы в набор и затем их модифицировать. Набор формируется функцией **ssget**:

```
(ssget режим точка1 точка2 )
```

Необязательный аргумент **режим** - строка, которая указывает способ выбора примитива. Возможны следующие значения режима:

"P", в русском варианте **"Т"** - выбирается текущий набор;
"L", русское **"П"** - выбирается последний сформированный на чертеже примитив;
"W", русское **"Р" < точка1 > <точка2>** - выбираются примитивы, попавшие в рамку с углами в **< точке1 >** и **<точке2>**;
"C", русское **"С" <точка1> <точка2>** - выбираются примитивы, пересекаемые рамкой.

Кроме этих возможны еще две конструкции:

(ssget< точка >) - Здесь выбираются примитивы, проходящие через < точку >.

(ssget "x" < фильтр >) - Здесь **<фильтр>** - список, элементами которого являются одна или несколько точечных пар.

Каждая точечная пара строится по следующей форме:

(< код >.< значение >) - Здесь **<код>** - одно из принятых чисел, показывающих, по какому признаку выбираются примитивы, например;
0 - примитивы одного типа;
2 - примитивы, входящие в один блок (INSERT) ;
6 - по типу линии;
7 - по гарнитуре шрифта;
8 - по имени слоя;
62 - по цвету и др.
< значение > - значение признака.

Примеры:

(ssget) Выбирает по одному объекты чертежа
(ssget "T") Выбирает текущий набор
(ssget "П") Выбирает последний примитив
(ssget '(2 2)) Выбирает примитив, проходящий через точку (2,2)
(ssget "P" '(0 0) '(5 5)) Выбирает примитивы в рамке от (0,0) до (5,5)
(ssget "X" '((8 . "SLI"))) вернет набор, включающий все примитивы, находящиеся на слое, именованном как "SLI";
(ssget "X" '((0. "LINE") (62 .1))) вернет набор, состоящий из всех линий красного цвета (1 - номер красного цвета).

После того, как Вы определили необходимый Вам набор примитивов, можно использовать его во всех функциях модифицирования чертежа - **Copy, move, erase,** etc...

Пример:

;; Выполнение последовательности функций
(setq sdel (ssget "X" '((6 . "CENTER"))))
(command "ERASE" sdel) - приведет к стиранию всех примитивов, изображенных осевыми линиями.

Работа с наборами/списками примитивов

(sslength список, сформированный ssget) - определение длины списка

Выдает длину списка = количество элементов.

(ssname список_сформированный_ssget номер) - извлечение элемента списка

В результате мы извлекаем **имя** примитива из списка, и оно уже может обрабатываться оператором **entget** для извлечения данных.

Пример:

```
; Выводим на экран подряд все типы примитивов из списка aa
(setq i 0)
(while (< i (sslength aa))
   (setq aa0 (entget(ssname aa i))) ; данные о примитиве
   (setq aa0 (assoc 0 aa0))
   (print (nth 1 aa0))
   (setq i(+ i 1))
)
```

3.3. Работа с именами примитивов

Рассмотрим функции, позволяющие извлекать имена примитивов из ГБД:

(entnext имяпримитива)

При вызове без параметров возвращает имя первого неудаленного примитива в ГБД. При задании имяпримитива возвращает имя первого неудаленного примитива, следующего в ГБД за примитивом имяпримитива.

(entlast)

Возвращает имя последнего неудаленного примитива в ГБД.

Если требуется получить от пользователя необходимый примитив, используется команда

(entsel строка-подсказка)

одновременно указать не только объект, но и точку, в которой этот объект указан (в Автокаде подобная операция используется в объектной привязке, а также в командах РАЗОРВИ, ОБРЕЖЬ и УДЛИНИ, следует использовать функцию **ENTSEL**.

Выбирается единичный примитив, при этом выбор должен быть сделан указанием на примитив. Возвращается список, первый элемент которого - имя выбранного примитива, а второй - координаты точки, в которой он был указан. Строка-подсказка используется для запроса примитива; если она не указана, то выдается стандартный запрос "Выберите объекты:".

Список, возвращаемый функцией **ENTSEL**, может быть введён в ответ на стандартный запрос Автокада **"Select object:" ("Выберите объекты:")**, он воспринимается как выбор примитива путём указания в точку с координатами, указанными в списке.

Из Автолиспа можно непосредственно модифицировать данные о существующих примитивах. Однако для добавления нового примитива необходимо использовать команды отрисовки или редактирования Автокада. Это ограничение связано с желанием защитить ГБД от неграмотного программиста: неправильно пользуясь командами Автокада, вы не сможете испортить саму ГБД - самое большее, что вы испортите, это свой чертеж.

(entdel ename) - удаление примитивов из чертежа

Если вы повторите эту операцию, то примитив будет восстановлен. Этот способ позволяет удалять из ГБД невидимые (перекрытые другими) примитивы, что невозможно обычным выбором объектов (нельзя выбрать невидимый объект).

3.4. Программное изменение примитивов

Для модификации геометрических характеристик примитива надо уметь находить в DXF-списке данных примитива (наша переменная **EDATA**) подсписки, в которых хранится нужная информация. Выбор и изменение различных данных, относящихся к примитиву, осуществляются по коду DXF (это всегда целое число) с помощью функций **ASSOC** и **SUBST**:

(assoc элементсписка сложныйсписок) - извлечение элемента

Извлекает из **сложногосписка** элемент списка по ключу **элементсписка**. Если **элементсписка** не найден, **ASSOC** возвращает **nil**. Например:

Исходный список: **goods ((1 "car" "volvo")(2 "price" 80000))**

Тогда **(assoc 2 goods)** возвращает список **(2 "price" 80000).**

(subst новыйэлемент старыйэлемент список) - замена элемента

Возвращает копию исходного списка с заменой всех найденных подсписков, идентичных старомуэлементу, на новыйэлемент. Если вхождений не обнаружено, **SUBST** возвращает копию старого списка (не nil!):

(subst '(2 "price" 100000) '(2 "price" 80000) goods)
возвращает: ((1 "car" "volvo")(2 "price" 100000))

Используя эту технику, попробуем извлечь из списка **EDATA** имя примитива:

Команда: (assoc 0 **EDATA**)
Автолисп возвращает: (0 . "LINE")

В приведенном выше примере мы фактически сказали Автолиспу: "Возврати мне подсписок с DXF-кодом 0". Автолисп просмотрел DXF-список примитива, нашел подсписок с кодом 0 и возвратил его. Полученный по коду подсписок все еще содержит DXF-код, который необходимо убрать: он больше не понадобится. Для этой цели лучше всего использовать функцию CDR:

Команда: (cdr (assoc O **EDATA**))
Автолисп возвращает: "line"

Извлекая из DXF-списков нужную информацию, можно программно обрабатывать ее и затем, внеся изменения в DXF-список примитива при помощи функции SUBST, модифицировать ГБД при помощи функции **ENTMOD**:

(entmod список)

Эта функция преобразует список в формат, возвращаемый функцией **ENTGET**, и обновляет информацию в ГБД. Следует иметь в виду, что функция **ENTMOD** не всесильна. Во-первых, нельзя изменить тип примитива (если вы хотите сделать это, вам остаётся только удалить его с помощью функции **ENTDEL** и создать новый примитив с помощью функции **COMMAND**). Во-вторых, все объекты, на которые ссылается список данных, должны быть известны Автокаду к тому моменту, когда вызывается функция **ENTMOD** (гарнитура шрифта, типы линий, имена форм и блоков и пр.). Исключением из этого правила является слой - если определённого в списке слоя нет, будет создан новый. Целые значения автоматически преобразуются в значения с плавающей точкой

Когда обновляется не основной примитив, а "подпримитив" (вершина полилинии или атрибут блока), изображение на экране не обновляется (это невыгодно) - для обновления изображения следует использовать функцию **ENTUPD**:

(entupd список);; Обновляет изображение примитива на экране

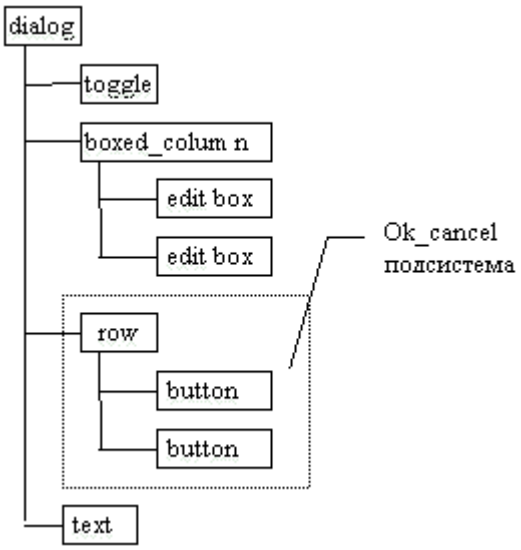
4. ДИАЛОГОВЫЕ ОКНА В AutoCAD'e

AutoCAD предусматривает возможность самостоятельного написания диалоговых окон, отличных от определенных в системе. Для этой цели был разработан специальный язык - DCL (Dialogue Control Language, или другими словами - язык управления диалоговыми окнами).

Диалоговые окна определяются файлами ASCII, написанными на языке управления диалогов (DCL). Элементы диалогового окна - кнопки, окна редактирования, известны как tile-элементы. Размер и функциональные возможности каждого элемента управляется атрибутами. Размер диалогового окна и размещение частей установлены автоматически с минимумом информации. VLISP имеет инструмент для просмотра диалоговых окон и функции для управления диалоговыми окнами из прикладных программ

4.1. Структура диалогового окна

Диалоговые окна определяются в текстовых файлах с соответствующим расширением *.DCL. Управления диалоговыми окнами осуществляется из лисп-программы. Диалоговое окно состоит из поля и элементов внутри этого поля. Файлы DCL организованы в древовидную структуру. Наверху дерева - (dialog) элемент, который определяет диалоговое окно непосредственно. Следующая диаграмма показывает структуру файла DCL:



Размещение, вид, и поведение элемента или подсистемы определяются атрибутами. Например, сам dialog, и наиболее предопределенные типы tile, имеют атрибут label, который определяет текст, связанный с tile. label диалогового окна определяет заголовок сверху диалогового окна, label кнопки определяет текст внутри кнопки, и так далее.

При выборе того или иного поля (в фокусе) диалоговое окно извещает об этом вызывающую программу, выработкой соответствующего кода.

Управление полем (видимость, момент появления) осуществляется его (поля) атрибутами. Поля могут заключаться в рамку, выстраиваться в ряд, колонку.

Визуализация/отображение диалоговых окон

В среде VisualLISP откройте/создайте файл в кодом DCL. Выберите `Tools -> Interface Tools -> Preview DCL`, чтобы отобразить диалоговое окно, определенное в текстовом редакторе. Т. к. можно отобразить несколько диалоговых окон, определенных в одном .dcl файле, VLISP запрашивает имя диалога, который Вы хотите

открыть:

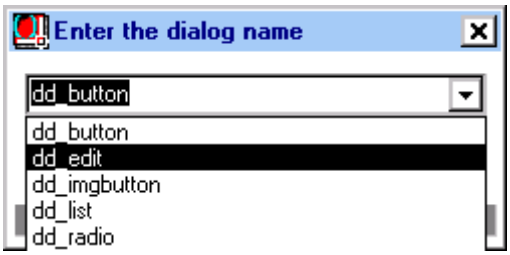


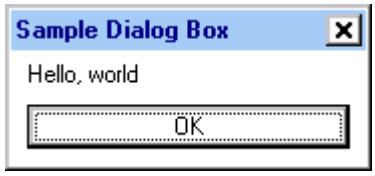
Рис 6. Диалоговое окно для выбора имени диалога

И после выбора Вы получаете на экране некий результат своего творчества.

Корректировка размера и размещения элементов диалоговых окон

Если ввести простейший код см. слева, то в окне AutoCAD при запуске увидим результат (справа).

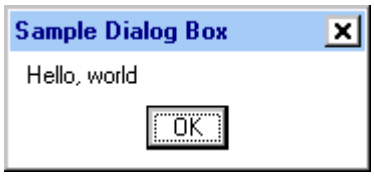
```
hello : dialog { label = "Sample Dialog Box";
 : text {  label = "Hello, world";}
 : button {  key = "accept";   label = "OK";
  is_default = true; }
}
```



Обратите внимание, что кнопка OK занимает почти полную ширину диалогового окна. Чтобы улучшить вид этого диалогового окна, отредактируйте файл DCL и добавьте два атрибута к элементу button. Чтобы ограничить ширину кнопки, добавьте атрибут fixed_width, и установите его = true. Кнопка сожмется и станет немного шире, чем текст внутри. Чтобы выровнять кнопку по центру, добавьте атрибут alignment. Элементы в столбце по умолчанию выравнивается по левому краю.

Корректировать размещение элементов следует лишь тогда, когда не подходит размещение по-умолчанию.

```
hello : dialog { label = "Sample Dialog Box";
 : text { label = "Hello, world";}
 : button {  key = "accept";   label = "OK";
  is_default = true;   fixed_width = true;
  alignment = centered; }
}
```



Размещение элементов в окне

При размещении элементов в диалоговом окне, нужно упорядочить их в строки и столбцы, основываясь на относительном размере каждого элемента. Ниже приведенный DCL определяет строку из трех элементов, которые располагаются выше одного большого элемента:

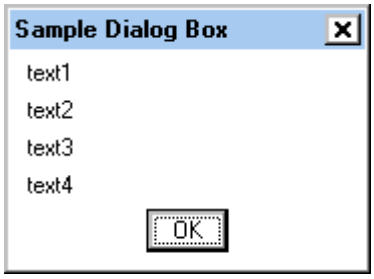
```
: column {
 : row {   : compact_tile { }   : compact_tile { } }
 : large_tile { }
```

```
}
```

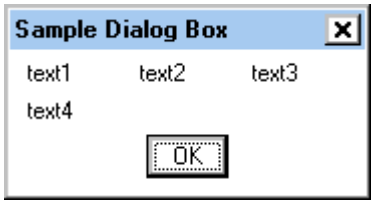
Расположение элементов. Примеры.

Ниже приведены примеры различного расположения элементов типа текст: код программы и результаты ее выполнения.

```
hello : dialog { label = "Sample Dialog Box";
 : text {label = "text1";} : text {label = "text2";}
 : text {label = "text3";} : text {label = "text4";}
 : button {  key = "accept";  label = "OK";
  is_default = true;   fixed_width = true;
  alignment = centered; }
}
```




```
hello : dialog {label = "Sample Dialog Box";
 : row { : text {label = "text1";} : text {label = "text2";}
  : text {label = "text3";} }
 : text {label = "text4";}
 : button {  key = "accept";  label = "OK";
  is_default = true;  fixed_width = true;  alignment =
centered; }
}
```



4.2. Описание элементов диалогового окна

Для создания различных элементов на поле диалога необходимо ознакомиться с элементами и их аттрибутами.

АКТИВНЫЕ ПОЛЯ

КЛАВИША button - Обычная клавиша Windows, кнопка

label: Строка в кавычках, определяет надпись, которая будет на клавише

is_default: Действие по умолчанию. Если значение FALSE или нет вообще кнопка не подсвечивается выбором. При значении TRUE (допускается иметь только одной кнопке окна) клавиша выделена и при нажатии клавиши <ENTER> происходит действие, сопряженное с этой кнопкой.

is_cancel: Тоже самое, что и is_default, но при нажатии клавиши <CANCEL>

Текстовое Поле edit_box

Поле для ввода или редактирования текста.

label: Текстовая строка в кавычках. Выводится слева от поля.

edit_width: Целое число - максимальное число символов, ограниченное рамкой, если значение не указано поле маштабируется по окну.

edit_limits: То же самое с точностью наоборот.

value: Строка в кавычках - исходное значение

Клавиша изображения - image_button

Поле с графическим изображением. Возвращаемое значение - координаты точки, в которой произошел выбор.

Color: Цвет фона (номер или символьное имя Автокада); dialog_line - цвет диалогового окна; dialog_forground - цвет символа, dialog_background - цвет фона, graphic_background - цвет фона графического экрана, graphic_forground - цвет символа (черным по белому)

Поле списка list_box

Поле с текстом в несколько строк. Обычно список. При количестве строк большем, чем высота окна с права автоматически появляется шкала прокрутки.

label: Строка в кавычках, выводимая над полем списка - типа заголовка.

multiple_select: Возможность множественного выбора с помощью <SHIFT+ левая кнопка мыши> при значении TRUE.

FALSE - выбор только одного пункта

list: Строка в кавычках - начальный выбор строки в поле списка.

value: строка из чисел (начиная с нуля), заключенных в кавычки, - первоначальный выбор пунктов списка. Соответственно multiple_select может содержать несколько чисел (TRUE) или одно (FALSE).

Раскрывающийся список popup_box

Поле с текстом в несколько строк, аналогичное list_box, но выглядит как поле со стрелкой справа, открывающей поле списка для выбора

label: Строка в кавычках, выводимая слева списка - значения по умолчанию нет

edit_widtht: Число - ширина поля в единицах ширины символа. При умолчании или нуле - выравнивается насколько возможно по окну.

list: Строка в кавычках - начальный выбор строки в поле списка.

value: чисел (начиная с нуля), заключенное в кавычки, - первоначальный выбор пункта в списке. Этот пункт будет выводиться в закрытом окне на экран.

Кнопка выбора rado_button

Группа или одна кнопочка ввиде кружка с текстом справа, объединенных в колонку или реже в ряд. Возможен выбор толко одной кнопки. При выборе другой кнопки из группы - первая отключится.

label: Строка в кавычках, выводимая справа от кнопки - значения по умолчанию нет

value: Строка в кавычках с атрибутом включения кнопки при открытии окна. "1" - кнопка включена "0" и все другие значения – кнопка выбора отключена.

Скользящая шкала slider

Перемещаемый пользователем движок, премещение соответствует изменению числового значения, передающегося в программу.

min_value; max_value: Целые числа, определяющие граничные значения.

small_increment: big_increment:

Целые числа, определяющие значения приращения. По умолчанию первое - сотая доля диапазона, второе - десятая. Атрибуты не обязательны

layout: Горизонтальная (по умолчанию) или вертикальная шкала.

label: Строка в кавычках, выводимая справа от кнопки - значения по умолчанию нет

value: Строка в кавычках (целое число) с текущим значением шкалы по умолчанию - min_value

Переключатель toggle

Переключатель "0" или "1" - небольшой прямоугольник с меткой "X", вкл./выкл.

label: Строка в кавычках, выводимая у кнопки - значения по умолчанию нет

value: Строка в кавычках (целое число "0" или "1") включен/ выключен. При значении "1" выводится "X".

АКТИВНЫЕ ГРУППЫ ПОЛЕЙ

Колонка column

Вертикальная колонка элементов

Колонка без рамки не имеет дополнительных атрибутов.

Колонка в рамке boxed_column

Вертикальная колонка элементов с рамкой по периметру

label: Строка в кавычках, выводимая в верхнем левом углу колонки, значение по умолчанию " ".

Ряд row

Горизонтальный ряд элементов

Колонка без рамки не имеет дополнительных атрибутов.

Ряд в рамке boxed_row

Горизонтальный ряд элементов с рамкой по периметру

label: Строка в кавычках, выводимая в верхнем левом углу ряда, значение по умолчанию " ".

Колонка выбора radio_column

Колонка, содержащая поля кнопок выбора, из которых можно выбрать только одну кнопку.

value: Строка в кавычках, содержащая значение ключа "key" выбранной текущей кнопки выбора.

Колонка выбора в рамке boxed_ radio_column

Колонка, содержащая поля кнопок выбора, из которых можно выбрать только одну кнопку с рамкой по периметру

label: Строка в кавычках, выводимая в верхнем левом углу колонки, значение по умолчанию " ".

value: Строка в кавычках, содержащая значение ключа "key" выбранной текущей кнопки выбора.

Ряд выбора radio_row

Ряд, содержащий поля кнопок выбора, из которых можно выбрать только одну кнопку.

value: Строка в кавычках, содержащая значение ключа "key" выбранной текущей кнопки выбора.

Ряд выбора в рамке boxed_radio_row

Ряд, содержащий поля кнопок выбора, из которых можно выбрать только одну кнопку с рамкой по периметру

label: Строка в кавычках, выводимая в верхнем левом углу ряда, значение по умолчанию " ".

value: Строка в кавычках, содержащая значение ключа "key" выбранной текущей кнопки выбора.

ДЕКОРАТИВНЫЕ И ИНФОРМАЦИОННЫЕ ПОЛЯ

Изображение image

Прямоугольник с векторным изображением: color: Цвет фона (номер или символьное имя Автокада), dialog_line - цвет диалогового окна, dialog_forground - цвет символа, dialog_background - цвет фона, graphic_background - цвет фона графического экрана, graphic_forground - цвет символа (черным по белому), acpect_ratio: Отношение ширины

изображения к высоте.

Надпись text

label: Строка в кавычках, выводимая в поле надписи.

value: Аналогична LABEL, но не влияет на компоновку полей. Если сообщение является неименяемым, следует определить атрибут LABEL без определения WIDTH и VALUE.

4.3. Функции автолиспа

Сначала DCL-файл надо найти и открыть. Обычно он располагается в том же каталоге где и вызывающая программа.

(load_dialod <имя_DCL_файла>)

Возвращаемое значение DCL_ID - целое

(unload_dialog DCL_ID)

Выгружает диалоговый файл. Она записывается после обработки всех полей и выхода из диалога.

В DCL-файле может быть сгруппировано несколько диалоговых окон. Следующая функция вызывает диалоговое окно (по имени) и начинает управление им.

(new_dialog <имя_окна> DCL_ID)

После вызова этой функции происходит установка всех полей, присвоение значений, создание изображений, списков. Возвращаемое значение t - при успешном открытии

(start_dialog).

4.4. Последовательность обработки при программировании

Создаете функцию. Внутри функции определяете идентификатор диалога **load_dialog**, проверяете, есть ли такой диалог **new_dialog**. Дальше - определяете действия, связанные с ключами **action_tile**, а дальше - диалог стартует **start_dialog**. С помощью оператора **cond** определяете выбранный пользователем вариант, и после этого диалог выгружаем из памяти **unload_dialog** и все.

Кнопки

Диалог

```
dd_button : dialog { label = "Тестирование кнопок";
fixed_height = true;
: button { key = "btn1"; value = "0"; fixed_height = true; alignment = center;
label = "Кнопка 1"; }
: button { key = "btn2"; value = "0"; fixed_height = true; alignment = center;
```

```
label = "Кнопка 2";
: row {cancel_button;}
}
```

Рис 7. Пример диалога для панели с кнопками

Программа

```
(defun dd_btn ( / dcl_id what_next)
; Загрузка диалога
(setq dcl_id (load_dialog "ot_tab.dcl"))
; Инициализация диалога
 (if (not (new_dialog "dd_button" dcl_id)) (exit) )
(setq what_next 8)
(action_tile "btn1" "(done_dialog 1)")
(action_tile "btn2" "(done_dialog 2)")
(setq what_next (start_dialog))
(cond ((= what_next 2) (alert "Запуск по кнопке 2!"))
((= what_next 1) (alert "Запуск по кнопке 1!"))
)
(unload_dialog dcl_id)
)
```

Переключатели

Следующий пример отличается тем, что используется цикл. Для того, чтобы дать возможность пользователю кликать/щелкать переключателями сколь угодно раз до нажатия ОК, необходимо процесс обработки данных заключить в цикл. Пока не будет what_next=1, цикл будет крутиться. Функция ok_tab формирует данные для выхода.

Диалог

```
dd_radio : dialog {label = "Тестирование переключателей";
fixed_height = true;
: boxed_column { label = "Выберите вариант";
: radio_column {
: radio_button {key = "radio1"; label = "Radio1"; value = 1;}
: radio_button {key = "radio2"; label = "Radio2"; }
: radio_button {key = "radio3"; label = "Radio3"; }}
}
: row { ok_button; cancel_button; }
}
```

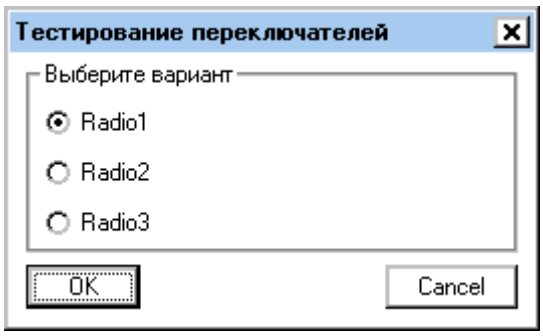


Рис 8. Пример диалога для панели с переключателями

Программа

```
(defun dd_radio ( / ret_value1 dcl_id what_next on_rad)
; функция, вызываемая по OK
(defun ok_tab ( / )
; формирование списка данных на выход
(setq ret_value1 (list "radio" on_rad))
)
; загрузка диалога
(set_tile "radio1" "1")
(setq on_rad "radio1")
(setq dcl_id (load_dialog "ot_tab.dcl"))
; инициализация диалога
(if (not (new_dialog "dd_radio" dcl_id))
; Exit if this doesn't work
(exit)
)
(setq what_next 8)
(while (< 2 what_next)
(action_tile "radio1" "(setq on_rad $key)")
(action_tile "radio2" "(setq on_rad $key)")
(action_tile "radio3" "(setq on_rad $key)")
(action_tile "accept" "(done_dialog 1) (ok_tab)")
(setq what_next (start_dialog))
)
(unload_dialog dcl_id) ; Unload the DCL file
(setq relst ret_value1)
)
```

Поля

В примере с полями особенностью является задание значений полей заранее - в тексте программы.

Диалог

```
dd_edit : dialog {label = "Тестирование флагов";
fixed_height = true;
: edit_box { key = "edit1"; value = "0"; fixed_height = true; alignment = center;
label = "edit1"; }
: edit_box { key = "edit2"; value = "0"; fixed_height = true; alignment = center;
label = "edit2"; }
: row {ok_button; cancel_button;}
}
```

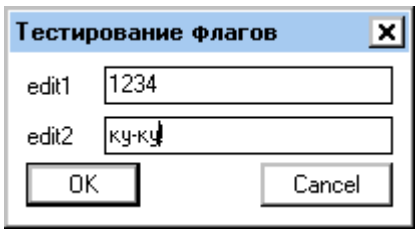


Рис 9. Пример диалога с полями для заполнения

Программа

```
(defun dd_edit ( / ret_value1 dcl_id what_next on_ed2 on_ed1)
; функция, вызываемая по OK
 (defun ok_tab ( / )
; формирование списка данных на выход
 (setq ret_value1 (list (list "ed1" on_ed1) (list "ed2" on_ed2)))
)
; загрузка диалога
(setq dcl_id (load_dialog "ot_tab.dcl"))
; инициализация диалога
(if (not (new_dialog "dd_edit" dcl_id))
; Exit if this doesn't work
(exit)
)
(set_tile "edit2" "ку-ку!")
(set_tile "edit1" "1234")
(setq on_ed1 "1234" on_ed2 "ку-ку!")
(setq what_next 8)
(while (< 2 what_next)
(action_tile "edit1" "(setq on_ed1 $value)")
(action_tile "edit2" "(setq on_ed2 $value)")
(action_tile "accept" "(done_dialog 1) (ok_tab)")
(setq what_next (start_dialog))
)
(unload_dialog dcl_id) ; Unload the DCL file
(setq relst ret_value1)
)
```

Флаги

Рассматривая пример с флагами, необходимо отметить, что в данном случае мы инициализируем заранее значения. Функция on_type_os изменяет элементы окна на заданные в параметрах функции. В зависимости от параметров, с помощью оператора set_tile устанавливаем значения флагов.

Диалог

```
// Описание диалога для панели с флагами
dd_toggle : dialog {label = "Тестирование флагов";
fixed_height = true;
: toggle { key = "tog1"; value = "0"; fixed_height = true; alignment = center; label = "Флаг1"; }
: toggle { key = "tog2"; value = "0"; fixed_height = true; alignment = center; label = "Флаг2"; }
: row  { ok_button; cancel_button; }
}
```

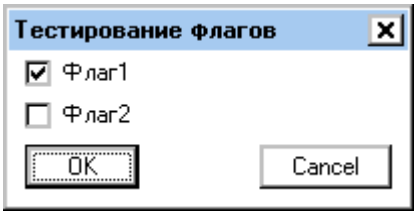


Рис 10. Пример диалога для панели с флагами

Программа

```
;-----------------------------------------
; функция вызова окна диалога
(defun dd_toggle (type_os / ret_value1 dcl_id what_next on_tog1 on_tog2)
; функция, вызываемая по OK
(defun ok_tab ( / ) ; формирование списка данных на выход
        (setq ret_value1 (list (list "tog1" on_tog1) (list "tog2" on_tog2)))
)
; выбор включенных-выключенных ключей в зависимости от введенных параметров
(defun on_type_os( / )
(cond
((= type_os "A") (setq on_tog1 "1" on_tog2 "0") (set_tile "tog1" "1") (set_tile "tog2" "0"))
((= type_os "B") (setq on_tog1 "1" on_tog2 "1") (set_tile "tog1" "1") (set_tile "tog2" "1"))
(t)
)
)
(on_type_os)
(setq dcl_id (load_dialog "ot_tab.dcl"))
; инициализация диалога
(if (not (new_dialog "dd_toggle" dcl_id))
(exit)
```

```
)
; Исходная инициализация ключей в зависимости от введенных параметров
(on_type_os)
(setq what_next 8)
(while (< 2 what_next)
(action_tile "accept" "(done_dialog 1) (ok_tab)")
(action_tile "tog1" "(setq on_tog1 $value)")
(action_tile "tog2" "(setq on_tog2 $value)")
(setq what_next (start_dialog))
)
(unload_dialog dcl_id) ; Unload the DCL file
(setq relst ret_value1)
)
```

Кнопки с изображением

В примере с кнопками-иллюстрациями ситуация еще интереснее. Необходимо создать слайды, которые будут использоваться на кнопке. Слайды создаются так: Рисуете на поле чертежа что-нибудь. Вводите команду mslide - вводите имя файла-слайда. Все! Слайд готов. Для отображения в окне ACAD необходимо ввести команду vslide. Тогда слайд временно выведется на рабочее поле.

Размеры изображения задаются в DCL-файле, и еще изображение надо инициализировать в лисп-файле - см. пример. А обработка производится также, кк и обработка кнопок.

Диалог

```
dd_imgbutton : dialog {label = "Тестирование кнопок с изображением";
: boxed_row {label ="Sizes"; fixed_height = true;
:image_button{color = 0; height = 1.5; width = 6; aspect_ratio = 0.36; fixed_height = true;
fixed_width = true; alignment = center; key = "btn1";}
:image_button{color = 0; height = 1.5; width = 6; aspect_ratio = 0.36; fixed_height = true;
fixed_width = true; alignment = center; key = "btn2";}
}
: row {cancel_button;}
}
```

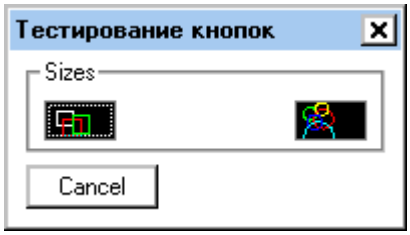


Рис 11. Пример диалога для панели с картинками-кнопками

Программа

```
(defun dd_imgbtn ( / dcl_id what_next)
```

```
; загрузка диалога
(setq dcl_id (load_dialog "ot_tab.dcl"))
; инициализация диалога
(if (not (new_dialog "dd_imgbutton" dcl_id))
(exit)
)
(start_image "btn1")
(slide_image 0 0 (dimx_tile "btn1") (dimy_tile "btn1") "btn1" )
(end_image)
(start_image "btn2")
(slide_image 0 0 (dimx_tile "btn2") (dimy_tile "btn2") "btn2" )
(end_image)
(setq what_next 8)
(action_tile "btn1" "(done_dialog 1)")
(action_tile "btn2" "(done_dialog 2)")
(setq what_next (start_dialog)) (cond
((= what_next 2) (alert "Запуск по кнопке 2!"))
((= what_next 1) (alert "Запуск по кнопке 1!"))
)
(unload_dialog dcl_id)
)
```

Списки

В работе со списками основная сложность - составление самого списка. Он составляется как символьная строка, через знак "\n" - перевод каретки.

Диалог

```
dd_list : dialog {
label = "Тестирование списков";
:popup_list{label = "1s&t: "; key = "lst1";
list = "None \nDatum Triangle Filled \nDatum Triangle \nIntegral \nUser Arrow...";
edit_width = 20;
}
:popup_list{label = "2s&t: "; key = "lst2"; list = "None \nDatum Triangle Filled \nDatum Triangle
\nIntegral \nUser Arrow..."; edit_width = 20;}
: row {ok_button;cancel_button;}
}
```

Рис 12. Пример диалога со списками

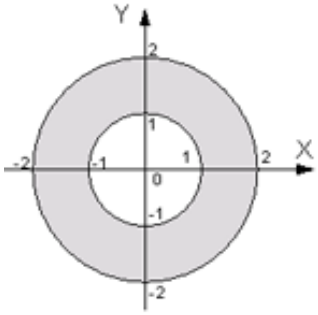
Программа

```
(defun dd_list ( / ret_value1 dcl_id what_next on_list1 on_list2)
; функция, вызываемая по OK
(defun ok_tab ( / ); формирование списка данных на выход
(setq ret_value1 (list (list "list1" on_list1) (list "list2" on_list2)))
)
; загрузка диалога
(setq on_list1 0 on_list2 0)
(setq dcl_id (load_dialog "ot_tab.dcl"))
; инициализация диалога
(if (not (new_dialog "dd_list" dcl_id))
; Exit if this doesn't work
(exit)
)
(setq what_next 8)
(while (< 2 what_next)
(action_tile "lst1" "(setq on_list1 $value)")
(action_tile "lst2" "(setq on_list2 $value)")
(action_tile "accept" "(done_dialog 1) (ok_tab)")
(setq what_next (start_dialog))
)
(unload_dialog dcl_id) ; Unload the DCL file
(setq relst ret_value1)
)
```

Рассмотренные в этой главе основные примеры должны помочь при обработке создаваемых Вами диалогов.

ПРИЛОЖЕНИЯ

Приложение А. Задание «Попадание точки в область – 1»



Необходимо получить карточку с заданием - область, с которой предстоит работа. Само задание простое - определение попадания точки в область, заданную рисунком.

Запрограммировать отображение заданной области на экране. Организовать ввод данных о точке в программу. Осуществить проверку попадания точки в заданную область. Показать введенную пользователем точку на экране. Вывести сообщение о попадании или не попадании точки в область.

Приложение Б. Задание «Попадание точки в область – 2»

Изменить программу таким образом, чтобы данные о размерах области вводились из файла area.txt. Запрограммировать отображение заданной области на экране. Организовать два варианта ввода:

- ввод данных о точке в программу с клавиатуры/мыши.
- ввод списка точек из файла

Осуществить проверку попадания точек в заданную область. Показать введенную/ые пользователем точки на экране. Вывести данные о попадании или не попадании точки в область:

- если одна точка - то alert
- если список из файла, то в файл result.txt записываются данные о времени и дате создания, авторе программы, данные об области, и по строкам: точка и результат (попадает в область или не попадает в область).

Приложение В. Для задания 2. Работа с файлами

Для осуществления чтения из файла, необходимо, чтобы файл-источник информации находился в текущем каталоге. Это означает, что перед запуском своей программы необходимо удостовериться, сохранен ли файл чертежа.

Для открытия файла необходимо его найти. Этим занимается функция **(Findfile "filename")**. Она возвращает путь в к файлу, который уже можно использовать в программе для открытия файла. Если nil - отсутствие упомянутого файла. Поэтому перед открытием файла для чтения необходимо проверять его наличие.

```
(if (/= (findfile "filename") nil)
  (progn
    (setq f (open (findfile "filename") "r"))
…
    (close f)
  )
;(вывод надписи, что нет такого файла)
)
```

Для того, чтобы осуществить чтение из файла, необходимо знать его структуру. Если данные в файле расположены столбиком (по одному в строке), то можно обойтись простым оператором read-line. С выполнением каждого оператора **read-line**, производится переход на следующую строку в файле.

```
; чтение одной строки из файла, открытого по дескриптору файла f.
(read-line f)
```

В результате мы получим некое строковое выражение. Его можно обработать с помощью операторов atoi или atof для получения целого или вещественного значений соответственно.

```
(setq a (atof(read-line f)))  ; извлекаем вещественное значение
```

Если строка сложная - то сначала придется разделить ее на составляющие части. Разделить строку на подстроки можно с помощью оператора substr:

```
(setq a(read-line f))
(setq b(substr a 1 7)) ; выделение первых семи позиций строки
```

В таблице представлена обработка строк файла - получение целого, вещественного, строкового значений, выделение подстроки и ее обработка. Слева вы видите пример файла с некоторой информацией, а справа - четыре варианта обработки строк файла и результаты обработки*.

Файл, открытый по дескриптору f													Обработка строк файла - код программы и получаемый результат			
номера позиций													вещественное	целое	подстрока	обработка подстроки
	1	2	3	4	5	6	7	8	9	10	11	12	13	(atof(read-line f))	(atoi(read-line f))	(substr (read-line f) 7)	(atof (substr (read-line f) 7))
1			5	5	.	4	5	6						55.456	55	"5 6 _ _ _ _ _"	56.0
2	7													7.0	7	"_ _ _ _ _ _ _"	0.0
3		3	.	7					8	.	9	8		3.7	3	"_ _ 8 . 9 8 _"	8.98
4		а	л	ы	е		к	р	ы	л	ь	я		0.0	0	"к р ы л ь я _"	0.0
5										6	9			69.0	69	"_ _ _ 6 9 _ _"	69.0

Рис 13. Файл с информацией. Обработка.

* Для наглядности представления, знак пробела в подстроках помечен как "_".

Запись в файл

Для создания файла необходимо воспользоваться оператором **(open "filename" "w").** Если **filename** - просто имя файла с расширением, то файл создастся в текущем каталоге. Если это путь вида:

```
Диск:\\каталог\\каталог\\имя.файла
```

то файл создастся в указанном каталоге.

Для записи в текстовый файл наиболее удобно использовать оператор **write-line**. Он позволяет записывать информацию в файл построчно.

```
(write-line "Test" f)
```

Для вывода в файл все данные должны быть преобразованы в строки. Целые - **itoa**, вещественные - **rtos**, объединение строк - strcat. В приведенном ниже примере производится формирование строки для вывода информации о точке.

```
;; функция для преобразование точки в строку
;; (использованы координаты X и У)
(defun pt_to_str(pt)
  (strcat "(" (rtos (nth 0 pt)) "," (rtos (nth 0 pt)) ")" )
)
```

Таким образом, в результате этого кода вы получите описание координат точки в виде "(x,y)".

При запуске: (pt_to_str (list 3.45 80))

Получим: "(3.45,80)"

Преобразуя и соединяя необходимые строки, можно организовать вывод информации в файл.

Работа с файлом, количество строк которого неизвестно

Чтение строк файла можно производить в цикле. Наиболее подходит для этого цикл while.

```
(while (условие)
  (setq a (read-line f))
  ; обработка данных
)
```

Условием цикла необходимо задать **(/= a nil)** - неравенство прочитанной строки концу файла. Кроме того, для проверки и первой строки, необходимо произвести чтение первой строки до проверки, а внутри цикла чтение сделать последней операцией:

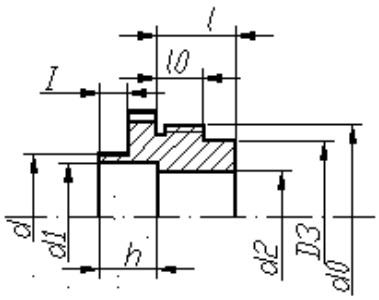
```
(setq a (read-line f))
; цикл с проверкой конца файла
(while (/=a nil)
  ; обработка данных
  (setq a (read-line f))
```

)

Тогда такая конструкция будет стабильно работать при неизвестном количестве строк в обрабатываемом файле.

Приложение Г. Задание «Параметрический чертеж оправы»

Написать программу, отображающую заданную оправу на поле чертежа в указанной точке. Размеры оправы должны вводиться в программу пользователем.

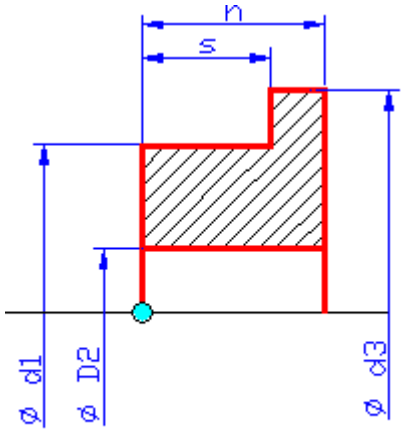


Программа должна состоять как минимум из двух подпрограмм: программы ввода данных и программы параметрического рисования оправы. Оправу отображать без масштабирования

Приложение Д. Пример выполнения задания: «Параметрический чертеж оправы»

```
;-------------------------------------------------------------
; Параметрический чертеж. Блок ввода данных
;-------------------------------------------------------------
;основная функция:
(defun Try_param ( / p1 d1 d2 d3 s h )
; инициализация всех данных для рисования
  (setq p1 (Getpoint "Где нарисовать?")
    d1 10.0       D2 5.0       d3 15.0
    s 6.0       h 12.0
  )
; запуск параметрического чертежа
  (DrawDetail p1 d1 d2 d3 s h)
)
(prompt "Try_param, ")
```

Пример задания:




```
;-------------------------------------------------------------------------
; Параметрический чертеж. Блок запуска
;-------------------------------------------------------------------------
;основная функция:
(defun DrawDetail (p1 d1 d2 d3 s h / p2 p3 p4 p5 p6 p7 x y det)
; обходим деталь по часовой стрелке
  (setq
    x (nth 0 p1)                          y (nth 1 p1)
    p2 (list x (+ y (/ d2 2.0)))          p3 (list x (+ y (/ d1 2.0)))
    p4 (list (+ x s) (+ y (/ d1 2.0)))    p5 (list (+ x s) (+ y (/ d3 2.0)))
    p6 (list (+ x h) (+ y (/ d3 2.0)))    p7 (list (+ x h) (+ y (/ d2 2.0)))
  )
; отключение привязок
  (command "_.osnap" "_none")
; рисование
  (command "color" 4) ; смена цвета (для выделения толстых линий)
  (command "_.pline" p2 "w" "0" "" p3       p4 p5 p6 p7 "c")
```

```
; забегая вперед, это - сохранение нарисованного примитива
  (setq det (entlast))
; отражение нарисованной детали
  (command "_.mirror" det "" p1 "@1,0" "N")
; рисование
  (command "_.pline" p2 (list x (- y (/ d2 2.0))) "")
  (command "_.pline" p7 (list (+ x h) (- y (/ d2 2.0))) "")
; установка белого цвета
  (command "color" 7)

; ось
  (command "_.line" (list (- x 10) y) (list (+ x h 10) y)"")

; штриховка
  (command "_.bhatch" "p" "ansi31"
    (/ (+ d3 h) 70.0) ; расчет относительного масштаба штриховки
    "0" ; угол наклона
    (list (+ x (/ h 10.0)) (+ y (/ d2 2.0) (/ (- d1 d2) 4.0))) ; точки внутри шртихуемой области
    (list (+ x (/ h 10.0)) (- y (/ d2 2.0) (/ (- d1 d2) 4.0))) ; точки внутри шртихуемой области
    ""
  )
)
(prompt "DrawDetail ")
```

Следующий шаг в совершенствовании программы - вставка проверки данных на допустимость/совместимость, которая будет обеспечивать стабильность работы программы.

Приложение Е. Задание «Доступ к примитивам»

1. Написать программу, запрашивающую у пользователя ширину и меняющую ширину всех красных полилиний на заданную.

Считать все исходные полилинии одной толщины. Красные полилинии - цвет 1.

2. *Написать программу, преобразующую полилинию в последовательность дуг и отрезков, и подсчитывающую суммарную длину отрезков.

Обработка вершин полилинии - записать их в список и вычислить расстояние между вершинами.

3. Написать программу, переносящую все объекты чертежа на слой 0.

4. Написать программу, изменяющую цвет всех размеров на зеленый.

Создать набор примитивов и заменить исходный цвет на зеленый.

5. Написать программу, изменяющую стиль всех размеров на созданный (информационный - со звездочкой).

Создать стиль размеров "Info", создать набор примитивов и заменить стиль на "Info".

6. Написать программу, удаляющую с чертежа все штрих-пунктирные (тип DASHDOT) линии.

Создать набор примитивов нарисованных этим типом линии, и удалить с чертежа этот набор.

7. Написать программу, создающую слой "Thick" и переносящую все элементы голубого

цвета на слой "Thick".
Создать слой "Thick", создать набор примитивов голубого цвета, заменить в каждом примитиве набора слой на новый.

8. Написать программу, подсчитывающую количество желтых отрезков на чертеже.
Создать набор примитивов, которые одновременно имеют цвет 2 и являются примитивами line. Посчитать количество примитивов в наборе.

9. * Написать программу, подсчитывающую окружности, центры которых находятся на одной оси, указанной пользователем.
Создать набор окружностей. Проверить, центры каких из них расположены на указанной пользователем оси, пересчитать их.

10. Написать программу, записывающую информацию о координатах центров всех окружностей в список.
Создать набор окружностей. Проити по набору, записывая в список все данные о центрах окружностей в список.

11. Написать программу, которая все примитивы типа TEXT переносит на слой "Text" .
Создать слой "Text", создать набор размеров, заменить в каждом примитиве набора слой на новый.

12. * Написать программу, автоматически проставляющую осевые линии на всех окружностях чертежа.
Создать набор окружностей. Пройти по списку, рассчитывая координаты для осевых линий и нанося их на чертеж.

13. Написать программу, запрашивающую у пользователя цвет и меняющую цвет выбранных примитивов на заданный.
Запросить примитив у пользователя. Заменить цвет в свойствах примитива на заданный.

14. Написать программу, которая преобразует примитивы типа TEXT к одинаковой величине шрифта - введенной пользователем.
Ввести в программу величину текста. Заменить в наборе примитивов типа текст атрибут величины шрифта на заданный.

15. * Написать программу, записывающую в список периметры всех прямоугольников чертежа.
Создать набор полилиний. Получить информацию о вершинах полилиний. Пределить, какие полилинии - прямоугольники. Вычислить периметры и записать в список.

16. Написать программу, записывающую в список радиусы всех дуг и окружностей чертежа.
Создать набор дуг и окружностей. Пройдя по набору примитивов, записать радиусы в список.

17. * Написать программу, формирующую список данных об оптической системе, отображенной на чертеже (пользователь указывает ось).
Считать, что поверхности отображены дугами. (Полилинии не учитывать). Создать набор дуг. Проверить, центры каких из них расположены на указанной пользователем оси, записать в списки центры и радиусы дуг. Произвести "перестановку".

18. Написать программу, создающую слой "Size" и переносящую все размеры на слой "Size".

Создать слой "Size", создать набор размеров, заменить в каждом примитиве набора слой на новый.

19. * Написать программу, преобразующую полилинию в последовательность дуг и отрезков, и записывающую в список параметры дуг (list (list центры...) (list радиусы)).

Приложение Ж. Пример выполнения задания «Доступ к примитивам»

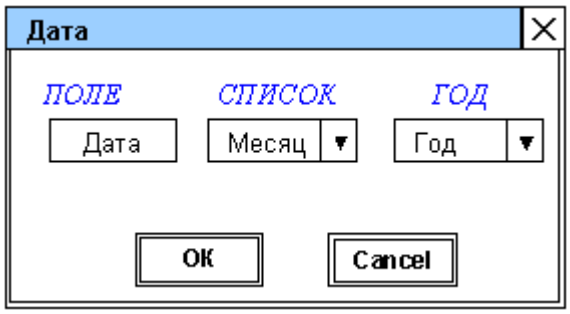
Программа выделяет все элементы - дуги на слое "1", создает слой "2", копирует все выделенные элементы на этот слой, со смещением @5,5,0. Также программа меняет цвет выделенных примитивов, и, извлекая данные о дугах, выводит эту информацию в текстовое поле AutoCAD.

```
;-------------------------------------------------------------------------------
; программа создания/включения слоя
;-------------------------------------------------------------------------------

(defun LayOn(laynam / )

(if (not (tblsearch "LAYER" laynam))
(command "_.layer" "_n" laynam "ON" laynam "")
(command "_.layer" "ON" laynam "")
)
)

 ;-------------------------------------------------------------------------------
; ПРОГРАММА
;-------------------------------------------------------------------------------

(defun $get_arcs ( / i aa aa2 el_t rad cen st_a en_a)

; программа создания слоя

(LayOn "2")

; подбор примитивов - дуг

(setq aa (ssget "X" '((-4 . "<AND")(0 . "ARC")(8 . "1")(-4 . "AND>")) ))

; обработка окружностей - копирование на другой слой

(setq i 0)
(if aa
(while (< i (sslength aa))
(setq aa5 (ssname aa i)) ; имя примитива

; копирование каждого элемента со сдвигом (5,5)
; можно было бы скопировать списком, но тогда
; сложнее менять цвет примитивов, и принадлежность слою

(command "_.copy" aa5 "" "0,0" "5,5")

; сохранение последнего элемента

(setq aa2 (entlast))

; взятие данных о нем

(setq el_t (entget aa2))

; замена информации о слое на новую
```

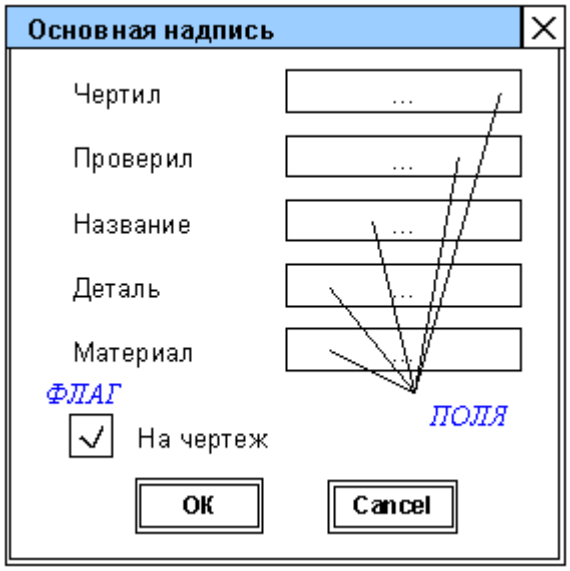
```
(setq el_t (subst (cons 8 "2") (assoc 8 el_t) el_t))
; замена/добавление информации о цвете
(if (assoc 62 el_t)
; если цвет указан - замена
(setq el_t (subst (cons 62 33) (assoc 62 el_t) el_t))
; если цвет не указан - добавление
(setq el_t (append el_t (list (cons 62 33))))
)
; обновить базу
(entmod el_t)
; извлечение информации о параметрах дуги
(setq
rad (cdr (assoc 40 el_t))  cen (cadr (assoc 10 el_t))
st_a(cdr (assoc 50 el_t))  en_a(cdr (assoc 51 el_t))
)
; вывод информации
(Prompt "\nРадиус:") (print rad)
(Prompt "\nЦентр:") (print cen)
(Prompt "\nСтартовый угол дуги:") (print st_a)
(Prompt "\nКонечный угол дуги:") (print en_a)

(setq i(+ i 1))
)
(alert "Извините, но на чертеже нет элементов с такими параметрами")
)
)
(prompt "$get_arcs ")
```

Приложение 3. Создание и программирование диалоговых окон. Задания

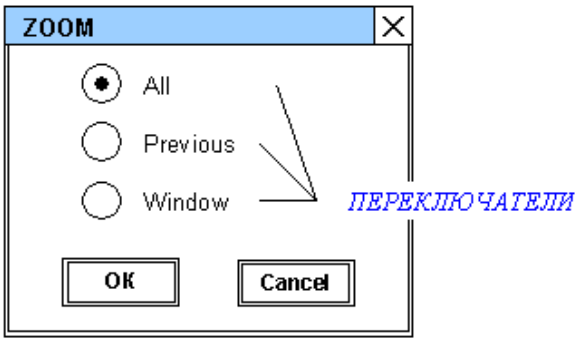
1. Организация диалога "Дата" - ввод данных о дате в диалоге. По ОК - результат печатается в командной строке.



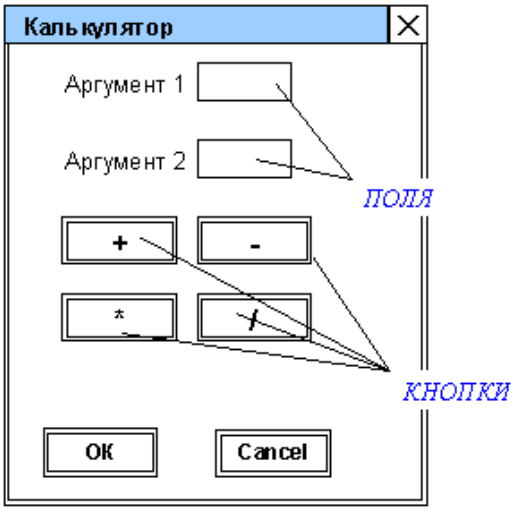
2. Заполнение основной надписи. Если в диалоге флаг установлен, то значения выводить в таблицу. Если не установлен, то список значений выдавать в командной строке.



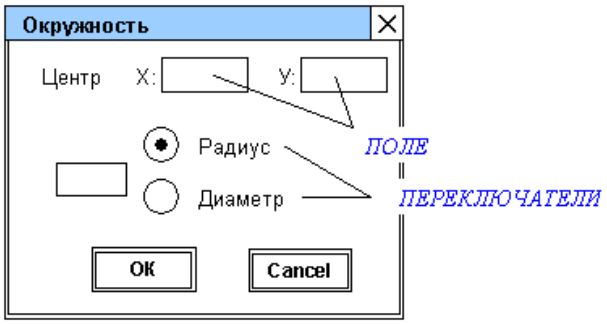
3. Запуск команды с вариантами выбора.



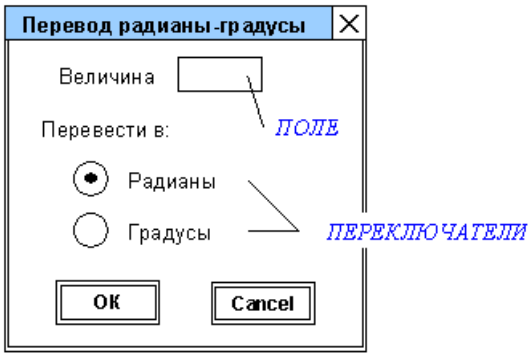
4. Реализация калькулятора в среде. Результат должен выводиться в командной строке.



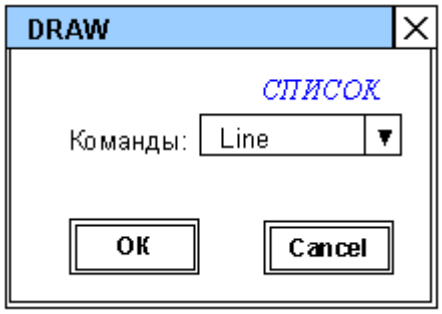
5. Запуск команды с вариантами выбора.



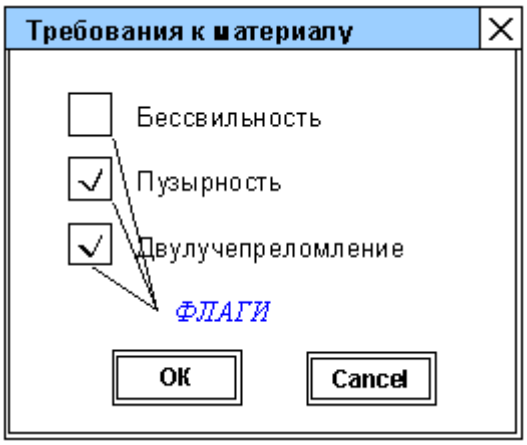
6. В диалоге - ввод данных, в командной строке - ответ.



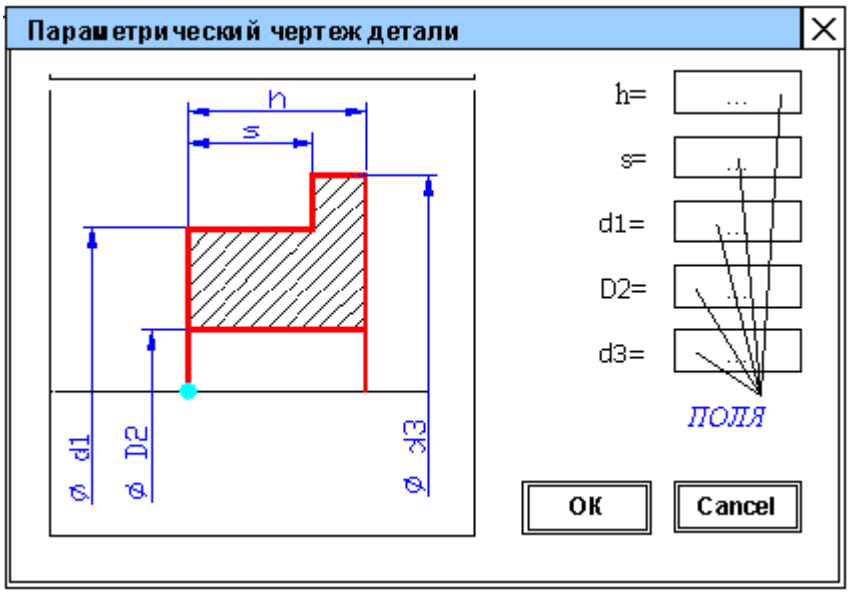
7. Запуск любой команды рисования, выбранной пользователем из списка.



8. Формирование набора необходимых параметров таблицы. В командной строке выдается список параметров стекла, указанных пользователем.



9-Х Задания, связанные с параметрическим чертежом. Каждый выполняет диалог для своей задачи, выполненной на этапе параметризации. Необходимо обеспечить наличие данных по-умолчанию, которые будет можно менять при работе с диалогом. По кнопке ОК производится запуск программы рисования оправы.



Литература

1. Гладков С.А. Программирование на языке Автолисп в системе САПР Автокад. - М.: “ДИАЛОГ-МИФИ”,1991.- 96с
2. Бугрименко Г.А. Автолисп - язык графического программирования в системе AutoCAD. - М.: Машиностроение, 1992. - 144 с.; ил.
3. Хювенен Э., Сеппанен Й. Мир Лиспа. В 2-х тт. М.: Мир, 1985.
4. А. Филд, П. Харрисон. Функциональное программирование. Пер. с англ. - М.: Мир, 1993. - 638с.
5. Кречко Ю. А. AutoCAD: Программирование и адаптация. – М.: ДИАЛОГ-МИФИ, 1995.
6. Полещук Н. Visual LISP и секреты адаптации AutoCAD. – СПб.: БХВ-Петербург, 2001.
7. Романычева Э. Т., Сидорова Т.М., Сидоров С.Ю. AutoCAD. Практическое руководство. Версии 12, 13, 14. – М.: ДМК. Радио и связь, 1998.

Толстоба Надежда Дмитриевна

Системы автоматизированного конструирования

Методические указания

В авторской редакции

Компьютерное макетирование Н.Д. Толстоба

Зав. редакционно-издательским отделом Н.Ф. Гусарова

Лицензия ИД № 00408 от 05.11.99

Подписано к печати

Отпечатано на ризографе. Тираж 100 экз. Заказ № ___.

Редакционно-издательский отдел
Санкт-Петербургского государственного
института точной механики и оптики
(технического университета)

197101, Санкт-Петербург, ул. Саблинская, 14