Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Информационные технологии»

В. С. Мурашко

СИСТЕМЫ КОМПЬЮТЕРНОЙ ГРАФИКИ В АВТОМАТИЗИРОВАННОМ ПРОЕКТИРОВАНИИ

КУРС ЛЕКЦИЙ
по одноименной дисциплине для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной формы обучения
В двух частях
Часть 1

ЯЗЫК AUTOLISP

Электронный аналог печатного издания

Гомель 2009

УДК 004.92(075.8)
ББК 30.2-5-05я73
М91

Рекомендовано к изданию научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(*протокол № 1 от 22.09.2008 г.*)

Рецензент: начальник сектора разработки средств АСУ ГГТУ им. П. О. Сухого *Н. С. Шестакова*

Мурашко, В. С.

М91 Системы компьютерной графики в автоматизированном проектировании : курс лекций по одноим. дисциплине для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» днев. формы обучения. В 2 ч. Ч. 1. Язык AutoLISP / В. С. Мурашко. – Гомель : ГГТУ им. П. О. Сухого, 2009. – 110 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://lib.gstu.local. – Загл. с титул. экрана.

ISBN 978-985-420-848-0.

В курсе лекций даны основы программирования на языке AutoLISP. Рассматриваются язык управления диалоговыми окнами DCL и вопросы программирования диалоговых окон в AutoLISP, а также механизм доступа к графической базе данных AutoCAD. Предлагаются два подхода к формированию чертежей: вариантный метод и генерирующий метод. Уделено внимание разработке объектно-ориентированного интерфейса: создание падающих и графических меню.

Для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной формы обучения.

УДК 004.92(075.8)
ББК 30.2-5-05я73

ISBN 978-985-420-848-0 (ч. 1)
ISBN 978-985-420-856-5

Введение

Предлагается курс лекций по дисциплине «Системы компьютерной графики в автоматизированном производстве» для студентов специальностей «Информационные системы и технологии» в двух частях, часть 1 «Язык AutoLISP».

Цель данного курса лекций:

– дать основы программирования на языке AutoLISP;

– познакомить с программированием диалоговых окон на языке DCL;

– изучить механизм доступа к графической базе данных AutoCAD;

– рассмотреть подходы (методы) к автоматизации разработки и выполнения конструкторской документации в AutoCAD;

– разработка объектно-ориентированного интерфейса с помощью графического и падающего меню.

Главное предназначение системы AutoCAD – не рисование чертежей на компьютере, а создание на ее основе специализированной САПР определенного класса изделий.

LISP – важнейший язык, используемый в символьной обработке и исследованиях по искусственному интеллекту. Символьная обработка и методы объектно-ориентированного программирования хорошо подходят для разработки рисунков, чертежей, которая имеет место в проектировании с использованием компьютерных технологий. Чертеж – сложная структура данных. На языке LISP написано математическое обеспечение AutoCAD.

В настоящей работе будет рассмотрен AutoLISP – один из диалектов языка LISP. По синтаксису и соглашениям он близок к COMMON LISP, но имеет много дополнительных функций, отражающих специфику AutoCAD.

С помощью AutoLISP можно создавать условия, значительно ускоряющие процесс проектирования. Например, известно, что разработка конструкций и чертежей в большей степени сводится к их формированию из однотипных, унифицированных или стандартных элементов. Целесообразно разрабатывать для подобных элементов функции или команды, вписывающиеся в интерфейс AutoCAD. Обращение к таким командам можно организовать, используя падающие и графические меню. Для этого потребуется создать интерфейс, обеспечивающий общение с компьютером в привычных для конструктора терминах, определениях и изображениях.

Тема 1. ВВЕДЕНИЕ В ЯЗЫК AUTOLISP

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 AutoCAD и доступен в течение всего сеанса работы с AutoCAD.

Таким образом, графический редактор AutoCAD и интерпретатор языка AutoLISP представляют собой единую систему: любая функция AutoLISP может быть вызвана из графического редактора, и любая команда редактора может быть использована в программе на AutoLISP.

Возможности применения AutoLISP весьма широки и разнообразны. Наиболее характерны следующие классы применений:

1. *Программирование чертежей типовых деталей с параметризацией*. Создается программа, позволяющая при каждом обращении к ней формировать новый чертеж, отличающийся от чертежей, построенных этой же программой, размерами, а также, возможно, и топологией. Время получения чертежа с помощью такой программы может быть в десятки раз меньше времени, необходимого для его создания с помощью AutoCAD вручную. При этом экономится память.

2. *Создание и ведение графических баз данных из приложений, написанных на AutoLISP*. Программы на AutoLISP в сочетании с пользовательскими меню могут организовывать просмотр, поиск, выбор и вставку необходимых чертежей.

3. *Анализ и (или) автоматическое преобразование графической базы данных (БД) AutoCAD*.

4. *Расширение системы команд графического редактора AutoCAD и построение на базе AutoCAD специализированных САПР*. AutoCAD является открытой и развивающейся системой. Расширение системы команд графического редактора AutoCAD 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 AutoLISP является возможность создания пользовательских меню.

Программа на AutoLISP может решать такие задачи, как:

– обнаружение пересечений электрических и других магистралей в производстве;

– подсчет суммарной длины трасс;

– расчет площадей сложных областей, центра масс и моментов инерции и другие.

Программа также может быстро осуществить преобразование чертежа, на которое при работе «врукопашную» пришлось бы затратить значительное время, например: заменить блоки чертежа на другие, перенести выделенные объекты со слоя на слой, отобрать объекты определенного типа и модифицировать их.

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*, булевые 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 путем 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 AutoLISP;

– характерные для всех языков программирования и обеспечивающие компактное описание действий в программе за счет 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 AutoCAD:

– 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 приема 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 AutoCAD;

– для включения в программу любой команды AutoCAD. Причем 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 AutoCAD.

1.3. Использование программ на языке AutoLISP в системе AutoCAD

Рассмотрим возможные способы использования программ на AutoLISP в зависимости от того, куда помещена разработанная программа.

1. Непосредственный ввод с клавиатуры.

В ответ на приглашение

command:

(*команда:*)

набираются команды языка и последовательно выполняются.

Однако такой способ неудобен, так как при каждом повторном выполнении программу нужно вновь полностью набирать. Может использоваться для отладки фрагментов программ.

2. Запись программы в виде текстового файла с расширением .LSP с последующей загрузкой в AutoCAD.

Загрузка файла программы выполняется:

– с помощью команды AutoCAD **Загрузить (_Load**);

– с помощью меню Сервис/ AutoLISP / Загрузить…;

– с помощью команды в командной строке **Загприл(_.appload**)

Любая функция загруженной программы вызывается путем указания ее имени и, возможно, параметров функции, заключив их в круглые скобки:

(myprog "Привет")

3. Оформление программы как готовой команды AutoCAD.

Программу можно оформить так, чтобы после загрузки файла с этой программой ее можно было вызывать по имени точно так же, как вызываются команды AutoCAD (т. е. без скобок).

Для этого имя главной функции в программе (последней, которая описана через DEFUN) нужно начать с символов "C:". Например, описана функция C:QUADR. Такую программу можно вызвать на выполнение, набрав строку QUADR. Таким способом можно оформлять только функции, не имеющие аргументов.

4. Автоматическая загрузка программ.

Программы на AutoLISP будут загружаться автоматически при загрузке AutoCAD, если их поместить в файл под названием ACAD.LSP.

5. Автоматический запуск программы на AutoLISP.

Можно оформить программу на AutoLISP так, чтобы она автоматически выполнялась после входа в AutoCAD. Для этого ее нужно включить в файл ACAD.LSP под именем S::STARTUP.

Чтобы просмотреть значение переменной, надо в командной строке AutoCAD набрать восклицательный знак, а за ним – имя переменной.

1.4. Основные понятия языка AutoLISP

Скобки – основной управляющий символ языка LISP.

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).

Язык LISP создан американским ученым Джоном Маккарти в 1959 г. и нашел широкое применение. Он отличается высокой надежностью, функциональным стилем программирования и использованием обратной польской нотации.

Основное понятие языка LISP – список.

Список – перечень атомов или списков, отделенных друг от друга пробелами и заключенных в скобки. Списки могут быть вложенными.

Атом – простой (в отличие от списка) тип данных: число, символьная строка, функция.

В LISP нет различия между текстом программы и обрабатываемыми ею данными. В LISP и данные, и текст программы являются списками.

Выражение AutoLISP имеет вид:
(функция аргумент1 аргумент2 ... аргументN),
где *функция* – имя операции (в том числе и арифметической), которая должна быть выполнена. Число аргументов может быть больше 2.

Примеры

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.

Каждый атом имеет *имя*, дающееся по следующим правилам – допускаются английские буквы, цифры, большинство имеющихся на клавиатуре знаков за исключением «;», «(», «)», «.», «,» , « », строчные и заглавные буквы не различаются, первым символом должна быть буква.

Буква *T* зарезервирована и *не должна использоваться* в качестве имени атома. Слово *NIL* зарезервировано и *не должно использоваться* в качестве имени атома.

Длина имени формально не ограничена, но для экономии памяти рекомендуется не превышать длину в шесть знаков.

AutoLISP поддерживает следующие типы данных:

– целое число со знаком от –32768 до 32767 или от 0 до 65535 (2 байта) без знака;

– вещественное число, записываемое через десятичную точку: 10.52 или в экспоненциальном формате: 2.52E-12;

– строка символов длиной до 127 знаков, заключенная в двойные кавычки. Символ "\" является служебным и, если он нужен в тексте, должен удваиваться: текст "3\2" запишется как "3\\2";

– логический тип, принимающий два возможных значения: истина (обозначается *T*) или ложь (обозначается *NIL*);

– ссылка на встроенную функцию языка;

– ссылка на созданный программистом список (программу или данные);

– ссылка на переменную;

– ссылка на таблицу диспетчера виртуальной памяти.

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

1.5. Присваивание значений в AutoLISP. Встроенные функции

Следует различать два понятия: *переменная* и *значение переменной*.

Переменная – указатель на область динамической памяти, имеющей имя.

Значение переменной – данные, записанные в динамической памяти, начиная с адреса, записанного в переменной.

Для присваивания значений переменным в AutoLISP имеются две функции – *SET* и *SETQ*.

Функция SETQ меняет значение переменной, а не саму переменную. Аргументами функции является перечень пар «переменная» – «значение». Функция возвращает результат последнего присваивания.

Например, запись *(SETQ a 10)* помещает число 10 в область памяти, на которую указывает переменная *a* и одновременно задает тип переменной *a* – целое число. Можно в одной функции присвоить несколько переменных. Порядок выполнения нескольких присваиваний в функции *SETQ* определен слева направо.

Изменение значения переменной на *NIL* освобождает занимаемую ее значением область памяти.

Функция SET – работает не со значениями, а с самими переменными. Например, *(SET a b)* заставляет переменную *a* ссылаться на ту же область памяти, что и переменная *b*.

Функция *QUOTE* запрещает вычисление списка, являющегося ее аргументом и возвращает сам этот список.

Например, *(SETQ a (QUOTE (0.25 "АБВ" 46)))*, тогда значением переменной *a* станет список (0.25 "АБВ" 46). Введена сокращенная запись функции *QUOTE* в виде апострофа:
(SETQ a '(0.25 "АБВ" 46))

Функцию *SET* можно использовать и для изменения значения переменной:
(SET 'a 10) делает то же, что и *(SETQ a 10)*, поскольку запись '*a* означает значение переменной *a*.

Предположим теперь, что мы записали в значение переменной как данные список, который на самом деле является программой:

(SETQ a '(+ 5 10))

Как теперь заставить AutoLISP вычислить этот список? Для этого есть очень полезная функция *EVAL*.

Функция *EVAL* вычисляет список, являющийся ее аргументом, и возвращает вычисленное значение.

В нашем случае *(EVAL a)* вернет 15. Таким образом, функция *EVAL* обратна по отношению к функции *QUOTE*. Ее использование позволяет «на ходу», в ходе выполнения программы, сформировать список-программу (содержащий вызовы функций) и исполнить его. В этом и заключается динамическая модификация текста программы, необходимая для решения задач искусственного интеллекта.

Математические и ряд других встроенных функций AutoLISP представлены в таблице 1.1.

Пример

Вычислить площадь треугольника S со сторонами a, b, c по формуле Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)}, \qquad p = \frac{a+b+c}{2}.$$

Программа вычислений:

```
( SETQ p ( / ( + a b c ) 2 ) )
(SETQ s (sqrt ( * p ( - p a ) ( - p b ) ( - p c ))))
```

В языке также предусмотрена встроенная переменная *PI*, со значением, равным числу π.

Таблица 1.1

Основные встроенные функции AutoLISP

Имя функции	**Аргументы**	**Возвращаемое значение**
+	a_1 a_2 ... a_n	$a_1+a_2+...+a_n$
–	a_1 a_2 ... a_n	$a_1–a_2–...–a_n$
*	a_1 a_2 ... a_n	$a_1a_2...a_n$
/	a_1 a_2 ... a_n	$a_1/a_2/.../a_n$
1+	a	a+1
1–	a	a–1
ABS	a	$\lvert a \rvert$
SQRT	a	$\sqrt{a}$
EXP	a	e^a
EXPT	a b	a^b
GCD	a b	НОД чисел a, b
LOG	a	ln (a)
MIN	a_1 a_2 ... a_n	Минимальное из чисел a_1 a_2 ... a_n
MAX	a_1 a_2 ... a_n	Максимальное из чисел a_1 a_2 ... a_n
REM	a b	Остаток от деления a/b
FIX	n1	Возвращает целую часть числа n1
Тригонометрические функции		
SIN	a	sin(a), a – в радианах
COS	a	cos(a), a – в радианах
ATAN	a	arctg(a), a – в радианах
Строковые функции		
STRCASE	s b	Переводит все буквы в строке s в верхний (при b=NIL) или в нижний (при b=T) регистр

Окончание табл. 1.1

Имя функции	Аргументы	Возвращаемое значение
SUBSTR	s n1 n2	Возвращает часть строки s, начинающуюся с n1-го символа и имеющую длину n2 символа
CHR	n	Возвращает символ с ASCII-кодом n
STRLEN	s	Возвращает число символов в строке s
ASCII	s	ASCII-код первого символа в строке s
Функции преобразования типов		
ITOA	n	Целое число n в его текстовое представление s
ATOI	s	Преобразует текстовую строку s в целое число
RTOS	n1 n2 n3	Преобразует вещественное число n1 в текстовую сроку s; n2 – код формата числа (1 – научный; 2 – десятичный); n3 – точность (число знаков после запятой)
ATOF	s	Преобразует текстовую строку s в действительное число n

1.6. Создание собственных функций

Стандартный способ создания пользовательской функции заключается в использовании функции *DEFUN*. В общем виде она записывается следующим образом:

```
(DEFUN name ( a1 a2 ... an / v1 v2 ... vm )
( выражение1 )
( выражение2 )
....
( выражение N )
),
```

где *name* – имя функции; ai – *i*-й аргумент функции; vi – *i*-я локальная переменная.

Функция *DEFUN* создает в памяти пользовательскую функцию с именем *name* и списком аргументов *a1*, *a2*, ..., *an*. Для выполнения функция должна быть явно вызвана.

Пример

Введем функцию с именем tan, вычисляющую тангенс угла, следующего вида:

```
(DEFUN tan ( a )
  ( / ( SIN a ) ( COS a ) )
)
```

Все переменные в AutoLISP делятся на два вида: глобальные и локальные.

Глобальные переменные постоянно находятся в оперативной памяти и, следовательно, доступны из любой функции. Глобальные переменные создаются автоматически при присваивании им значения. Например, функция *(SETQ a 5)* создает глобальную переменную *a*.

Локальные переменные явно описываются в заголовке функции *DEFUN* и видны только внутри соответствующей пользовательской функции. В начале работы пользовательской функции локальные переменные имеют значение NIL. По окончании работы этой функции они автоматически удаляются из памяти.

Аргументы пользовательских функций также являются локальными переменными. В начале работы пользовательской функции аргументы принимают значения, переданные функции при ее вызове.

Пользовательская функция может не иметь как локальных переменных, так и аргументов. Поэтому возможны записи:

(DEFUN	(DEFUN	(DEFUN	(DEFUN
s (a b / d)	s (a b)	s (/ c d)	s ()
…)	…)	…)	…)

В любом случае пара скобок после имени функции сохраняется.

Пример

Угол в тригонометрических функциях должен выражаться в радианах. Определим функции для преобразования градусов в радианы *dtr* и радианов в градусы *rtd.*

```
;функция преобразования градусов в радианы
 (defun dtr (a)
 (* pi (/ a 180.0)))
; функция преобразования радианов в градусы
 (defun rtd (a)
 (/ (* a 180.0) pi))
```

В дальнейшем мы будем использовать эти функции.

Пример

```
 (setq a 30)
 (setq S (sin (dtr a)))
! S
```

0.5 – возвращает значение sin (30).

Для случая частого использования функций dtr и rtd следует поместить их описание в файл acad.lsp, который загружается AutoCAD автоматически.

Пример

Написать функцию вычисления площади треугольника со сторонами *a*, *b*, *c*. Воспользуемся формулой Герона. Назовем функцию *trisqu.* Ее аргументами будут стороны треугольника. Введем локальные переменную *p* для записи полупериметра и локальную переменную *s* для записи площади треугольника.

```
( DEFUN trisqu ( a b c  / p  s)
 ( SETQ p ( / ( + a b c ) 2 )
s (sqrt ( * p ( - p a ) ( - p b ) ( - p c ))))
 )
```

При вызове функции *trisqu* ей обязательно должны быть переданы три аргумента, значения которых запишутся в локальные переменные *a*, *b*, *c*, например: *(trisque 3 4 5)* или *(trisque a 4.5 (/ 2 b))*.

1.7. Организация диалога с пользователем

Для интерактивного запрашивания информации в AutoLISP есть набор функций, обеспечивающих ввод-вывод информации.

Экран AutoCAD может работать в двух режимах: текстовом и в графическом. Эти режимы переключаются функциями AutoLISP: *(TEXTSCR)* и *(GRAPHSCR)*

Для вывода на текстовый экран предназначены следующие функции (табл. 1.2).

Таблица 1.2

Функции вывода информации на текстовый экран

Наименование	Аргументы	Описание
PRINC	Любое выражение	Вывод выражения на экран без учета управляющих кодов
PRIN1	Любое выражение	Вывод выражения на экран с учетом управляющих кодов
PRINT	Любое выражение	Вывод выражения на экран с учетом управляющих кодов, с новой строки и с пробелом в конце
PROMPT	Текст	Вывод текста на экран с учетом управляющих кодов
TERPRI	Нет	Вывод пустой строки

AutoLISP «понимает» следующие управляющие коды в выводимых на экран текстовых строках (табл. 1.3).

Таблица 1.3

Управляющие коды

Код	Значение
\e	Символ с кодом 27 (ESC)
\n	Переход на новую строку
\r	Переход в начало той же строки
\t	Переход на следующую позицию табуляции (8 пробелов)
\nnn	Ввод символа с восьмеричным кодом nnn

Если нужно просто вывести на печать символ "\", его нужно удвоить, т. е. написать "\\".

Функция PRIN1 не возвращает значения и удобна для «тихого» завершения работы головной программы.

Ввод данных осуществляется семейством *GET-функций* (табл. 1.4).

Таблица 1.4

Ввод данных с клавиатуры

Наименование	Аргументы	Описание
GETINT	Текст подсказки	Ввод целого числа
GETREAL	Текст подсказки	Ввод вещественного числа
GETSTRING	Флаг пробела (T или NIL) Текст подсказки	Ввод текста. Если флаг пробела =T, в тексте могут быть пробелы, иначе (по умолчанию) пробел воспринимается как окончание ввода
GETPOINT	[p1] s	Ввод координат точки p при помощи мышки. s – текст подсказки. Если задана точка p1, то от нее к текущей точке отображается "резиновая линия"
GETANGLE	[<точка>] [<подсказка>]	Эта функция создает паузу для того, чтобы пользователь ввел угол
GETCORNER	<точка> [<подсказка>]	Возвращает точку так же, как GETPOINT. Однако для GETCORNER требуется <базовая точка> и функция строит прямоугольник от <базовой точки>, в то время, как пользователь передвигает курсор по экрану
GETDIST	[<точка>] [<подсказка>]	Функция создает паузу для того, чтобы пользователь ввел расстояние. Вы можете указать расстояние, набрав число на клавиатуре в текущих для AutoCADa единицах измерения. Вы можете так же «показать» AutoLISP расстояние, указав две точки на экране. AutoCAD рисует «резиновую» линию от первой точки до текущего положения курсора для того, чтобы визуалировать расстояние

Окончание табл. 1.4

Наименование	Аргументы	Описание
GETKWORD	[<подсказка>]	Эта функция запрашивает ключевое слово у пользователя. Список имеющих смысл ключевых слов задается, прежде чем вызывается функция GETKWORD, пользуясь функцией INITGET (описывается ниже). GETKWORD возвращает ключевое слово, соответствующее введенному пользователем как строковую константу. AutoCAD переспросит, если введенное не является одним из заданных ключевых слов. Пустой ввод возвращает nil (если позволяется вводить пустой ввод). Если не было установлено ни одного ключевого слова, также возвращается nil
INITGET	n	NIL устанавливает защиту от неправильного ввода на одну следующую GET-функцию. n – сумма следующих значений: 1 – запрет пустого ввода; 2 – запрет ввода нуля; 4 – запрет ввода отрицательных чисел; 8 – не контролируются пределы, даже при включенном LIMCHECK; 16 – возвращение трехмерных точек, предпочтительнее, чем двухмерных точек; 32 – применяется пунктир для изображения «резиновой» линии или рамки

Пример

```
(initget 1 "Yes No")
(setq x (getkword "Are you sure? (Yes or No) "))
```

запросит пользователя и установит в переменную *x* либо «Yes», либо «No», в зависимости от его ответа. Если ввод не соответствует ни одному ключевому слову, или же пользователь ответил пустым вводом, AutoCAD попросит пользователя повторить ввод еще раз.

Список управляющих сигналов и ключевых слов, устанавливаемые INITGET, применимы только к следующей за INITGET вызываемой GETxxx функции (кроме GETSTRING и GETVAR) и затем автоматически сбрасываются. Это позволяет Getxxx-функций избежать необходимости при следующем вызове очищать специальные состояния.

Пример

Написать интерактивную программу вычисления площади треугольника

```
( DEFUN C:trisque ( / a b c p s )
 ( TEXTSCR )
( SETQ a ( GETREAL "\n Длина стороны A:" ) )
( SETQ b ( GETREAL "\n Длина стороны B:" ) )
( SETQ c ( GETREAL "\n Длина стороны C:" ) )
 ( SETQ p ( / ( + a b c ) 2 ) )
 (SETQ s (expt   (* p ( - p a) (- p b) ( - p c)) 0.5 ) )
( PROMPT "\nПлощадь треугольника равна "  )
( PRIN1 s ) )
```

Для вывода сообщений на графический экран предназначена функция *(alert <text>)*.

Тема 2. ВЫЗОВ КОМАНД AUTOCAD

2.1. Функция вызова команд AutoCAD

Большинство команд AutoCAD могут быть выполнены из программы на AutoLISP 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: *МТЕКСТ (_.DTEXT), РАСТЯНУТЬ (_.STRETCH), ПЕЧАТЬ (_.PLOT),* 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

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

```
Команда: _line Первая точка: 1,1
Следующая точка или [Отменить]: @21,0
Следующая точка или [Отменить]: @0,21
Следующая точка или [Замкнуть/Отменить]: @-21,0
Следующая точка или [Замкнуть/Отменить]: з
```

На AutoLISP это будет выглядеть так:

```
(command "_.line" "1,1" "@21,0" "@0,21" "@-21,0" "З" )
```

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 AutoLISP. 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.).

Элементами в программе могут быть:

– *числа*, для которых можно использовать функции преобразования типов (***itoa, rtos, atof, atoi***);

– *строки* – преобразование строк – ***strcat, substr***;

– *точка* – ввод точки 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 AutoLISP для указания координат. Например, если координата *X* точки записана в переменной *A*, а координата *Y* равна $\frac{(A+20)^2}{4}$, то следует записать:

```
( list A ( / ( * ( + A 20 ) ( + A 20 ) ) 4 ) )
```

Язык AutoLISP имеет ряд специальных встроенных функций для вычисления координат точек.

Основная геометрическая функция – ***polar:***

(polar a angle dist),

где *a* – список из двух элементов (координаты точки);

angle – угол в радианах;

dist – расстояние в текущих единицах измерения.

Функция *polar* возвращает в виде списка координаты точки, отстоящей от точки *a* на расстояние *dist* под углом *angle*.

Положительное направление отсчета углов – против часовой стрелки.

При геометрических расчетах используются также следующие функции:

– функция (*inters p1 p2 p3 p4 признак*) возвращает точку пересечения двух отрезков, проходящих через точки *p1*, *p2*, *p3*, *p4* соответственно. *Признак* показывает, следует ли находить точку пересечения бесконечных прямых, проходящих через точки *p1* и *p2* и *p3* и *p4* (если *признак* равен NIL) или же только отрезков (если *признак* не равен NIL). Если точка пересечения отсутствует, функция возвращает NIL;

– функция (*angle p1 p2*) возвращает угол в радианах между положительным направлением оси X и прямой, проходящей через точки *p1* и *p2*;

– функция (*distance p1 p2*) возвращает расстояние от точки *p1* до точки *p2* в текущих единицах измерения расстояний.

2.2. Системные переменные AutoCAD

Единицы измерения, как и многие другие параметры, определяются значениями системных переменных AutoCAD. *Системная переменная* – ячейка памяти, содержащая определенное значение и имеющая неизменное имя. Значения системных переменных задают различные режимы работы команд AutoCAD.

К системным переменным нельзя обращаться напрямую, как к обычным переменным AutoLISPa. Для доступа к системным переменным в AutoLISPe имеются две функции.

Функция (getvar "имя") возвращает значение системной переменной с именем *имя*, заданным как текстовая строка. Например, системная переменная "LASTPOINT" содержит координаты текущей точки. Для их использования в программе следует использовать функцию getvar в виде:

(getvar "LASTPOINT")

Если в ходе отрисовки полилинии следующую точку удобнее рассчитать от предыдущей при помощи функции *polar*, необязательно записывать все промежуточные точки в переменные. Можно использовать функцию *getvar*, например:

```
( COMMAND "_.PLINE" ( LIST ( + A 10 ) ( - B 20 ) )
 ( POLAR ( GETVAR "LASTPOINT" ) 0 40 ) "" )
```

В приведенном примере координаты начальной точки рассчитываются. Чтобы не записывать эту точку в отдельную переменную, следующая точка, координаты которой рассчитывается при помощи функции *polar*, использует в качестве опорной координаты текущей (т. е. начальной) точки, всегда записываемые в виде списка в системную переменную *"LASTPOINT"*.

Функция *(setvar "имя" значение)* меняет значение соответствующей системной переменной. Эти значения записываются в файл чертежа. Часть системных переменных (например, переменная, содержащая номер версии Автокада) доступна только для чтения и их значения нельзя изменить.

2.3. Описание вызова команд AutoCAD из AutoLISP

Вызов команд AutoCAD в командной строке можно осуществлять как на русском языке, так и на английском. Рассмотрим некоторые из них.

Работа с цветом. Изменение текущего рабочего цвета. Команда *Цвет (_Color)*

(command "_.color" "цвет")

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.

```
;; задаем цвет, соответствующий основному цвету слоя
 (command "_.color" "_bylayer")
;; цвет, рассчитанный по счетчику
(command "_.color" (itoa col))
```

Команда *Кольцо (_Donut)*

(command "_.donut" "диаметр_внутренний" "диаметр_внешний" точка_установки1 точка_установки2 ... до завершения – то есть до знака ""),

где диаметр_внутренний – центр бублика, если=0, будет закрашенный круг;
диаметр_внешний – диаметр круга;
точка_установки – центры установки колечек.

```
;;Круг диаметром 3 устанавливаем в ранее определенные точки p1, p2, p3
(command "_.donut" "0.0" "3.0" p1 p2 p3 "")
;;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" "")
```

Разъединение группы примитивов – команда *Расчленить (_Explode)*

```
;превращение полилинии в набор линий и дуг
(command "_.explode" p "")
;P- точка на объекте или выбор объектов.
```

Сетка визуальная – управление отображением – команда *Сетка (_Grid)*

(command "_.grid" "O")

Другие опции:

```
Шаг сетки(X) или [Вкл/Откл/Шаг привязки/Аспект] <10.0000>:
Grid spacing(X) or ON/OFF/Snap/Aspect <10.0000>:
```

Вставка блока – команда Вставить (_Insert)

(command "_.insert" "имя" "точка_вставки" "масштаб по X" "по Y" "угол разворота")

```
; Вставка блока рамки
(command "_.insert" blockf "0,0" "1" "1" "0")
```

Управление слоями – команда *Слой(_Layer)*

(command "_.layer" опции)

Задайте опцию [?/Создать/Установить/Новый/Вкл/Откл/Цвет/Тип линий/вЕс линий/ Печать/Пстиль/Заморозить/Разморозить/Блок / Разблок/ Конфигурация

```
;; Создание слоя "sizes", придание ему имени и цвета. Активизация соз-
;данного слоя.
(command "_.layer" "Н" "sizes" "Ц" "3" "sizes" "у" "sizes" "")
;;Переход на слой "0"
(command "_.layer" "У" "0" "")
```

Опции для английской версии

```
?/Make/Set/New/ON/OFF/Color/Ltype/LWeight/Plot/Freeze/Thaw/LOck/
Unlock/stAte
```

Рисование линии – команда *Отрезок (_Line)*

(command "_.line" p1 p2 ... pn "")

p1 и прочее – точки, через который проходит прямая. Для окончания ломаной – пустой ввод - "".

Смена типа линии – команда *Типлин (_.Linetype)*

(command "_.linetype" "опции")

Задайте опцию [?/Создать/Загрузить/Установить]

```
;; ставим тип линии в соответствии со слоем
(command "_.linetype" "У" "bylayer" "")
;; ставим тип линии штрихпунктирный
(command "_.linetype" "У" "Невидимая2" "")
```

Установка масштаба линнй. Команда глобальный масштаб типов линий *Лмасштаб (_Ltscale)*

(command "_.ltscale" "масштаб")

Иногда для верного отображения линий необходимо корректировать масштаб отображения линий.

Установка привязок к элементам чертежа – команда *Привязка (_Osnap).*

(command "_.osnap" "привязки")

Совет. Использовать привязки необходимо только для организации ввода данных пользователем. Если вы в процессе рисования – привязки только мешают. Их лучше отключать при работе в чертеже, иначе вы можете получить неверный чертеж.

```
;;отключение привязок
(command "_.osnap" “Ничего")
```

Привязки:

КОНточка	ЦЕНтр	КАСательная
СЕРедина	УЗЕл	БЛИжайшая
ПЕРесечение	КВАдрант	ПАРаллельно
ПРОдолжение	ТВСтавки	
КАЖущееся пересечение	НОРмаль	

```
;; подключение привязок к пересечению, к ближайшей точке, к конечной
;точке, перпендикулярно, и к центру окружности соответственно
(command "_.osnap" "пер,бли,кон,нор,цен")
(command "_.osnap" "_int, _nea, _end, _per, _cen")
```

Рисование полилинии – команда *Плиния (_Pline)*

(command "_.pline" нач. точка точки или опции)

Опции для линии

Следующая точка или [Дуга/Полуширина/длИна/Отменить/Ширина]:

Опции для дуги

[Угол/Центр/Замкнуть/Направление/Полуширина/Линейный/Радиус/Вторая/Отменить/Ширина]

```
(command "_.pline" p1 p2 p3 ""); ломаная
(command "_.pline" p1 "д" "в" p2 p3 ""); дуга
```

Английская версия.

Опции для линии

Arc/Close/Halfwidth/Length/Undo/Width/<Endpoint of line>

Опции для дуги

Angle/CEnter/CLose/Direction/Halfwidth/Line/Radius/Second pt/ Undo/ Width/<Endpoint of arc>:

Рисование прямоугольника – команда *Прямоуг (_Rectangle)*

(command "_.rectangle" p1 p2)

Рисование квадрата с помощью функции *Мн-Угол (_Polygon)*

(command "_.polygon" "4" pnt "_i" dstn),

где pnt – центр квадрата; _i – вписанного в окружность; dstn – радиус окружности. Для квадрата, описанного вокруг окружности, используют ключ_C.

Рисование круга с помощью команды *Круг (_Circle)*

(command "_.circle" pnt dstn),

где pnt – центр круга; dstn – радиус круга.

Рисование точки на экране – команда Точка (_Point)
(command “_.Point” bp),

где bp – точка вставки «Точки».

Очистка экрана – команда *Стереть (_Erase)*

```
(command "_.erase" "_all" "")
```

Регенерировать изображение (обновить) – команда *Реген (_Regen)*

(command "_.regen")

Создание стиля текста и активизация стиля – команда Стиль (_Style)

(command "_.style" "имя стиля" "шрифт" "высота" "коэф. ширины" "угол" "Перевернутый? <N>" "Справа налево? <N>" "Вертикальный? <N>")

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

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

Вывод текста – команда *Текст (_Text)*

(command "_.text" "опции" "текст")

Вывод производится в определенную точку. Для текста эта точка является по умолчанию нижней левой точкой (ВЛ). Изменить это можно с помощью опций команды из цикла

Выравнивание:
[впИсанный/Поширине/Центр/сЕредина/впРаво/ВЛ/ВЦ/ВП/СЛ/СЦ/СП/НЛ/НЦ/НП]:

Английская версия

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 Поширине). Обратите внимание на задание второй точки в
;относительных полярных координатах.
(command "_.text" "П" p1 "@17<0" txt)
;; вывод текста с указанием центральной точки
(command "_.text" "Ц" "189,148" "0" txt)
```

Увеличение, вид, показать – *Показать (_Zoom)*

(command "_.zoom" "опции")

Опции
[Все/Центр/Динамика/Границы/Предыдущий/Масштаб/Рамка] <реальное время>

```
;; Увеличение, при котором все элементы чертежа видны на рабочем
;поле
 (command "_.zoom" "в")
```

Опции в английской версии
All/Center/Dynamic/Extents/Previous/Scale(X/XP)/Window/<Realtime>:

```
;; Увеличение окном, с указанием точек - границ окна
 (command "_.zoom" "p" p1 p2)
```

Создание блока примитивов и сохранение их в отдельном файле – команда *Пблок (_Wblock)*
(command "_.wblock" "имя_файла" "имя_блока" "точка_вставки" "выделение элементов чертежа" "")

```
;Выделение элементов - хоть по одному, хоть рамкой.
;; 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 листа формата ;A4
(command "_.wblock" ff "" "0,0" "_c" "0,0" "210,297" "")
```

Вывод штриховки – команда *Штрих (_Bhatch)*
(command "_.bhatch" разные опции)
Опции
Properties/Select/Remove islands/Advanced /<Internal point>:
Specify a point or enter an optionProperties:
Enter pattern name or [? / Solid / User defined] <current>:
Advanced: Boundary set / Retain boundary / Island detection / Style / Associativity / <eXit>:

```
;; вывод штриховки для области, указанной внутренней точкой 12,12
; с масштабом 0,5 и углом наклона 30º
(command "_.bhatch" "_p" "ansi31" "0.5" "30" "12,12" "")
```

Вместо ключа “_p” (Properties) можно указать русский ключ “с” (Стандартная)

Размер линейный – команда *Рзмлинейный (Размер1) (_Dimlinear)*

Простановка линейного размера между точками *p1* и *p2* расположение размерного текста в точке *p3* (рис. 2.1).

(command "_.dimlinear" p1 p2 p3)

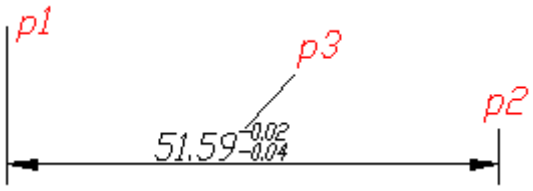


Рис. 2.1

```
(COMMAND "РАЗМЕР1" "ВЕРТИКАЛЬНЫЙ" Т2 Т3 TR "%%c<>")
(COMMAND "РАЗМЕР1" "ГОРИЗОНТАЛЬНЫЙ" ВР Т3 TR "")
(COMMAND  "РАЗМЕР1" "БАЗОВЫЙ" Т3  "" )
```

Радиус – команда *Рзмрадиус* (*_Dimradius*)
(command "_.dimradius" p1 p2)

Точка на дуге – p1 определяет и расположение размерной линии по направлению к центру окружности. *Случай 1* (рис. 2.2, *а*) обозначение радиуса внутри окружности – в этом случае размерная линия до центра не доходит. *Случай 2* (рис. 2.2, *б*) размер ставится вне окружности. Тогда размер идет от центра окружности через точку *p1* к указанной точке *p2*.

а) *б*)

Рис. 2.2

Стиль размеров – управляет сохранением, удалением, активизацией стилей. Команда ***Рзмстиль (_Dimstyle)***
(command "_.dimstyle" "ключ" "имя_стиля")

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.

```
;;Активизация стиля:
(command "_.dimstyle" "y")
```

Стрелочка – выноска – команда *Выноска (_Leader)*

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

Этот набор параметров позволяет нарисовать стрелку от *p1* до *p2* без подписей.

Удаление информации из памяти чертежа

(command "_.purge" "опции")

Опции

Purge unused Blocks/ Dimstyles/LAyers/LTypes/Shapes /Styles /Mlinestyles/ All:

```
;; Удалить все неиспользуемые слои
(command "_.purge" "_la" "" "_n")
```

Тема 3. AUTOLISP – ЯЗЫК ОБРАБОТКИ СПИСКОВ

3.1. Списки и их применение. Точечные пары

Любой, кто знаком с такими языками программирования как Бейсик, Паскаль, С, Фортран, заметит в AutoLISP ряд отличий от этих языков, в частности, отсутствие такого важного способа организации данных, как массивы (переменные с индексами), обеспечивающего компактную запись повторяющихся действий с помощью циклов. Подобные задачи решаются в AutoLISP с помощью списков.

Как уже известно, геометрические точки представляются в AutoLISP в форме списков из двух или трех координат. В дальнейшем будет показано, что примитивы AutoCAD также представляются в виде сложных списковых структур. Таким образом, для понимания специфики AutoLISP и его возможностей по обработке геометрических и графических данных AutoCAD, необходимо изучить функции работы со списками.

Как указывалось, список – это объект формы:

< список >::=([< элемент > ...]),

где < элемент > – это список или атом.

Особым случаем списка является точечная пара:

< точечная пара >::=(< атом > . <атом>)

При хранении точечных пар требуется меньше места в памяти, чем при хранении обычных списков. Точечная пара может быть сформирована функцией ***CONS***. Некоторые функции обработки списков не могут работать с точечными парами. Если точечная пара выдается на экран, то между атомами AutoLISP печатает точку. Об использовании точечных пар будет сказано позже.

3.2. Создание списков

Можно сформировать список (из двух или трех элементов) и записать его в качестве значения переменной путем ввода точки с помощью ***GETPOINT***.

Список можно сформировать и путем указания примитива на экране, «развернуть» его функцией ***ENTGET***. Кроме того, имеются функции, позволяющие конструировать списки из данных, определенных ранее в программе.

Функция ***LIST*** формирует список из любого числа элементов (атомов или списков) и возвращает список

(LIST < элемент > ...).

Функция ***(CONS <элемент> <список>)*** добавляет < элемент > (атом или список) в качестве первого элемента к уже существующему списку и возвращает обновленный список.

Таким образом, если функция ***LIST*** формирует новый список, то ***CONS*** – расширяет существующий. Однако возможно использование ***CONS*** и в следующей форме:

(CONS <атом> <атом>)

В этом случае создается точечная пара.

Функция ***(APPEND <список>...)*** соединяет несколько списков в один общий список. Отличие ***APPEND*** от ***LIST*** поясняется примерами, приведенными ниже.

Функция ***(REVERSE <список>)*** возвращает список с элементами, расставленными в обратном порядке.

Примеры

(SETQ A '(1.0 2.0))
(SETQ B '(C D E))
(SETQ F (LIST A B)) – создает в F список ((1.0 2.0)(C D E)).
(SETQ G (LIST 'X B)) – создает в G список (X (C D E)).
(SETQ G (CONS 'X B)) – создает (X C D E).
(SETQ G (CONS '(X) B)) – создает ((X) C D E).
(SETQ G (CONS 3 4)) – создает точечную пару (3.4).
(SETQ H (APPEND A B)) – создает (1.0 2.0 C D E), т. е. происходит слияние списков в отличие от функции ***LIST***, формирующей «список списков».
(SETQ H (APPEND '(X) B)) – создает (X C D E) – в отличие от функции ***CONS*** с такими же аргументами.
(SETQ H (APPEND 'X B)) – ошибка, аргументы должны быть списками.
(SETQ B (REVERSE B)) – создает в B (E D C).

3.3. Выделение элементов списков

Любая структура данных имеет смысл тогда, когда возможен доступ к отдельным элементам этой структуры. При работе со списками необходимо выделять элементы и другие части списков. Например, если необходимо изменить координату *X* точки, заданной как *(LIST X Y)*, то элемент *X* предварительно следует извлечь из списка:

Функция ***(CAR < список >)*** возвращает первый элемент списка.

Если список пуст, то возвращается NIL.

Например:

(car '(a b c)) – возвращает a
(car '((a b) c)) – возвращает (a в)
(car '()) – возвращает nil

Функция ***(CDR < список >)*** возвращает новый список, отличающийся отсутствием первого элемента (обрезает список спереди). Если список пуст, возвращается NIL.
(cdr '(a b c)) – возвращает (*b c*)
(cdr '((a b) c)) – возвращает (*c*)
(cdr '()) – возвращает nil

Когда аргументом <списка> является точечная пара (***CONS***), ***CDR*** возвращает второй элемент, не заключая его в список.
(cdr '(a . b)) – возвращает *b*
(cdr '(1 . "Text")) – возвращает "Text"

В AutoLISP определены функции, обеспечивающие последовательное выполнение ***CAR*** и (или) ***CDR*** вплоть до четырех уровней вложенности: ***CAAR, CADR, CDDR, CADAR*** и т. д.

Пусть
(SETQ X '((A B) C D)),
тогда
(CAAR X) эквивалентно ***(CAR (CAR X))*** возвращает A
(CDAR X) эквивалентно *(CDR (CAR X))* возвращает (B)
(CADAR X) эквивалентно *(CAR (CDR (CAR X)))* возвращает B
(CADR X) эквивалентно *(CAR (CDR X))* возвращает C
(CDDR X) эквивалентно *(CDR (CDR X))* возвращает (D)
(*CADDR X*) эквивалентно (*CAR* (*CDR* (*CDR X*))) возвращает D

Функции типа ***CAR*** и ***CDR*** часто используются для выделения координат точек (CAR – для *X*, CADR – для *Y*, CADDR – для *Z*).

Функция ***(LAST < список >)*** возвращает последний <элемент> *списка*, который может быть атомом либо списком.

Функция ***(MEMBER < элемент >< список>)*** просматривает <список > и ищет в нем заданный < элемент >. Если <элемент> найден, то возвращается список от <элемента> до конца исходного <списка>. Иначе возвращается NIL.

Функция ***(NTH < номер > < список>)*** возвращает элемент <списка>, имеющий заданный <номер>. <Номер> должен иметь целое неотрицательное значение. Примем во внимание, что первый элемент списка имеет номер 0. Если <номер> превышает число элементов в < списке > –1, то возвращается NIL.

Примеры

Пусть
(SETQ A '(B C (D E))),
тогда
(LAST A) возвращает (D E)
(MEMBER 'C A) возвращает (C (D E))

(NTH 0 A) возвращает B
(NTH 2 A) возвращает (D E)
(NTH 3 A) возвращает NIL

Функция ***(ASSOC <элемент> <список>)*** обеспечивает «ассоциативный поиск» в структурированном списке, т. е. в списке, элементы которого являются списками. Эти списки построены по схеме:

(< ключевой элемент >< значение > ...)

Поиск проводится по совпадению <элемента>, заданного в функции ***ASSOC***, с <ключевым элементом> в списке – элементе <списка>, заданного в ***ASSOC***. Если поиск удался, то возвращается найденный список, иначе – NIL. Таким образом, осуществляется поиск данных по ключу в сложной структуре данных. 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.).

Пусть
(SETQ DETAL '((КОД 123456) (КОДМ 001234) (М 1.45))),
тогда
(ASSOC 'КОДМ DETAL) возвращает *(КОДМ 001234)*

Теперь с помощью функции CADR можно «отрезать» ключевой элемент КОДМ и выделить значение 001234.

Функция ***(SUBST < новый элемент > < старый элемент > <список>)*** дополняет функцию ASSOC, позволяя заменять элементы в списке.

Производится просмотр списка, и каждый найденный <старый элемент> заменяется на <новый элемент>. Возвращается обновленный список.

3.4. Анализ списков

Имеются функции проверки условий, связанные с анализом списков.

Функция ***(LISTP <элемент>)*** возвращает *T*, если <элемент> является списком, иначе – NIL.

Функция ***(ATOM <элемент>)***, обратная к ***LISTP***, т. е. возвращает *T*, если <элемент> является атомом.

Функция ***(EQ <выражение1> <выражение2>)*** проверяет идентичность двух выражений, которые, как правило, являются списками. Списки считаются идентичными, если они связаны друг с другом через

SETQ. Если два списка идентичны (***EQ***), то они всегда равны (***EQUAL***). Если же два списка равны, то они необязательно идентичны.

Пример

Пусть

(SETQ X '(A B C)) (SETQ Y X),

тогда

(EQUAL X Y) дает T

(EQUAL X '(A B C)) дает T

(EQ X Y) дает T

(EQ X '(A B C)) дает NIL

3.5. Работа со списками

Как отмечалось в п. 3.1, имеются возможности компактной записи некоторых действий с помощью списков. Для этого можно использовать функции *APPLY*, *FOREACH*, *MAPCAR*, *LAMBDA*.

Функция ***(APPLY '< имя функции > <список>).***

Таким образом, выполняется какая-либо функция AutoLISP, имя которой указано и которая может иметь несколько аргументов. Эти аргументы задаются < списком >. Например,

(APPLY '+ '(1 2 3)) эквивалентно (+ 1 2 3)

Функция ***(FOREACH <переменная> <список> <выражение> ...)*** организует циклическое выполнение <выражений>. Предполагается, что в этих выражениях присутствует <переменная>. Функция работает так: присваивает <переменной> значение первого элемента <списка>, подставляет это значение в <выражения> и выполняет их. То же самое делается для каждого последующего элемента <списка>.

Пример

(SETQ E 0)

(FOREACH I '(1 2 3) (SETQ E (+ E I)))

эквивалентно последовательности

(SETQ E (+ E 1))

(SETQ E (+ E 2))

(SETQ E (+ E 3))

Функция ***(MAPCAR '<имя функции> <список1>... <список N>)*** также организует цикл. В нем выполняется функция, имя которой указано, имеющая *N* аргументов, т. е. столько, сколько перечислено <списков>. Во всех <списках> при этом должно быть одинаковое число элементов. Таким образом, цикл будет иметь столько шагов, сколько элементов в любом <списке>. Возвращается список значений функции, полученных для каждого номера элементов < списков >.

Пример

(SETQ S (MAPCAR ‘ + '(10 20 30) '(1 2 3)))

Формируется *S* в виде списка (11 22 33), каждый элемент которого есть результат применения функции "+" к паре аргументов, взятых из двух списков.

Пример

Все элементы списка размеров *size* нужно умножить на масштабный коэффициент *k*. Функция умножения требует как минимум два аргумента, в данном случае – текущий элемент списка и коэффициент *k*. Придется сформировать вспомогательный список, состоящий из стольких значений *k*, сколько есть размеров в списке *size*. При таком решении приходится создавать лишний список *coeff*. Делается это так.

```
( SETQ coeff NIL )
( REPEAT ( LENGTH size )
( SETQ coeff ( APPEND coeff ( LIST k ) ) )
```

После этого можно применить функцию MAPCAR:

```
( SETQ size ( MAPCAR '* size coeff ) )
```

Было разработано так называемое λ-исчисление Черча. Само по себе оно представляет математический аппарат для описания функций. В AutoLISP λ-исчисление заключается в наличии возможности создать «одноразовую» функцию, даже не имеющую своего имени. Причем внутри этой функции будут видны все текущие локальные переменные. «Одноразовая» функция создается функцией ***(LAMBDA (arg1 ... argn) exp1 ... expm)***. Здесь *arg1 ... argn* – список аргументов «одноразовой» функции, а *exp1 ... expm* – выражения AutoLISP, выполняющие какие-то операции над аргументами.

Функцию LAMBDA можно прямо записать как аргумент функции MAPCAR, не забыв поставить апостроф для подавления ее выполнения: (MAPCAR '(LAMBDA (x) (* x k)) size).

Здесь определена функция с одним аргументом *x*, которая вычисляется для каждого элемента списка *size*. Полученные произведения «слепливаются» в список, являющийся возвращаемым значением функции MAPCAR.

Тема 4. УПРАВЛЕНИЕ ПОРЯДКОМ ВЫЧИСЛЕНИЙ. ФУНКЦИИ ОБРАБОТКИ СТРОК

4.1. Управляющие конструкции AutoLISP – ветвление

Рассмотрим управляющие конструкции AutoLISP, которые в обязательном порядке содержат проверку условия. В качестве условий в AutoLISP используются логические функции, возвращающие *T* (true – истина) или *NIL* (ложь) и функции сравнения: «<», «>», «<=», «>=», «=», «/=».

Функции сравнения могут применяться к целым и вещественным числам, а также к текстовым строкам, но не к спискам.

Если сравниваются вещественные числа, то следует помнить об ограниченной точности вычислений с ними. Для этого следует использовать специальную функцию сравнения с заданной точностью *(EQUAL e1 e2 точность)*. Здесь *точность* – число, указывающее, сколько знаков поле запятой принимается во внимание при сравнении выражений *e1* и *e2*.

Функции сравнения могут объединяться при помощи логических функций, образуя сложные условия. В AutoLISP имеется четыре логические функции (табл. 4.1).

Таблица 4.1

Логические функции языка AutoLISP

X	Y	X AND Y	X OR Y	X XOR Y	NOT X
T	NIL	NIL	T	T	NIL
T	T	T	T	NIL	NIL
NIL	T	NIL	T	T	T
NIL	NIL	NIL	NIL	NIL	T

Рассмотрим функцию ***IF***, обеспечивающую ветвление в программе. Ее общий вид: *(IF c f1 [f2])*. Здесь *c* – условие (простое или сложное), *f1* – функция, выполняемая, если условие истинно (часть «то»), а *f2* – функция, выполняемая, когда условие ложно (часть «иначе»), причем квадратные скобки говорят о том, что часть «иначе» может отсутствовать.

Пример

(if (= 1 3) "Да!!" "Нет") – возвращает "Нет"
(if (= 2 (+ 1 1) "Да!!" "Нет") – возвращает "Да!!"

Если нужно в случае выполнения (или невыполнения) условия выполнить не одну, а сразу несколько функций, то в AutoLISP используется *функция* ***(PROGN f1 f2 .. fn)***. Она объединяет функции *f1 f2 ... fn* в один блок, который можно подставить в функцию ***IF***.

Простейший пример: пользователь вводит число, если оно меньше 5, то необходимо построить круг и эллипс, иначе вписанный многоугольник с заданным числом сторон.

Пример

```
(defun oper_IF ()
  (setvar "cmdecho" 0)                    ;отключение эха
  (setvar "osmode" 0)                     ;отключение объектной привязки
  (command "_.erase" "_all" "")                  ;очистка экрана
  (setq pnt (getpoint "\nУкажите центр фигуры"))
  (setq dstn (getdist pnt "\nУкажите начальный радиус"))
  (setq what (getint "\nУкажите число"))

  (if
    (< what 5);условие
;;этот блок выполняется, если условие истинно
     (progn
       (command "_.circle" pnt dstn )
       (command    "_.ellipse"pnt(polar pnt 0 dstn) (/ dstn 5))
)
;а этот, если ложно
  (command "_.polygon" what pnt "_i" dstn)
  )
)
```

Функция ***COND*** обеспечивает множественное ветвление аналогично паскалевскому оператору **CASE**. Ее общий вид:

(COND
(c1 f11 f12 ... f1n_1)
(c2 f21 f22 ... f2n_2)
...
(cm fm1 fm2 ... fmn$_m$)
)

Здесь *c1 ...cm* – логические условия, *fmn* – функции, выполняемые при выполнении того или иного условия. Причем условия проверяются последовательно до первого истинного. Если истинно сразу

несколько условий, то выполняются только функции, относящиеся к первому из них, а остальные условия даже не проверяются.

Пример

```
(setq s (getstring "\nБыть или не быть?"))
(cond ((= s "Д") 1)
((= s "д") 1)
((= s "Н") 0)
((= s "н") 0)
(t nil))
```

Пример

```
(COND (( > A 0) (SETQ B "PLUS"))
((= A 0) (SETQ B "ZERO"))
((< A 0) (SETQ B "MINUS"))
(T NIL)
)
```

4.2. Управляющие конструкции AutoLISP – циклы

Цикл выполняет двоякую функцию:

– выполнение одного и того же участка программы более одного раза;

– передача управления (под управлением понимается выполняемый в данный момент элемент программы) «назад» или «вверх» (т. е. ближе к началу программы).

Циклы в AutoLISP обеспечиваются функциями. Простейшая из них – функция ***(REPEAT n f1 f2 ... fm)***. Здесь *n* – число повторений, а *f1...fm* – те функции, которые будут выполняться *n* раз.

Назначение цикла ***REPEAT*** – в основном решение задач отрисовки.

Пример

```
(defun oper_repeat ()
  (setvar "cmdecho" 0)                ;отключение эха
  (setvar "osmode" 0)                 ;отключение объектной привязки
  (command "_.erase" "_all" "")              ;очистка экрана
  (setq pnt (getpoint "\nУкажите центр фигуры "))
  (setq dstn (getdist pnt "\nУкажите начальный радиус "))
;;;цикл
```

```
  (repeat 10
;число повторений цикла (положительная целая константа)
    ;тело цикла
     (command "_.circle" pnt dstn )
     (command "_.polygon" "4" pnt "_i" dstn)
     (setq dstn (/ dstn (sqrt 2)));конец тела
  );конец цикла
)
```

В AutoLISP возможен цикл по условию, который задается функцией ***(WHILE c f1 f2 ... fm)***. Здесь *c* – логическое условие. Цикл выполняется, пока это условие истинно.

В качестве примера рассмотрим функцию рисования n-параллельных линий.

Пример

```
(defun npar ()
(setq nl (getint "\nЧисло линий: "))
(setq pnt1 (getpoint "\nВведите первую точку: "))
(setq pnt2 (getpoint "\nВведите вторую точку: "))
(setq a (angle pnt1 pnt2))
(command "_.line" pnt1 pnt2 "")
(setq n 1)
(setq d (getreal "\nВведите смещение "))
(while (< n nl)
(setq pnt3 (polar pnt1 (+ a ( / pi 2)) d))
(setq pnt4 (polar pnt2 (+ a (/ pi 2)) d))
(command "_.line" pnt3 pnt4 "")
(setq n (+ 1 n))
(setq pnt1 pnt3
pnt2 pnt4)
))
```

В AutoLISP для обработки каждого элемента списка предусмотрены функции ***FOREACH***, ***MAPCAR*** и ***LAMBDA*** (они были рассмотрены в теме 3).

4.3. Функции работы со строками

Функции работы со строками обеспечивают работу со строковыми переменными, выполняя их различные преобразования.

Функция ***(strcase <строка> <признак>)*** возвращает копию<строки>, переведя все символы алфавита в верхний или нижний регистр в зависимости от аргумента <признак>. Если <признак> опущен или равен *nil*, то все символы в <строке> будут переведены в верхний регистр. Если <признак> присутствует и не *nil*, все символы в строке будут переведены в нижний регистр.

Пример

(*strcase "Пример")* возвращает "ПРИМЕР"
(strcase "АБРАКАДАБРА" T) возвращает "абракадабра"

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

Пример

(setq S1 "АВТОКАД ")
(setq S2 "2004")
(setq S3 "2007")
(setq S4 (strcat S1 S2)) возвращает "АВТОКАД 2004"
(setq S5 (strcat S1 S3)) возвращает "АВТОКАД 2007"

Функция ***(strlen <строка>...)*** возвращает длину строковых констант (в символах), как целую величину.

Пример

(strlen "скажи-ка - дядя...") возвращает 16.

Функция *(substr <строка> <начало> <длина>)* возвращает подстроку <строки>, начинающуюся с символа, номер которого указан в аргументе <начало> и содержащую число символов, заданное в аргументе <длина>. Если длина не указана, то подстрока продолжается до конца <строки>. Первый символ строки имеет номер 1 (у списка первый элемент имеет номер 0).

Пример

(substr "international" 6) возвращает "national".

4.4. Функции для изменения типа данных

В AutoLISP имеются довольно богатые возможности для получения данных одного типа из данных другого типа. Эти возможности обеспечиваются с помощью следующих функций.

Функция ***(ANGTOS <угол> [< представление > [<точность>]])*** преобразует <угол>, выраженный в виде действительного числа в радианах, в форму текстовой строки. Аргумент < представление > указывает, в каком формате должен быть представлен угол:

0 – градусы,
1 – градусы/минуты/секунды,
2 – грады,
3 – радианы,
4 – топографические единицы ("N", "W", "S". "O").

Аргумент <точность> – это целое число, задающее число знаков после запятой. Если аргументы не указаны, то используются значения переменных AutoCAD: **AUNITS** и **AUPREC**. Если значение угла отрицательное, то выполняется преобразование в эквивалентный положительный угол в пределах 0...2π. Таким образом, в текстовой части чертежа может быть записан угол, взятый с чертежа.

Функция ***(ASCII < текст >)*** преобразует текстовую переменную <текст> следующим образом: выделяет первый символ и возвращает соответствующий ему ASCII-код (т. е. целое число). Функция может оказаться полезной, если при вводе текста с клавиатуры нажимают на такие «несимвольные» клавиши, как <Enter>. Сигналы от таких клавиш могут быть отслежены в программе только по их кодам.

Функция ***(CHR < число>)*** выполняет преобразование, обратное ASCII: преобразует <число> в символьный ASCII-код и возвращает соответствующий символ в форме текстовой константы. Число должно быть целым, и его значение должно соответствовать коду какой-либо клавиши.

Функция может быть использована для формирования в программе последовательности символов, эквивалентных последовательности нажатия клавиш, например, последовательности управляющей графическим дисплеем через ANSI-драйвер MS-DOS.

Примеры

(ASCII "A") – возвращает 65
(CHR 65) – возвращает "A"
(ASCII "B") – возвращает 66
(CHR 66) – возвращает "B"
(ASCII "ABBA") – возвращает 65
(CHR 27) – эквивалентно нажатию клавиши ESC

Функции ***(ATOF < текст >)*** и ***(ATOI < текст >)*** осуществляют преобразование аргумента, имеющего тип строки текста, в соответственно действительное и целое числа. При этом <текст> должен содержать допустимое преобразование (например, не содержать в своем составе букв, кроме "E").

С помощью этих функций можно превратить данные, считанные из внешнего файла (а эти данные всегда имеют текстовую форму), в числовые значения параметров, используемых для построения изображения.

Примеры

(ATOF "123.7") – возвращает 123.700000
(ATOI "123.7") – возвращает 123
(ATOI "123") – возвращает 123
(ATOF "123") – возвращает 123.000000
(ATOF "1.2E-3") – возвращает 0.001200
(ATOI "A2") – не определено

Функции ***(ITOA < целое число>)*** и ***(RTOS < число> [< режим > [< точность >]])*** осуществляют преобразование из соответственно целого или действительного числа в текст.

Аргумент <режим> может задавать форму представления действительного числа в виде:

1 – < мантисса > E < порядок >;

2 – < целая часть >.< дробная часть >.

Аргумент < точность > задает число знаков после запятой. Если режим и точность не указаны, то используются значения системных переменных AutoCAD: **LUNITS** и **LUPREC**.

Эти функции ***(FIX <число>)*** и ***(FLOAT < число)*** совершают преобразование числового аргумента в соответственно целую и действительную формы. Приведение к целому выполняется путем отбрасывания дробной части (а не округлением).

Тема 5. ИЗУЧЕНИЕ AUTOLISP. РАБОТА С ФАЙЛАМИ

В AutoLISP имеются возможности для обмена данными между программами через магнитные носители, при этом могут быть использованы данные не только из различных программ на AutoLISP, но и из программ на других языках. Например, расчетная программа на языке ФОРТРАН (Паскаль, С и др.) может записать свои результаты на диск, а программа на AutoLISP прочитает эти данные и использует как параметры для построения чертежа.

Файл с данными можно также использовать для «пакетного ввода» исходных данных в программу, вместо организации ввода в режиме диалога. Это целесообразно, когда при каждом запуске программы вводится большой объем данных, изменяющихся незначительно. Тогда проще отредактировать текстовый файл, чем заново набирать эти данные.

5.1. Открытие и закрытие файла

Перед тем как начать работу с файлом, его нужно открыть с помощью функции ***OPEN***:

(OPEN < имя файла > < режим >).

Здесь < *имя файла* > – обозначение открываемого файла. Значение <режима> может быть символами "r", или "w", или "a", где:

"r" – существующий файл открывается для чтения; если такого файла нет, то функция вернет NIL;

"w" – файл открывается для записи; если файла с таким именем нет, то создается новый файл с указанным именем; если такой файл уже существует, то он заменяется новым файлом с тем же именем;

"a" – файл открывается для добавления; если такого файла нет, то он создается; если файл уже существует, то появляется возможность добавления новых записей "в хвост" уже существующим.

Функция ***OPEN*** всегда используется в сочетании с функцией ***SETQ***, так как ***OPEN*** возвращает условное имя файла, используемое внутри программы, которое будем называть *дескриптором файла*. Дескриптор файла необходимо записать с помощью ***SETQ*** в какую-либо переменную, эту переменную затем придется использовать в операциях ввода-вывода.

Таким образом, строка

```
(SETQ F1 (OPEN "\\K1\\FILE1.DAT" "r"))
```

означает, что на рабочем диске в подкаталоге К1 ищется файл *FILE1* с расширением *DAT*, этот файл открывается для чтения, значение дескриптора файла присваивается переменной F1.

После того как работа с данным файлом закончена, его следует закрыть с помощью функции ***CLOSE***:

(CLOSE <дескриптор файла >)

Например, файл F1 должен быть закрыт так: *(CLOSE F1)*.

Итак, любой вид работы с файлом должен «открываться» функцией ***OPEN*** и «закрываться» функцией ***CLOSE***. Например, пусть в программе предусмотрены сброс промежуточных данных на магнитный диск, а затем использование этих данных в следующей части программы. Это может быть организовано следующим образом:

```
……..
(SETQ F (OPEN "FD.DAT" "w"))
……..
;(В том числе операции записи в файл)
(CLOSE F)
……..
(SETQ F (OPEN "FD.DAT" "r"))
……..
;(В том числе операции чтения из файла)
(CLOSE F)
```

5.2. Чтение из файла

Функция *(findfile <имя-файла>)* позволяет отыскивать файл, указанный в <имя файла>, в каталогах, хранящих файлы AutoCAD, и возвращает полное имя файла или *nil*, если файл не найден. Каталоги просматриваются в следующем порядке:

1. Текущий каталог.
2. Каталог, в котором хранится текущий рисунок.
3. Каталоги, имена которых заданы в переменной среды AutoCAD (если заданы).
4. Каталог, в котором хранятся программные файлы AutoCAD.

Если <имя-файла> содержит имя дисковода и каталогов (путь доступа), AutoCAD просматривает только указанный каталог.

Текущий каталог – это каталог, в котором сохранен файл чертежа, с которым вы работаете. 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 к файлу, который уже можно использовать в программе для открытия файла или *nil* – свидетельствует об отсутствии упомянутого файла.

Пусть текущий каталог c:/acad и в нем хранится файл abc.lsp. Тогда *(findfile "abc.lsp")* возвращает "/acad/abc.lsp"

```
;строка для открытия существующего файла
(setq f (open (findfile "filename") "r"))
```

Поэтому перед открытием файла для чтения необходимо проверять его наличие.

```
(if (/= (findfile "filename") nil)
(prong
(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 простой функцией ***(read-line <описатель файла>)***, которая считывает и возвращает строку символов с клавиатуры или из открытого файла, заданного <описателем файла>. Если чтение невозможно, то возвращается *nil*. С выполнением каждой функции ***read-line***, производится переход на следующую строку в файле.

Функция ***(read-char <описатель файла>)*** считывает единичный символ из буфера ввода клавиатуры или из открытого файла, задаваемого <описателем файла>. Возвращает ASCII-код считываемого символа. Если не задан <описатель файла> и в буфере ввода клавиатуры нет символов, *(read-char)* ждет от пользователя ввода с клавиатуры (заканчивающегося нажатием клавиши ввода <Enter>).

В результате мы получим некое строковое выражение. Его можно обработать с помощью функций ***ATOI*** или ***ATOF*** для получения целого или вещественного значений соответственно.

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

Если строка сложная, то сначала придется разделить ее на составляющие части. Разделить строку на подстроки можно с помощью функции ***SUBSTR***.

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

После отделения необходимой части строки можно ее обработать с помощью тех же функций-обработчиков для получения целых или вещественных значений.

На рис. 5.1 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.

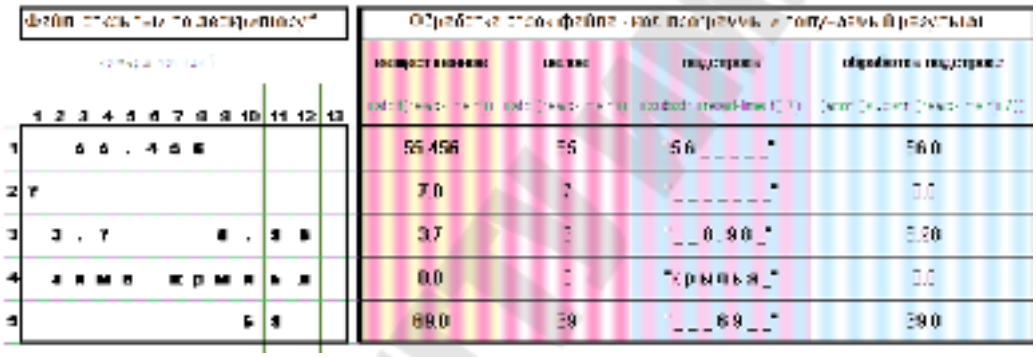


Рис. 5.1. Иллюстрация процесса обработки файла

5.3. Запись в файл

Для записи в текстовый файл наиболее удобно использовать функцию ***(write-line <строка> <описатель файла>)***, которая выводит строковую константу <строка> на экран или в открытый файл, заданный <описателем файла>. Возвращает строку, взятую в кавычки, и отпускает кавычки, когда строка записывается в файл.

Функция ***(write-char <число> <описатель файла>)*** записывает один символ на экран или в открытый файл, заданный <описателем файла>. <число> – это ASCII-код символа, и является значением, возвращаемым функцией.

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 Y)
(defun pt_to_str(pt)
  (strcat "(" (rtos (nth 0 pt)) "," (rtos (nth 1 pt)) ")" )
)
```

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

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

5.4. 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.

5.5. Примеры работы с файлом

Пример 1

На рис. 5.2 представлен листинг программы, которая читает содержимое файла и выводит содержимое построчно на экран.

В этой программе используется функция ***(getfiled title default ext flags)***, которая предлагает пользователю стандартное диалоговое окно для выбора файла и возвращает имя файла. Здесь ***title*** – заголовок диалогового окна; ***default –*** текущий путь для диалогового окна; ***ext*** – расширение файла по умолчанию (“” – означает выбор .* – всех файлов); ***flags*** – флаг контроля диалогового окна.

```
;-------------------------------------------------------------------------------
;....Чтение содержимого файла и вывод на экран
;-------------------------------------------------------------------------------
(defun Example_1 ( / pr fname lst k)
; открытие существующего файла
  (setq
; установка текущего пути
     pr (getvar "DWGPREFIX")
; взятие имени файла
     fname (getfiled "Укажите файл" pr "" 4)
; функция чтения файла и записи элементов в список
     lst (FilToList fname)
; счетчик
     k 0)
; цикл, пока не кончится файл
  (while (< k (length lst))
; печать элемента списка
     (print (nth k lst))
; наращивание счетчика
     (setq k (+ k 1))
  )
)
;-------------------------------------------------------------------------------
;....Запись содержимого файла в список
;-------------------------------------------------------------------------------
;....strfrom - строка с именем файла
;-------------------------------------
(defun FilToList (strfrom / f fk a aa)
; проверка существования файла
  (setq f (findfile strfrom))
; если существует
  (if (/= f nil)
     (progn
; открытие существующего файла
        (setq fk (open strfrom "r")
           aa(list)
           a (read-line fk))
; цикл с проверкой конца файла
        (while (/= a nil)
; добавление элемента в список
```

Рис. 5.2. Листинг программы «Чтение и вывод на экран содержимого файла» (1-й фрагмент; окончание см. на с. 47)

```
        (setq aa (append aa (list a)))
; чтение следующего
        (setq a (read-line fk))
      )
; закрытие файла
      (close fk)
    )
  )
; возвращение данных в основную программу
  aa)
```

Рис 5.2. Окончание (начало см. на с. 46)

Пример 2

На рис. 5.3 представлен листинг программы чтения данных из файла и отображение их с заданной высотой букв, под заданным углом в указанную точку графического поля рисунка. Имя файла и путь доступа к нему указывается в командной строке.

```
;*************************************************************
;
;Функция размещения текста из файла на чертеже
;*************************************************************
;
(DEFUN TEXTIN1 ()
  (SETVAR "CMDECHO" 0)
  ;(SETVAR "TEXTSCR")
  (SETQ  FN (GETSTRING "\n Введите имя файла с текстом:" )
         BP (GETPOINT "\n Введите точку ввода текста : [можно мышкой]")
         P  (GETREAL "\n Введите высоту букв текста:" )
         U  (GETREAL "\n Введите угол наклона текста:"))
  (TYPE FN)
  (PRINT FN)
 (setq f (findfile fn))
; если существует
   (if (/= f nil)
     (progn
  (SETQ OF (OPEN FN "r")
         H  (/ P 4)
         RL (READ-LINE OF ))
  (COMMAND "_.style" "" "" P H "" "" "" "")
 (COMMAND "_.text" BP U RL)
  (WHILE (SETQ RL (READ-LINE OF ))
(SETQ BP (LIST (CAR BP) (- (CADR BP) 10)))
    (COMMAND "_.text" BP U RL)
   )
  (CLOSE OF)
  )
    ))
```

Рис. 5.3. Листинг программы
«Вывод информации из файла на графическое поле рисунка»

Тема 6. ПРОГРАММИРОВАНИЕ ДИАЛОГОВЫХ ОКОН

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 управляются *атрибутами*. Размер диалогового окна и размещение частей установлены автоматически с минимумом информации.

6.1. Структура диалогового окна

Диалоговые окна определяются в текстовых файлах с соответствующим расширением *.DCL. Управление 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 (рис. 6.1) показывает структуру файла DCL.

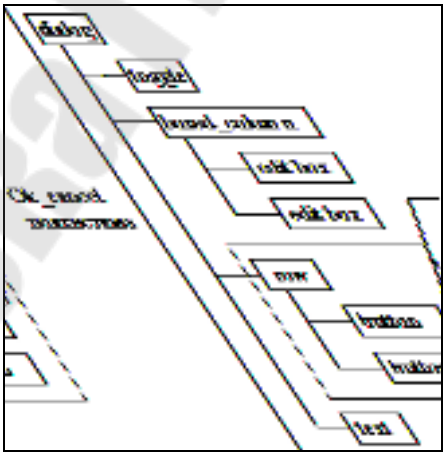


Рис. 6.1. Структура файла DCL

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 т. д.

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, в колонку.

6.2. Визуализация/отображение диалоговых окон

В среде VisualLISP откройте/создайте файл с расширением .DCL. Выберите *Сервис/Иструменты интерфейса/Просмотр DCL из редактора*, чтобы отобразить диалоговое окно, определенное в текстовом редакторе.

Если ввести простейший код рис. 6.2 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;
 }
}
```

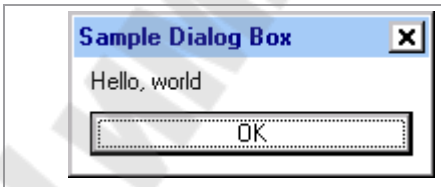


Рис. 6.2. Код DCL файла и результат его запуска

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 равным *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 (рис. 6.3).

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

Рис. 6.3. Модифицированный код DCL-файла

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

Ниже приведены примеры (рис. 6.4) 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;
 }
}
```

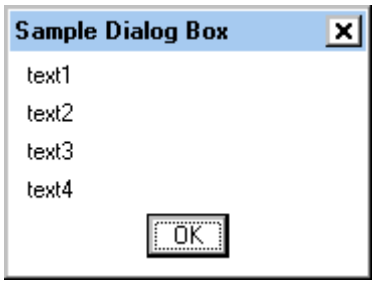
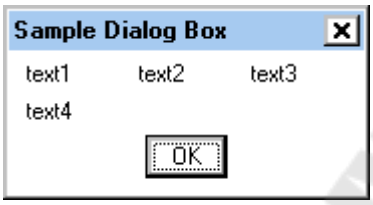


Рис. 6.4. Примеры различного расположения элементов типа текст
(1-й фрагмент; продолжение и окончание см. на с. 52 и 53)

```
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;
 }
}
```




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

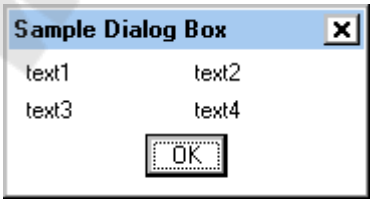


Рис. 6.4. Продолжение (начало см. на с. 51, окончание – на с. 52)

```
hello : dialog {
 label = "Sample Dialog Box";
 : row {
  :column {
   : 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;
 }
}
```

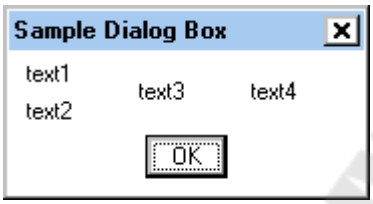


Рис. 6.4. Окончание (начало см. на с. 51 и 52)

6.3. Программное описание диалоговых окон

Для описания диалога необходим файл-описание *.dcl и файл-программа для управления диалогом. Файл DCL должен лежать в доступном для AutoCAD каталоге, например, рядом с чертежом.

Для работы диалога необходимо:

1. Создать функцию.
2. Внутри функции определить идентификатор диалога *load_dialog*.
3. Проверить, есть ли такой диалог *new_dialog*.
4. Определить действия, связанные с ключами *action_tile*.
5. Дальше – диалог стартует *start_dialog*.
6. С помощью функции *cond* определяете выбранный пользователем вариант.
7. И после этого диалог выгружаем из памяти *unload_dialog*.

Самый простой пример – *обработка диалога с кнопками*.

Рассмотрим подробно пример:

1. Запуск функции работы с диалоговым окном.

```
(defun dd_btn ( / dcl_id what_next)
```

2. Функция позволяет загрузить диалоги из указанного файла в память.

```
; загрузка диалога
(setq dcl_id (load_dialog "ot_tab.dcl"))
```

3. Если среди новых диалогов нет того, который нам необходим, то программа завершает работу.

```
; инициализация диалога - проверка существования
(if (not (new_dialog "dd_button" dcl_id))
(exit)
)
```

4. Функции *action_tile* описывает действия, которые должны происходить при выборе того или иного элемента управления. Элемент управления определяется по *"key"*. Действия записываются в кавычках.

```
; Две кнопки = два действия, по первой кнопке результат - 1
(action_tile "btn1" "(done_dialog 1)")
; По второй - 2
(action_tile "btn2" "(done_dialog 2)")
```

5. В данном случае диалог возвращает 1 или 2 в зависимости от нажатой кнопки. Следующая строка программы запускает диалог, результат которого будет записываться в переменную what_next.

```
; запуск диалога - результат запишется в what_next
(setq what_next (start_dialog))
```

6. Анализ результата работы диалога

```
(cond
((= what_next 2) (alert "Запуск по кнопке 2!"))
((= what_next 1) (alert "Запуск по кнопке 1!"))
)
```

7. Выгружаем из памяти диалоги

```
(unload_dialog dcl_id)
)
(prompt "dd_btn ")
```

6.4. Проверка работы примеров диалогов с различными элементами

Для тестирования примеров, показанных ниже, необходимо:

– создать два файла в среде VisualLisp: один для диалогов, другой для программы обработки событий диалога;

– назвать файл с описанием лучше как в примере: ot_tab*n*.dcl, т. к. к такому имени будет обращаться программа-обработчик. Файл с программой можно назвать как угодно, *.lsp;

– прописать путь к каталогу с файлом DCL, чтобы он был виден для программы диалога (*Сервис/Настройка/Файлы/Путь доступа к вспомогательным файлам*);

– скопировать DCL-окно в файл ot_tab.dcl, а LSP-окно в файл программы;

– сохранить файлы;

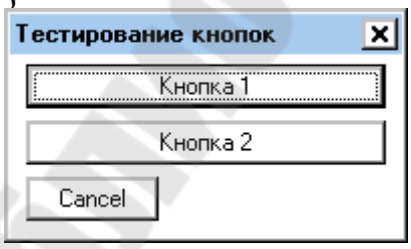
– загрузить программу;

– запустить программу в AutoCAD.

Кнопки (рис. 6.5)

DCL

```
// Описание диалога
//для панели с кнопками
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 = cen-
ter; label = "Кнопка 2"; }
: row {
cancel_button;
}
}
```



LISP

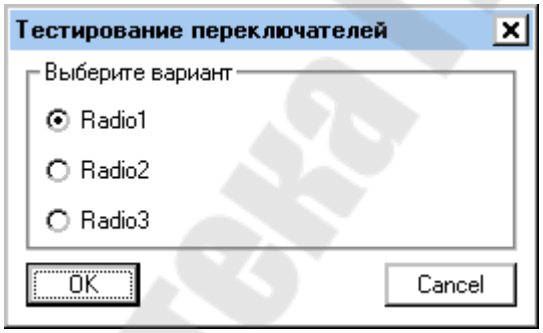
```
(defun dd_btn ( / dcl_id what_next)
; загрузка диалога
  (setq dcl_id (load_dialog "ot_tab1.dcl"))
; проверка существования диалога, если не
;существует - выход
  (if (not (new_dialog "dd_button" dcl_id))
(exit))
; Две кнопки = два действия, по первой
;кнопке результат - 1
  (action_tile "btn1" "(done_dialog 1)")
; По второй - 2
  (action_tile "btn2" "(done_dialog 2)")
; запуск диалога - результат запишется в
; what_next
  (setq what_next (start_dialog))
; проверка результатов работы
  (cond
    ((= what_next 2) (alert "Запуск по кнопке
2!"))
    ((= what_next 1) (alert "Запуск по кнопке
1!"))
  )
; выгрузка диалога из памяти
  (unload_dialog dcl_id)
)
```

Рис. 6.5. Описание диалога для панели с кнопками и лисп-программа

Переключатели

Следующий пример отличается тем, что используется цикл. Для того, чтобы дать возможность пользователю кликать/щелкать переключателями сколько 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 (рис. 6.6).

```
// Переключатели
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;
}
}
```



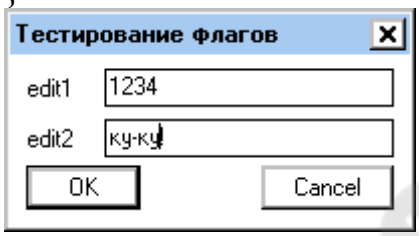

```
(defun dd_radio ( / ret_value1 dcl_id what_next on_rad)
; функция, вызываемая по OK
;(формирование списка данных на
;выход)
  (defun ok_tab ( / )
    (setq ret_value1 (list "radio" on_rad))
  )
; загрузка файла-диалога
  (setq dcl_id (load_dialog "ot_tab2.dcl"))
; проверка существования диалога,
;если не существует - выход
  (if (not (new_dialog "dd_radio" dcl_id)) (exit))
; начальные установки переменных
;и элементов
  (set_tile "radio1" "1")
  (setq on_rad "radio1")
  (setq what_next 8)
; цикл
  (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))
  )
; выгрузка диалога из памяти
  (unload_dialog dcl_id) ; Unload the DCL file
  ret_value1
)
```

Рис. 6.6. Описание диалога с переключателями и лисп-программа

Поля

В примере с полями особенностью является задание значений полей заранее – в тексте программы (рис. 6.7).

```
// Диалог с полями для за-
//полнения
dd_edit : dialog {
label = "Тестирование фла-
гов";
fixed_height = true;
: edit_box { key = "edit1";
value = "0"; fixed_height =
true; alignment = center; la-
bel = "edit1"; }
: edit_box { key = "edit2";
value = "0"; fixed_height =
true; alignment = center; la-
bel = "edit2"; }
: row {
ok_button;
cancel_button;
}
}
```




```
(defun dd_edit ( / ret_value1 dcl_id what_next
on_ed2 on_ed1)
; функция, вызываемая по OK (формирование
;списка данных на выход)
  (defun ok_tab ( / )
    (setq ret_value1 (list (list "ed1" on_ed1) (list "ed2"
on_ed2)))
  )
; загрузка диалога
  (setq dcl_id (load_dialog "ot_tab3.dcl"))
; проверка существования диалога, если не суще
;ствует - выход
  (if (not (new_dialog "dd_edit" dcl_id))(exit))
; начальные установки переменных и элементов
  (set_tile "edit2" "ку-ку!")
  (set_tile "edit1" "1234")
  (setq on_ed1 "1234" on_ed2 "ку-ку!")
  (setq what_next 8)
; цикл
  (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))
  )
; выгрузка диалога из памяти
  (unload_dialog dcl_id)
  ret_value1
)
```

Рис. 6.7. Диалог с полями для заполнения и лисп-программа

Флаги

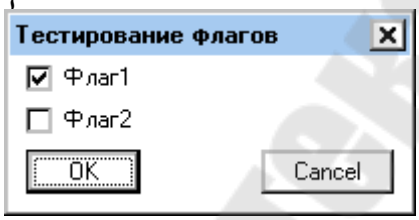
Рассматривая пример с флагами, необходимо отметить, что в данном случае мы инициализируем заранее значения. Функция *on_type_os* изменяет элементы окна на заданные в параметрах функции. В зависимости от параметров с помощью оператора *set_tile* устанавливаем значения флагов (рис. 6.8).

```
// 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;
}
}
```




```
;----------------------------------------
; 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)))
  )
; функция выбора включенных-
;выключенных позиций 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)
    )
  )
  (setq dcl_id (load_dialog "ot_tab4.dcl"))
; проверка существования диалога, если не
;существует - выход
  (if (not (new_dialog "dd_toggle"
dcl_id))(exit))
; Инициализация ключей в зависимости от
;введенных параметров
  (on_type_os)
  (setq what_next 8)
; цикл
  (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))
  )
; выгрузка диалога из памяти
  (unload_dialog dcl_id) ; Unload the DCL file
  ret_value1
)
```

Рис. 6.8. Описание диалога для панели с флага и лисп-программа

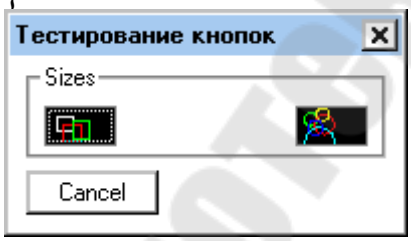
Кнопки с изображением

Необходимо создать слайды, которые будут использоваться на кнопке. Слайды создаются так: рисуете на поле чертежа что-нибудь. Вводите команду _.mslide – вводите имя файла-слайда. Слайд готов. Для отображения в окне AutoCAD необходимо ввести команду _.vslide. Тогда слайд временно выведется на рабочее поле.

Размеры изображения задаются в DCL-файле (рис. 6.9, слева), и еще изображение надо инициализировать в lisp-файле – (рис. 6.9, справа). А обработка производится так же, как и обработка кнопок.

```
// Описание диалога для панели с
//картинками-кнопками
dd_imgbutton : dialog {
label = "Тестирование кнопок";
: 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;
}
}
```



В результате Вы должны получить на экран

```
(defun dd_imgbtn ( / dcl_id what_next)
; загрузка диалога
(setq dcl_id (load_dialog "ot_tab5.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!"))
)
; выгрузка диалога из памяти
(unload_dialog dcl_id)
)
```

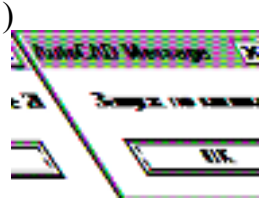
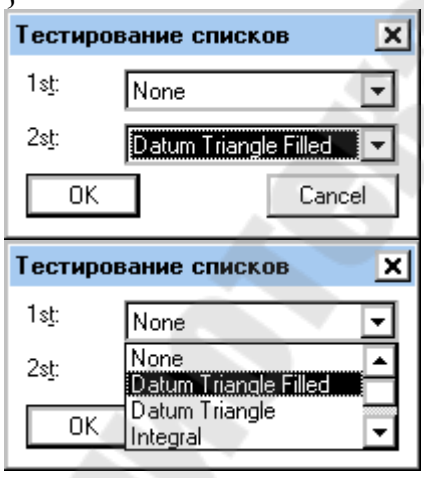


Рис. 6.9. Описание диалога для панели с картинками-кнопками и лисп-программа

Списки

В работе со списками основная сложность – составление самого списка. Он составляется как символьная строка, через знак "\n" – перевод каретки (рис. 6.10).

```
// Диалог со списком
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;
}
}
```




```
(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_tab6.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))
)
; выгрузка диалога из памяти
(unload_dialog dcl_id) ; Unload the DCL file
ret_value1
)
```

Рис. 6.10. Диалог со списком и лисп-программа

6.5. Создание параметрического чертежа

Пример

Разработать параметрический чертеж детали (рис. 6.11).

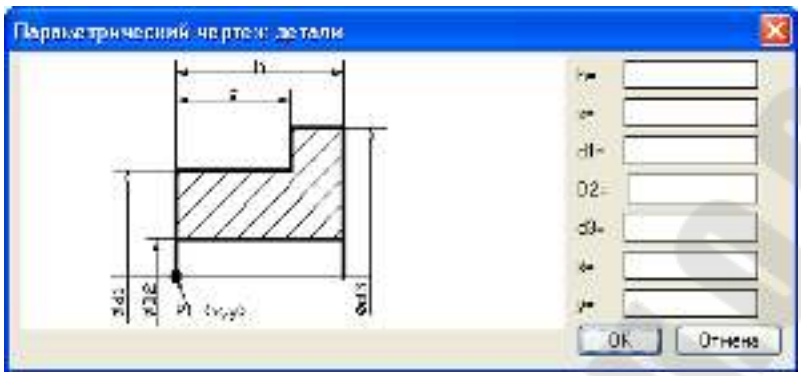


Рис. 6.11. Диалог со списком и лисп-программа

DCL-файл представлен на рис. 6.12.

```
// Диалог параметрическогочертежа детали
dd_param :
dialog {label = "Параметрический чертеж детали";
fixed_height = true;
: row {
:image {
key="ff";
color =0;
fixed_height = true;
fixed_width = true;
aspect_ratio=0.5;
 width=60;
 alignment = center;
 }
:column {
 :column{
 : edit_box { key = "edit1";
value = "0";
fixed_height = true;
fixed_width = true ;
width =20 ;
label = "h=  ";
}
: edit_box { key = "edit2";
value = "0";
fixed_height = true;
fixed_width = true ;
width = 20 ;
```

Рис. 6.12. Код DCL-файла (1-й фрагмент; окончание см. на с. 61)

```
label = "s=  ";
}
: edit_box { key = "edit3";
value = "0";
fixed_height = true;
fixed_width = true ;
width = 20 ;
label = "d1=";
}
: edit_box { key = "edit4";
value = "0";
fixed_height = true;
fixed_width = true ;
width = 20 ;
label = "D2=";
}
: edit_box { key = "edit5";
value = "0";
fixed_height = true;
fixed_width = true ;
width = 20 ;
label = "d3=";
}
 : edit_box { key = "edit6";
value = "0";
fixed_height = true;
fixed_width = true ;
width =20 ;
label = "x=  ";
}
 : edit_box { key = "edit7";
value = "0";
fixed_height = true;
fixed_width = true ;
width =20 ;
label = "y=  ";
}
}
: row {
ok_button;
cancel_button;
}
}
}
}
```

Рис. 6.12. Окончание (начало см. на с. 60)

На рис. 6.13 показаны ключевые точки для прорисовки детали, а на рис. 6.14 представлена лисп-программа.

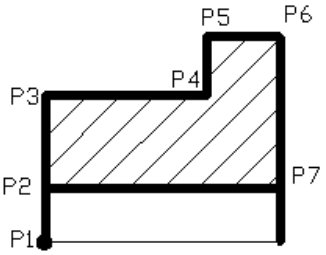


Рис. 6.13. Эскиз детали

```
; функция, вызываемая по OK (формирование списка данных на выход)
 (defun ok_tab ( / )
   (setq h (atof on_ed1)
         s (atof on_ed2)
         d1 (atof on_ed3)
         D2 (atof on_ed4)
         d3 (atof on_ed5)
         x (atof on_ed6)
         y (atof on_ed7)
         p1 (list x y)
))
;-------------------------------------------------------
; 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 "ТИПЛИН" "УСТАНОВИТЬ" "CONTINUOUS"  "")
 (command "Цвет" 4) ; смена цвета (для выделения толстых линий)
  (command "_.pline" p2 "Ш" "1" "" p3       p4 p5 p6 p7 "З")
```

Рис. 6.14. Лисп-программа
(1-й фрагмент; продолжение и окончание см. на с. 63 и 64)

```
; забегая вперед, это - сохранение нарисованного примитива
  (setq det (entlast))
; отражение нарисованной детали
  (command "_.mirror" det "" p1 "@1,0" "H")
; рисование
  (command "_.pline" p2 (list x (- y (/ d2 2.0))) "")
  (command "_.pline" p7 (list (+ x h) (- y (/ d2 2.0))) "")
; установка красного цвета
; осевой линии
(COMMAND "ТИПЛИН" "УСТАНОВИТЬ" "ОСЕВАЯ2"  "")
 (COMMAND "ЦВЕТ" 1 )
 (command "_.line" (list (- x 10) y) (list (+ x h 10) y)"")
; штриховка
 (COMMAND "ТИПЛИН" "УСТАНОВИТЬ" "CONTINUOUS"  "")
  (command "_.bhatch" "c" "ansi31"
    (/ (+ d3 h) 100.0) ; расчет относительного масштаба штриховки
    "0" ; угол наклона
    (list (+ x (/ h 10.0)) (+ y (/ d2 2.0) (/ (- d1 d2) 4.0))) ; точки внутри шртихуемой об-
ласти
    (list (+ x (/ h 10.0)) (- y (/ d2 2.0) (/ (- d1 d2) 4.0))) ; точки внутри шртихуемой об-
ласти
    ""
  ))
(defun prim_param ( / ret_value1 dcl_id1 what_next on_ed2 on_ed1)
 ( SETQ flag T )
( WHILE flag
; удаляем все объекты
 (command "_.erase" "_all" "")
  ; загрузка диалога
 (setq dcl_id1 (load_dialog "dd_param1.dcl"))
 ; проверка существования диалога, если не существует - выход
 (if (not (new_dialog "dd_param" dcl_id1))(exit))
 ; начальные установки переменных и элементов
 (set_tile "edit2" "")
 (set_tile "edit1" "")
 (set_tile "edit3" "")
 (set_tile "edit4" "")
 (set_tile "edit5" "")
 (set_tile "edit6" "")
 (set_tile "edit7" "")
 (setq on_ed1 ""
       on_ed2 ""
       on_ed3 ""
       on_ed4 ""
       on_ed5 ""
       on_ed6 ""
       on_ed7 ""
```

Рис. 6.14. Продолжение (начало см. на с. 62, окончание – на с. 64)

```
        )
( SETQ flag1 T )
 ( WHILE flag1
 (start_image "ff")
 (slide_image 0 0  (dimx_tile "ff") (dimy_tile "ff")  "param" )
 (end_image)
 (setq what_next 8)
 ; цикл
 (while (< 2 what_next)
   (action_tile "edit1" "(setq on_ed1 $value)")
   (action_tile "edit2" "(setq on_ed2 $value)")
     (action_tile "edit3" "(setq on_ed3 $value)")
     (action_tile "edit4" "(setq on_ed4 $value)")
     (action_tile "edit5" "(setq on_ed5 $value)")
     (action_tile "edit6" "(setq on_ed6 $value)")
     (action_tile "edit7" "(setq on_ed7 $value)")
   (action_tile "accept" "(done_dialog 1) (ok_tab)")
   (setq what_next (start_dialog))
 )
(if ( or ( <= d1 d2)  ( <= d3 d1 )  ( <= h s) )
(progn
 (alert "Неверно заданы параметры" )
    (new_dialog "dd_param" dcl_id1)
 (set_tile "edit1" on_ed1)
 (set_tile "edit2" on_ed2)
  (set_tile "edit3" on_ed3)
  (set_tile "edit4" on_ed4)
  (set_tile "edit5" on_ed5)
  (set_tile "edit6" on_ed6)
  (set_tile "edit7" on_ed7)
 )
  ( progn
        (setq flag1 nil)
; рисование детали - запуск параметрического чертежа
    (DrawDetail p1 d1 d2 d3 s h)
 )   ) )
 ( SETQ ans ( GETSTRING "\n Повторить<Д/Н>: " ) )
  ( SETQ flag ( OR ( = ans "Д" ) ( = ans "д" ) ) )
 ); конец WHILE
  ; выгрузка диалога из памяти
 (unload_dialog dcl_id1)
 ( PRIN1 )
   )
```

Рис. 6.14. Окончание (начало см. на с. 62, 63)

Тема 7. АВТОМАТИЗАЦИЯ РАЗРАБОТКИ И ВЫПОЛНЕНИЯ КОНСТРУКТОРСКОЙ ДОКУМЕНТАЦИИ В СРЕДЕ AUTOCAD

7.1. Формирование чертежей с использованием вариантного метода

Существует множество методов формирования конструкторской документации в среде графических систем. Рассмотрим метод пооперационной обработки заготовки детали. Его особенность заключается в том, что способов механической обработки заготовки для превращение ее в изделие относительно мало. Многократно используя эту технологическую операцию, можно получить чертежи деталей очень широкого класса. Более того, если все способы обработки независимы друг от друга, то, добавив даже одну новую операцию, вы значительно расширите разнообразие получаемых чертежей.

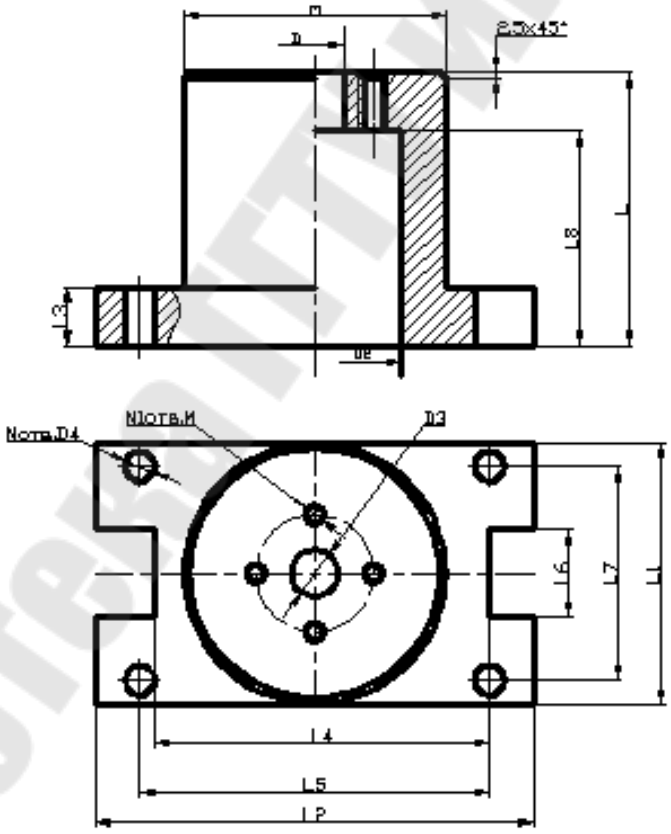


Рис. 7.1. Параметрический типовой чертеж комплексной детали

Чтобы создать набор формирования чертежей определенного класса деталей, сначала нужно выбрать из уже имеющихся рисунков наиболее сложные и полно отражающие все особенности дан-

ного класса. Далее на их основе разрабатывается чертеж типовой детали. Все его размеры должны быть выражены в параметрах. С такого параметрического типового чертежа детали и начинает свою работу программист. До начала разработки программного обеспечения необходимо выделить в этом чертеже *основу детали* и *функциональные элементы.*

Основа любой детали – это заготовка, из которой с помощью последующей обработки (сверления, точения, фрезерования и пр.) получается требуемое изделие. В принципе все основы можно представить как заготовку в форме либо цилиндра, либо параллелепипеда без отверстий, однако на практике заготовки бывают более сложными по форме и в некоторых случаях – со сквозными отверстиями.

Функциональный элемент, с точки зрения разработчика программного обеспечения – это одна параметрическая обработка заготовки. При обработке модели заготовки необходимо корректно модифицировать весь ее чертеж. Отсюда некоторая двойственность термина «функциональный элемент», с одной стороны, это технологическая операция над деталью заготовкой, а с другой – программа, модифицирующая чертеж заготовки. Для пользователя функциональный элемент – это программа или команда, модифицирующая чертеж заготовки в полном соответствии с некоторой технологической обработкой детали-заготовки. Если для параметрического прототипа, приведенного на рис. 7.1, принять за основу рис. 7.2, то кроме основы можно выделить 3–4 функциональных элемента: круговую систему отверстий, резьбовое отверстие с проточкой, цилиндрическую расточку и др.

Все функциональные элементы должны быть независимы друг от друга, то есть иметь возможность выполняться в любой последовательности и любое количество раз, если это не противоречит корректному осуществлению соответствующих им операций над деталью заготовки. При разработке программ, формирующих функциональные элементы, следует иметь в виду, что они обязательно привязаны к *базовым точкам* заготовки. Эти точки обычно выбираются либо на пересечении осей проекции, либо на пересечении оси и какой-либо поверхности, но не более одной на каждой проекции заготовки. На рис. 7.3 и 7.4 показаны некоторые функциональные элементы для комплексной детали, изображенной на рис. 7.1.

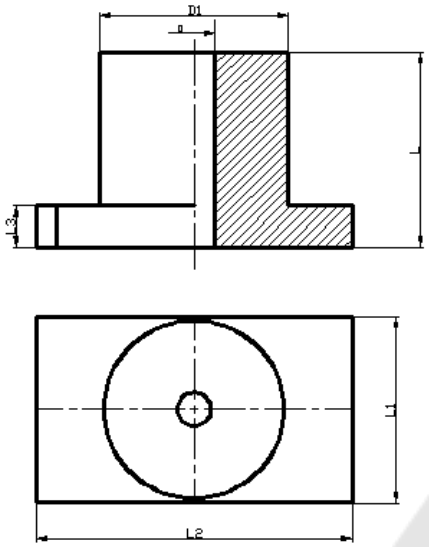


Рис. 7.2. Параметрический чертеж основы комплексной детали

На AutoLISP вызов программы и ввод входных параметров можно оформить двумя способами: в виде функции или в виде команды. У каждого способа есть свои преимущества и недостатки, но с точки зрения пользователя, более удобен вызов в виде команды с вводом данных в процессе диалога. Более того, если программу формирования основы еще можно описать в виде функции, то функциональный элемент – только в виде команды, включающей развитый диалог с пользователем. Диалог необходим по двум причинам: во-первых, ввод координат базовых точек в большинстве случаев возможен только с помощью объектных привязок в интерактивном режиме, а во-вторых, ввод значений параметров функциональных элементов, определяемых элементами заготовки, требует измерения непосредственно на чертеже.

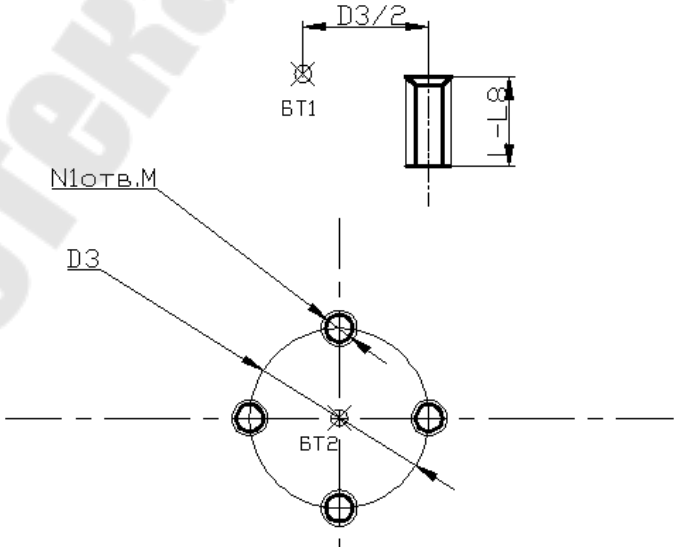


Рис. 7.3. Функциональный элемент – резьбовые отверстия

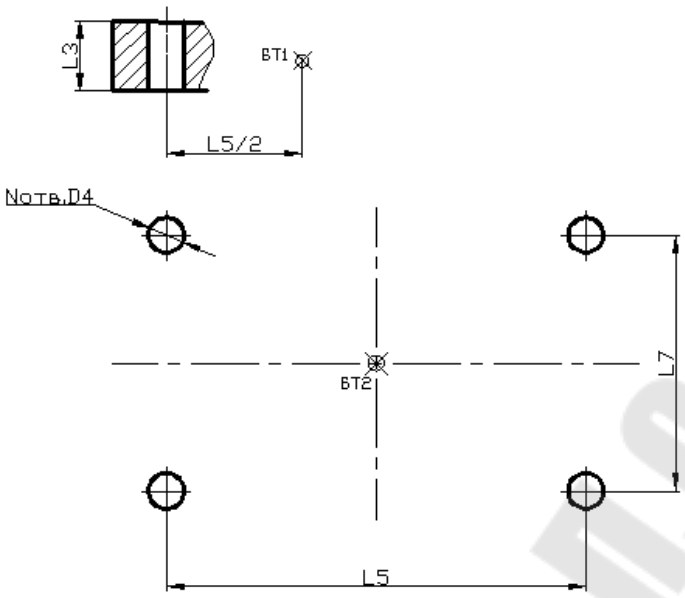


Рис. 7.4. Функциональный элемент – прямоугольная система отверстий

При создании сценария диалога следует придерживаться некоторых правил:

– осуществлять ввод только минимально достаточной информации. Например, на рис. 7.3 размер функционального элемента определен через разность двух размеров прототипа. Ошибочно требовать от пользователя ввода обоих размеров, чтобы получить их разность; правильно сразу запросить ее (L–L8). Кроме ввода значений параметров, определяющих геометрию функционального элемента, необходим ввод не менее одной базовой точки. В языке AutoLISP это реализуется либо путем ввода координат точки в параметрах разрабатываемой модели, либо при помощи функции GETPOINT с последующим извлечением данных координат. Для основы необходима одна базовая точка, определяющая ее местоположение на чертеже, а для функционального элемента – столько, на скольких проекциях модифицируется изображение заготовки;

– по возможности вводить информацию различными способами. В AutoLISP это означает, что, например, ввод значения параметра D лучше осуществлять с помощью функции GETDIST. В отличие от функции GETREAL, GETDIST позволяет вводить значения и с клавиатуры (указывая число), и с помощью мыши (задавая расстояние между двумя выбранными на экране точками). Разнообразие способов ввода данных значительно облегчает работу пользователя с программой;

– не запрашивать и не вводить информацию, которую можно получить из чертежа, например, при формировании цилиндрической расточки по отверстию (рис. 7.5), данный функциональный элемент определяют значения параметров L8, D2 и базовой точки. Для удаления части линии отверстия программисту необходимо знать также его диаметр D. Кажется соблазнительным просто запросить это значение у пользователя, но оно не относится к данному функциональному элементу. Решить проблему позволяет сканирование определенной области чертежа и анализ тех объектов, которые попали в эту область. В AutoLISP сканирование осуществляется с помощью функции SSGET, обычно с ключом Crossing.

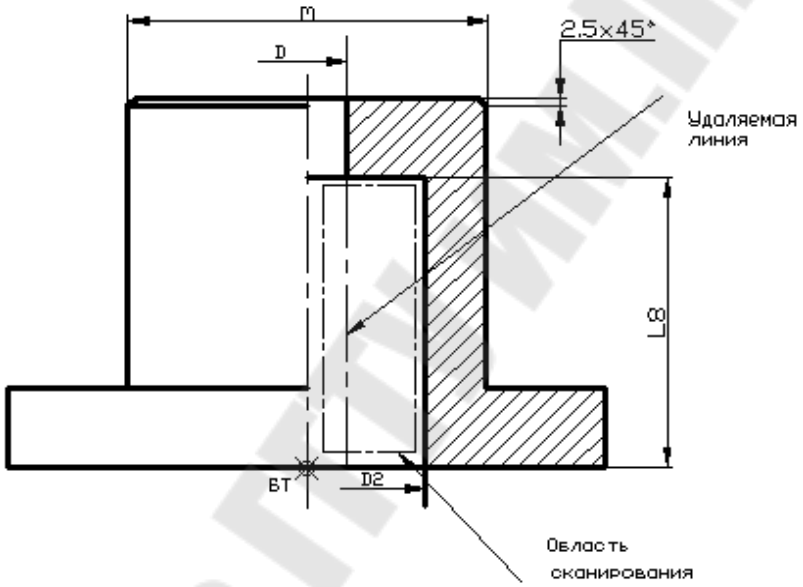


Рис. 7.5. Модификация основы детали
при установке функционального элемента цилиндрической расточки

Разработчик программного обеспечения должен учитывать независимость функциональных элементов. Применительно к командам редактирования AutoCAD это означает, что при выполнении любой разработанной команды не важно, каким образом и в какой последовательности обрабатывается графическое изображение. Независимость функциональных элементов означает, что они должны модифицировать любой чертеж заготовки без учета того, в какой последовательности и с помощью каких средств формировался чертеж. Если разработчик не придерживался данного принципа, то функциональный элемент разработан не корректно.

Ниже приведены функции, формирующие основу комплексной детали и функциональный элемент.

```
;Очистка поля чертежа формата А4
;(удобно использовать при отладке программы)
;(COMMAND "_.ERASE" "_C" "0,0" "210,297")
;Функция формирования параметрически управляемой основы
;комплексной детали.
;За базовую точку взят левый нижний угол прямоугольной проекции (x,y)
 (DEFUN OSN (x y D D1 L L1 L2 L3 / L2P L1P R1 R X1 X2)
(COMMAND "_.ERASE" "_C" "0,0" "210,297" "")
  (SETQ
        L2P (/ L2 2.0)       ;определение
        L1P (/ L1 2.0)             ;локальных переменных
        R1  (/ D1 2.0)             ;переменных
        R   (/ D 2.0)
)
  ;формирование контура и окружностей горизонтальной проекции.
  ; изображение размещается на заранее созданном слое  OSN.
  (COMMAND "_.LAYER" "_Set" "OSN" "")
  (COMMAND "_.LINE" (LIST X Y)
                (LIST (+ X L2) Y)
                (LIST (+ X L2) (+ Y L1))
                (LIST X (+ Y L1)) "_C"
)

(COMMAND "_.CIRCLE" (LIST (+ X L2P) (+ Y L1P)) R1)
(COMMAND "_.CIRCLE" (LIST (+ X L2P) (+ Y L1P)) R)
  ;формирование осевых линий  горизонтальной проекциии.
  ;Изображение находится в заранее созданном слои OSI,
;которому присвоен штрихпунктирный тип линий
(COMMAND "_.LAYER" "_Set" "OSI" "")
(COMMAND "_.LINE" (LIST (+ X L2P) (- Y 5.0))
                  (LIST (+ X L2P) (+ Y L1 5.0)) "")
(COMMAND "_.LINE" (LIST (- X 5.0) (+ Y L1P))
                  (LIST (+ X L2 5.0) (+ Y L1P)) "")
;Формирование фронтальной проекции в слое OSN
        (SETQ
          X1 X                        ;Определение координат
          Y1 (+ Y L1 20.0)     ;локальной базовой точки
         )                            ;для фронтальной проекции
        (COMMAND "_.LAYER" "_Set" "OSN" "")
        (COMMAND "_.LINE"
                (LIST (+ X1 L2P) (+ Y1 L3))
                (LIST X1 (+ Y1 L3))
                (LIST X1 Y1)
                (LIST (+ X1 L2) Y1)
                (LIST (+ X1 L2) (+ Y1 L3))
                (LIST (+ X1 L2P R1) (+ Y1 L3))
                (LIST (+ X1 L2P R1) (+ Y1 L))
```

Рис. 7.6. Результат работы программы, формирующей основу комплексной детали (1-й фрагмент; окончание см. на с. 71)

```
                (LIST (- (+ X1 L2P) R1) (+ Y1 L))
                (LIST (- (+ X1 L2P) R1) (+ Y1 L3)) ""
        )
         (COMMAND "_.LINE"
                (LIST (+ X1 L2P R) Y1)
                (LIST (+ X1 L2P R) (+ Y1 L)) ""
        )
;Формирование осевой линии фронтальной проекции.
;Изображениенаходится на слое OSI
(COMMAND "_.LAYER" "_Set" "OSI" "")
(COMMAND "_.LINE"
                (LIST (+ X1 L2P) (- Y1 5.0))
                (LIST (+ X1 L2P) (+ Y1 L 5.0)) ""
)
)
;(osn 20 20 15 45 40 45 80 10)
;(osn 140 10 6 15 120 20 30 50)
;(osn 20 200 25 35 20 35 180 5)
```

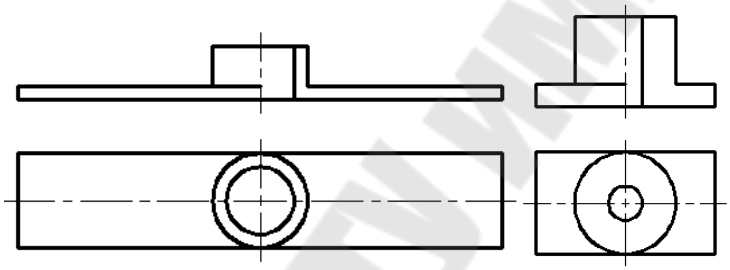


Рис. 7.6. Окончание (начало см. на на с. 70)

```
;Команда FUN1 Формирует прямоугольную проточку.
;Основа комплексной детали модифицируется автоматически
 (DEFUN C:FUN1 (/ BP1 XB1 YB1 L2P L4P L6P BP2 XB2 YB2 L3)
  ;Ввод базовой точки 1, расположенной в центре горизонтальной
  ;проекции, с объектной привязкой Intersection (пересечение)
  (COMMAND "_.OSNAP" "_INT")
  (SETQ BP1 (GETPOINT "\nВведите базовую точку 1:")
        XB1 (CAR BP1) ;Вычисление координат
        YB1 (CADR BP1);Базовой точки 1
)
  ;Ввод параметров проточки на горизонтальной проекции .
  ;Используется эффект "резиновой КИТУ от базовой точки 1
   (COMMAND "_.OSNAP" "_Perpend")
  (SETQ L2P (GETDIST BP1 "\nEnter L2/2 :"))
   (COMMAND "_.OSNAP" "_NONE")
   (COMMAND "_.OSNAP" "_NEA")
(SETQ L4P (GETDIST BP1 "\nВведите L4/2 :")
     L6P (GETDIST (LIST (+ XB1 L2P) YB1) "\n Enter L6/2 :")
)
```

Рис. 7.7. Результат работы команды Fun1,
формирующей прямоугольную проточку
(1-й фрагмент; продолжение и окончание см. на с. 72 и 73)

```
 (COMMAND "_.OSNAP" "_NONE")
 ;
 ;Формирование проточки на горизонтальной проекции.
 (COMMAND " _.LAYER" "_Set" "OSN" "")
 (COMMAND "_.PLINE" (LIST (+ XB1 L2P) (- YB1 L6P))
                    (LIST (+ XB1 L4P) (- YB1 L6P))
                    (LIST (+ XB1 L4P) (+ YB1 L6P))
                    (LIST (+ XB1 L2P) (+ YB1 L6P))
                    ""
 )
 (SETQ PPP1 (ENTLAST))
 (COMMAND "_.PLINE"
          (LIST (- XB1 L2P) (- YB1 L6P))
          (LIST (- XB1 L4P) (- YB1 L6P))
          (LIST (- XB1 L4P ) (+ YB1 L6P))
          (LIST (- XB1 L2P) (+ YB1 L6P))
          ""
)
 (SETQ PPP2 (ENTLAST))
;Модернизация контура горизонтальной проекции основы детали.
(COMMAND "_.TRIM" PPP1 ""
         (LIST (+ XB1 L2P) (+ YB1 (/ L6P 2.0)))
         ""
)
(COMMAND "_.TRIM" PPP2 ""
         (LIST (- XB1 L2P) (+ YB1 (/ L6P 2.0)))
         ""
)
;Ввод базовой точки 2, расположенной внизу в центре фронтальной
;проекции, с объектной привязкой Intersection (Пересечение)
(COMMAND "_.OSNAP" "_INT")
(SETQ BP2 (GETPOINT "\n Enter base point 2:")
    XB2 (CAR BP2) ;Вычисление координат
    YB2 (CADR BP2);базовой точки 2
 )
;Ввод параметров проточки на фронтальной проекции.
;Используется эффект "резиновой нити" от базовой точки 2
(COMMAND "_OSNAP" "_Perpend")
(SETQ L3 (GETDIST BP2 "\n Enter L3 :"))
;Формирование проточки на фронтальной проекции.
(COMMAND "_OSNAP" "_NONE")
(COMMAND "_.LINE" (LIST (+ XB2 L4P) YB2)
                  (LIST (+ XB2 L4P) (+ YB2 L3 ))
                  ""
)
)
 ;Команда запускается из командной строки AutoCAD
 ;указанием ее имени-FUN1
```

Рис. 7.7. Продолжение (начало см. на с. 71, окончание – на с. 73)

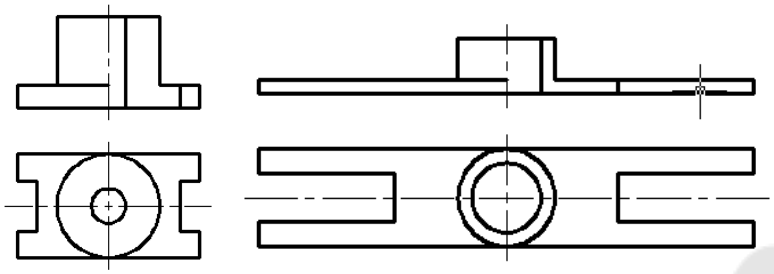


Рис. 7.7. Окончание (начало см. на с. 71)

Программа значительно упрощается при решении аналогичной задачи средствами объемного моделирования.

```
;Функция формирования параметрически управляемой основы
;комплексной детали
(DEFUN OSN (X Y Z D D1 L L1 L2 L3 / L1P L2P X1 Y1 Z1 H1 S C1 C2)
  (SETQ
    L2P (/ L2 2.0)
    L1P (/ L1 2.0)
    X1 (+ X L2P)
    Y1 (+ Y L1P)
    Z1 (+ Z L3)
    HL (- L L3)
    )
  ;Формирование параллелепипеда, лежащего в основании детали
  (COMMAND "_.BOX" (LIST X Y Z) "_L" L2 L1 L3)
  (SETQ B (ENTLAST))
  ;Формирования внешнего и внутреннего  цилиндров
  (COMMAND "_.CYLINDER" (LIST X1 Y1 Z1) "_D" D1 HL)
    (SETQ C1 (ENTLAST))
    (COMMAND "_.CYLINDER" (LIST X1 Y1 Z ) "_D" D L)
   (SETQ C2 (ENTLAST))
  ;Вычитание внутреннего цилиндра
  (COMMAND "_.SUBTRACT"  b C1 "" C2 "")
   (SETQ D1 (ENTLAST))
  ;Объединение параллелепипеда и цилиндров
 ; (COMMAND "_.UNION" B D1 "")
  ;Формирование фаски на цилиндре
  (COMMAND "_.CHAMFER" "_D" 2.5 2.5 "")
  (COMMAND "_.CHAMFER" (LIST X1 Y L) ""
            "" ""
                        (LIST X1 Y L) ""
)
  ;Удаление скрытых линий на изображении
  (COMMAND "_.HIDE")
  )
```

Рис. 7.8. Результат программы, формирующей основу комплексной детали (1-й фрагмент; окончание см. на с. 74)

```
;Примеры вариантов исполнения основы комплексной детали
;(osn 50 50 0 15 45 40 45 80 10)
;(osn 50 50 0 5 60 140 65 100 110)
;(osn 80 50 0 25 30 80 60 100 10)
```

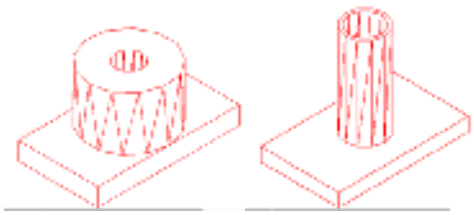


Рис. 7.8. Окончание (начало см. на с. 73)

7.2. Формирование чертежей с использованием генерирующего метода

Используя для описания чертежей ***технические конструктивные*** (***конструкционные***) ***элементы***, т. е. генерирующий метод, системы автоматизации конструкторской документации (АКД) позволяют создавать чертеж из отдельных элементов. Система AutoCAD содержит все необходимые средства для изготовления чертежей этим методом. В качестве языка программирования используется AutoLISP, что позволяет создать надстройку над редактором AutoCAD. На рис. 7.9 приведен пример команды выполнения цилиндрической проточки (проекция дуга) в призме (проекция – произвольная ломаная). Сложность и универсальность конструктивных элементов определяется классом деталей, а также уровнем квалификации пользователя AutoCAD.

Использование AutoCAD, AutoLISP позволяет строить открытые САПР системы АКД, т. е. вносить изменения в существующие элементы и разрабатывать новые. Кроме того, AutoCAD дает возможность строить разнообразные интерфейсы пользователя с системой путем использования: экранного меню пользователя, падающего меню, графического меню. Возможность увеличивать запас конструктивных элементов самому пользователю (создавать команды описания элементов) является важнейшим качеством систем, использующих, генерирующий метод.

Принцип работы системы, использующей генерирующий метод, основан на разделении чертежа изделия на элементы и создании новых чертежей изделий из имеющихся элементов. Различают следующие группы элементов: основные (функциональные), вспомогательные (конструктивные геометрические и геометрические элементы) и технологические. С помощью основных элементов описывается геометрическая форма детали (наружные и внутренние контуры), про-

точки (внутренняя и наружные). Это дает прежде всего общее описание детали. С помощью вспомогательных элементов осуществляется более подробное описание детали, что позволяет полностью передать геометрическую форму детали. Технологические элементы или характеристики относятся и к основным, и к вспомогательным элементам. Они влияют на простановку размеров. Конструктор должен встретить и использовать привычные для него элементы и методы описания детали. В AutoCAD это можно реализовать благодаря разнообразным возможностям организации интерфейса с конструктором, например, с использованием пиктограмм, обеспечивающих графическое отображение и выбор имеющихся в системе элементов.

```
 (DEFUN C:cilprot ( / entp entc r pt pr)
;цилиндрическая проточка
 (SETQ entp (ENTSEL "\nSelect object and point to prot:"))
 (SETQ pt (OSNAP (CADR entp) "_NEA")) ; центр протчки
 (SETQ pr  (GETPOINT pt "\nRadius to prot:")
       )
 (SETQ r (DISTANCE pt pr));радиус окружности
 (COMMAND "_.CIRCLE" pt r) ;окружность для проточки
 (SETQ entc (entlast)) ; имя примитива окружности
 (COMMAND "_.TRIM" entp entc "" ;контур отсечения
          ;указание противоположной точки на окружности
(LIST entc (POLAR pt (+ (ANGLE pt pr) PI) r))entp "") ;указание на объекте проточки
)
```

Рис. 7.9. Пример команды выполнения проточки
для произвольной ломаной

Тема 8. РАБОТА С ПРИМИТИВАМИ. ДОСТУП К БАЗЕ ДАННЫХ

8.1. Получение информации о примитивах

Любой создаваемый в AutoCAD чертеж состоит из примитивов, геометрическое описание которых хранится в специальном формате (формате AutoCAD) в файле чертежа (расширение .dwg). Для оперирования примитивами необходимо в программе на AutoLISP сначала найти имя примитива в базе данных AutoCAD. Попробуем извлечь это имя из графической базы данных (ГБД) при помощи AutoLISP.

Нарисуйте отрезок (команда _.line). Для того, чтобы указать на примитив, используются различные способы. В частности, можно указать последний нарисованный элемент. Получение имени последнего созданного примитива функция **(*ENTLAST*)**. Имя примитива необходимо сохранить в переменной с помощью функции присваивания *setq*.

Введите в командной строке:

```
(setq ENAME (entlast))
```

AutoCAD возвращает: *<Entity name: 60000018>*

Тем самым мы присвоили переменной *ENAME* имя последнего примитива (в данном случае отрезка). Имена примитивов в AutoCAD – 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)*.

Данные о примитиве выдаются списком, поэтому должны быть записаны в переменную, которую потом можно анализировать:

```
(setq EDATA (ENTGET ENAME))
```

В результате выполнения команды вы получите сообщение, содержащее в своей структуре всю информацию о примитиве, начиная от его имени и заканчивая слоем и цветом:

```
((-1 . <Имя примитива: 60000020>) (0 . "LINE") (8 . "0") (10 1.0 2.0 0.0) (11 6.0 6.0 0.0))
```

Эти списки получили название DXF-кодов.

Список данных о примитиве состоит из кодов формата DXF (Drawing exchange Format – формат обмена рисунками). 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 следующая за ним строка – имя 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, равным –1. Если цвет или тип линии не указываются, значит они соответствуют типам линии и цвету для заданного слоя.

Можно привести следующую иерархию описаний графических примитивов в AutoCAD по последовательности доступа к ним:

1. Набор графических примитивов.
2. Имя примитива.
3. Список примитива.
4. Элементы списка примитива (подписки).

8.2. Функции имени примитива

Как уже говорилось, каждый примитив в наборе имеет имя. Это могут быть не такие имена, как LINE или CIRCLE.

Извлечь имена примитивов из набора можно с помощью функций ***(entsel)***, ***(entlast)***, ***(entnext)***.

Функция ***ENTNEXT*** получает имя следующего примитива. Если нет аргумента, то она возвращает имя первого не удаленного примитива, а если аргумент задан, то возвращает имя примитива, следующего за определенным в аргументе:

(ENTNEXT [<имя примитива>])

Пример

(SETQ e1 (ENTNEXT)); e1 – первый примитив в рисунке

(SETQ e2 (ENTNEXT e1)); e2 – второй примитив в рисунке

Функция (***ENTLAST***) получает имя последнего не удаленного примитива. Обычно используется для извлечения имени примитива, который был создан с помощью функции *COMMAND*.

Функция ***ENTSEL*** выбирает примитив путем его указания. Если нет подсказки, то выдается запрос *Выберите объект*. Данная функция возвращает список, первый элемент которого – имя примитива, а второй точка указания, представляющая собой список из двух координат (***ENTSEL [<подсказка>]***)

Пример

(SETQ e (ENTSEL "Укажите примитив"))

В этом случае *(CAR e)* – имя примитива; *(CADR e)* – точка указания.

Некоторые команды AutoCAD, например, BREAK (РАЗОРВИ), TRIM (ОБРЕЖЬ) и EXTEND (УДЛИНИ) требуют, чтобы вместе с именем передавалась точка, с помощью которой был выбран примитив. Функция ***ENTSEL*** обрабатывает ключевые слова, если они предварительно были описаны функцией ***INITGET***.

8.3. Функции манипуляции набором примитивов

Функция ***SSGET*** получает набор примитивов:

(SSGET [<режим>] [<точка1>] [<точка2>] [<список точек>] [<фильтр-список>])

Аргумент <режим> определяет способ формирования набора примитивов и может принимать следующие значения:

"_W" ("Р")	– выбор окном;
"_C" ("С")	– выбор секущим окном;
"_L" ("П")	– последний созданный примитив;
"_P" ("Т")	– текущий набор;
"_F" ("Л")	– выбор линией (разомкнутым многоугольником);
"_CP" ("СМ")	– выбор секущим многоугольником;
"_WP" ("РМ")	– выбор рамочным многоугольником;
"X"	– выбор всех примитивов.

Аргументы <точка1>, <точка2> – точки, используемые для создания набора. При отсутствии аргументов применяется обычный механизм AutoCAD для формирования набора примитивов (по запросам *Выберите объект*).

Пример

(SSGET) – обычный способ формирования набора.

(SSGET "_P") – выбор текущего набора.

(SSGET "_L") – выбор последнего объекта, добавленного к базе данных AutoCAD.

(SSGET '(2 2)) – выбор примитива, проходящего через точку 2,2.

(SSGET "_W" '(0 0) '(5 5)) – выбор примитива в окне от 0,0 до 5,5.

Переменная набора примитива может быть передана в AutoCAD в ответ на любой запрос *Выберите объект*, допускающий ответ *Last*.

Аргумент <список точек> используется для выбора с помощью многоугольника.

Аргумент <фильтр-список> может использоваться в любом режиме выбора примитива и предоставляет средство точного управления процессом отбора.

<фильтр-список> – список, содержащий подсписки, каждый из которых – точечная пара, включающая код или характеристику и значение.

Пример

(SSGET “X” (LIST (CONS 0 “CIRCLE”))); создает набор, включающий все окружности.

Возможны следующие варианты определения аргумента <код>:

0 – тип примитива;

2 – имя блока;

6 – имя типа линии;

7 – имя стиля шрифта;

8 – имя слоя;

62 – код цвета (0 – BYBLOCK, 256 – BYLAYER).

По умолчанию для каждого элемента <фильтра-список> применяется условие равенства. Для числовых значений (целых, вещественных, точек и векторов) можно указывать другие условия, включив в фильтр-список специальные списки с кодом, равным –4 и значением, представляющим собой строковую константу, содержащую оператор сравнения, который будет применен к следующему подсписку в фильтре-списке:

“*” – любое значение;

“=” – равно;

“!=”, “/=”, “<>” – не равно;

“<” – меньше;

“>” – больше;

“<=” – меньше или равно;

“>=” – больше или равно;

“&” – побитовое И (только для целых значений);

“&*” – побитовое равенство по маске (только для целых значений).

Пример

(SSGET “X” ‘((0 . “CIRCLE”) (-4 . “>=”) (40 . 2.0))) – отбирает все круги, радиус которых (код 40) больше или равен 2.

Для точек сравнения координаты *x*, *y*, *z* могут быть сцеплены в одну строку символов, где операторы перечислены через запятую (например, “<, >, *”). Если оператор пропущен в строке (например, “=, /=” и нет сравнения для координаты *z*), по умолчанию применяется оператор “*” (любое значение).

Векторы направления (код типа 210) могут сравниваться только с помощью операторов “*”, “=”, “!=” (или одного из эквивалентов оператора «не равно»).

Применять операторы сравнения со строковыми значениями нельзя, необходимо пользоваться символами глобальной замены.

Вышеописанные операторы сравнения являются двоичными, то есть функциями двух аргументов. Однако можно создать логические выражения для тестирования нескольких подсписков путем использования групповых операторов, включая в фильтры-списки такие последовательности:

(-4 . “<AND”) <один или несколько операндов> (-4 .”AND>”)

(-4 . “<OR”) < один или несколько операндов > (-4 .”OR>”)

(-4 . “<XOR”) <два операнда> (-4 .”XOR>”)

(-4 . “<NOT”) <один операнд> (-4 .”NOT>”)

Пример

(SSGET “X” ‘((-4 . “<OR”) (-4 . “<AND”) (0 . “CIRCLE”) (40 . 1.0) (-4 . “AND >”) (-4 . “< AND”) (0 . “LINE”) (8 . “ABC”) (-4 . “AND >”) (-4 . “OR>”)))

Отбирает все круги радиусом 1.0 плюс все отрезки на слое ABC.

Для фильтрации по расширенным данным примитивов следует использовать с кодом, равным –3.

Пример

(SSGET “X” ‘((0 . “CIRCLE”) (-3 “APFNAME”))) ;отбирает все круги, содержащие расширенные данные приложения APFNAME.

В пользовательских программных приложениях используются расширенные данные примитива, которые записываются после обычных данных его описания. Идентифицировать такой фрагмент позволяет код, равный –3, который находится в списке непосредственно перед первой группой расширенных данных. Расширенные данные состоят из одной или более групп 1001, каждая из которых начинается с уникального имени приложения:

Код группы	Описание группы
((-1, -2	Имя примитива))
((0-239	Поля обычных данных описания примитива))
((-3	Сигнал расширенных данных))
((1001	Имя зарегистрированного приложения))
((1000, 1002-1107	Поля расширенных данных))
((1001….))	

Если значение подсписка с кодом, равным –3 содержит более одного имени приложения, подразумевается оператор <логическое И>, а это значит, что примитив должен содержать расширенные данные для всех приложений, перечисленных в списке.

Пример

(SSGET “X” ‘((0 . “CIRCLE”) (-3 (“APPL1”) (“APPL2”)))) ;выбирает все круги с расширенными данными как для приложения “APPL1”, так и для приложения “APPL2”.

(SSGET "X" '((0 . "CIRCLE") (-3 ("APPL1, APPL2")))) ;выбирает все круги, содержащие расширенные данные, хотя бы одного из перечисленных приложений.
(SSGET "X" '((0 . "CIRCLE") (-3 ("APPL[1 2]"))) ; можно применять шаблон глобальных символов.

Символьные имена, задаваемые в фильтрах-списках – тип примитива (0), имя блока (2), имя размерного стиля (3), тип линии (6), гарнитура шрифта (7), имя слоя (8) – могут содержать те же шаблоны глобальных символов, что и для функции ***WCMATCH***. При фильтрации блоков без имени следует перед символом * ставить апостроф (‘), так как * воспринимается функцией ***SSGET*** глобальный символ.

Пример

*((SSGET "X" '(2 . " '*U2"))* ;Включает в набор блок без имени *U2

При необходимости быстро стереть все с заданного слоя будет полезна нижеприведенная программа (рис. 8.1).

```
(defun C:ERLAY ()
(SETQ a (GETSTRING "\nИмя слоя")
        b (SSGET "X" (LIST (CONS 8 a)))
)
(COMMAND "_ERASE" b "")
)
```

Рис. 8.1. Листинг программы ERLAY

После введения имени в ответ на запрос : *Имя слоя* все содержимое будет стерто.

Функция ***SSLENGTH*** вычисляет количество примитивов в наборе. Возвращает целое, представляющее число примитивов в наборе: *(SSLENGTH <набор>)*

Пример

(SETQ sset (SSGET "_L"))

(SSLENGTH sset) ;возвращает 1

Ниже приводится короткая программа (рис. 8.2), создающая из группы объектов единую полилинию

```
(defun  C:unline ( /  a)
(setq a (SSGET)
(IF (> (SSLENGTH a ) 1 )
(COMMAND "_pedit" a "_y"  "_j" a "" "_x")
)
)
```

Рис. 8.2. Листинг программы UNLINE

При использовании этой программы следует выбрать любую группу объектов, применив рамку выбора или секущую рамку, а затем подтвердить выбор. В результате элементы будут соединены как единая полилиния.

Функция ***SSNAME*** получает имя примитива из набора. Возвращает имя примитива, который входит в <набор> под указанным <номером>. Нумерация начинается с нуля:

(SSNAME <набор> <номер>)

Пример

```
(SETQ sset (SSGET))                ;Создание набора
(SETQ ent1 (SSNAME sset 0)) ;Имя первого элемента
(SETQ ent2 (SSNAME sset 1)) ;Имя второго элемента
```

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

Функция ***SSADD*** добавляет примитив в набор. Функции без аргументов соответствует созданию пустого набора. Если есть только аргумент <имя примитива>, то создается новый набор, который содержит лишь один примитив. Если есть оба аргумента, то примитив добавляется в набор.

(SSADD [<имя примитива> [<набор>]]

Пример

```
(SETQ e1 (ENTNEXT)) ;     e1 первый примитив
(SETQ ss (SSADD)) ; ss пустой  набор
(SSADD e1 ss) ; e1 добавлен в  ss
(SSADD (ENTNEXT e1) ss) ; добавлен в ss примитив следующий за e1
```

Независимо от способа добавления примитивов в набор повторные примитивы в нем никогда не содержатся. Если добавляется примитив, уже существующий в наборе, последнее добавление игнорируется.

Функция ***SSDEL*** удаляет примитив из набора. Если примитива в наборе нет, то возвращается NIL

(SSDEL <имя примитива> <набор>)

Функция ***SSMEMB*** проверяет наличие примитива в наборе. Если примитив не входит в набор, то возвращается NIL, в противном случае – имя примитива.
(SSMEMB <имя примитива> <набор>).

8.4. Функции оперирования данными примитивов чертежа

Функция ***ENTDEL*** удаляет примитив:
(ENTDEL <имя примитива>)
Примитив, определенный аргументом, удаляется из чертежа или восстанавливается на чертеже, если был удален прежде. Удаленные примитивы уничтожаются из базы данных по окончании сеанса редактирования, т. е. восстановление возможно только в течение сеанса.

Известно, что в редакторе AutoCAD имеется команда ERASE ("СОТРИ"), которая позволяет стереть последний элемент чертежа или элемент, указываемый конструктором. Однако оба эти способа могут оказаться неудобными в следующей ситуации: конструктор создает сложный насыщенный чертеж и ему не сразу удается правильно построить нужный элемент. Конструктор хотел бы повторить попытку, но при этом сохранить неправильный элемент в качестве «отправной точки», а после удачного построения – стереть этот элемент. Имея описанную выше функцию, он сможет это сделать, подав команду ERASE2.

Приведенная ниже программа решает задачу просто и изящно (рис. 8.3): стирается последний примитив (но его имя записывается в переменную secondlast); стирается примитив, ставший последним, т. е. бывший предпоследний; путем повторного применения функции ***ENTDEL*** по отношению к примитиву secondlast стертый последний элемент восстанавливается на экране

```
; Удаление предпоследнего элемента
(DEFUN C:ERASE2 ( )
(SETQ secondlast (ENTDEL (ENTLAST)))
(ENTDEL (ENTLAST))
(ENTDEL secondlast)
)
```

Рис. 8.3. Листинг программы ERASE2

Функция ***ENTGET*** получает информацию о примитиве:
(ENTGET <имя примитива> [<список>])

Примитив, имя которого задано аргументом функции, выбирается из базы данных и возвращается в виде списка, который состоит из данных, определяющих примитив. Результирующий список оформлен как ассоциативный – его элементы могут быть легко извлечены с помощью функции ***ASSOC***. Для примера нарисуем отрезок, а затем извлечен из базы данных информацию о нем.

```
(COMMAND “_LINE” “1,1” “6,6” “”)
(SETQ a (ENTGET (ENTLAST)))
```

В этом случае символу *a* присваивается значение

((-1 . <Имя примитива: 60000020>)
(0 . "LINE"); тип примитива
(8 . "0"); слой
(10 1.0 2.0 0.0); начальная точка
(11 6.0 6.0 0.0)); конечная точка

Элемент с ключом, равным –1 в начале списка содержит имя примитива, которому этот список соответствует. Функция ***ENTMOD***, описанная ниже, использует имя для идентификации примитива, который будет модифицироваться. Целое число 0 представляет собой код типа примитива (*LINE*). Код 8 говорит о том, что следующая за ним строка – имя слоя. Код 10 – начальная точка примитива, код 11 – конечная и т. п.

Функции обработки подобных списков необходимо делать нечувствительными к порядку расположения подсписков. Использование функции ASSOC гарантирует это.

Аргумент <список> позволяет задавать имена зарегистрированных приложений. При этом будут возвращены и расширенные данные, связанные с указанным приложением.

Функция ***ENTMOD*** модифицирует информацию о примитиве в базе данных AutoCAD:

(ENTMOD <список>)

Аргумент <список> представлен в том же виде, что и список, возвращаемый функцией ***ENTGET***. Основной механизм, с помощью которого AutoLISP обновляет базу данных, – это извлечение информации о примитивах с помощью ***ENTGET***, изменение списка, описывающего примитив (для чего удобна функция ***SUBST***), и обновление примитива в базе данных с помощью ***ENTMOD***.

Пример

(SETQ en (ENTNEXT)); *en* – имя первого примитива
(SETQ ed (ENTGET en)); *ed* – информация о примитиве en
(SETQ ed ; модификация информации

(*SUBST (CONS 8 “1”)*; создание списка для слоя 1
(ASSOC 8 ed); поиск списка, определяющего слой
ed)) ; список информации о примитиве
(ENTMOD ed); модификация информации в базе данных

Пример

В процессе работы с AutoCAD иногда удобно изменять свойства примитива указанием на другой примитив. Программа изменения свойства примитива (слой) указанием на другой примитив (рис. 8.4).

```
 (defun chlay ()
(setq a (ssget))
(setq lays (cdr (assoc 8
(entget (car (entsel "\nВыберите примитив для создания слоя "))))))
(setq n (sslength a))
(setq index 0)
(repeat n
(setq ent (entget (ssname a index)))
(setq index (1+ index))
(setq lay (assoc 8 ent))
(setq l (cons (car lay) lays))
(setq ent1 (subst l lay ent))
(entmod ent1)
))
```

Рис. 8.4. Листинг программы CHLAY

Возможности функции ***ENTMOD*** ограничены. Прежде всего, она не позволяет изменить тип примитива – этого удастся достичь, только удалив его с помощью функции ***ENTDEL*** и создав заново с помощью ***COMMAND***. Все объекты, на которые создается список примитива, должны быть известны AutoCAD к тому моменту, когда вызывается функция ***ENTMOD***. Сказанное относится к стилю шрифта, типу линии, именам блоков. Исключение составляет имя слоя, который может быть создан функцией ***ENTMOD***. Если зафиксирована ошибка и база данных не может быть изменена, то возвращается NIL, в противном случае возвращается список, соответствующий аргументу функции.

Использование вышеописанных функций проиллюстрировано на примере программы уменьшения и восстановления ширины полилинии.

Полилинии сложно выбирать, если им присвоен параметр ширины, отличный от нуля, и включена опция *Fill* (*Закрась*). Зачастую бывает проще решить эту проблему, уменьшив ширину полилинии до 0 и восстановив ее позже. Ниже приведенная программа уменьшает ширину полилинии до 0 (рис. 8.5).

```
(Defun C:PWO (/ a )
(SETQ a (ENTGET (SSNAME (SSGET)))
;Bw (CDR (ASSOC  40 a))
;ew (CDR (ASSOC  41 a))
a (SUBST '(40 .  0 ) (ASSOC 40 a ) a)
a  (SUBST '(41 . 0 ) (ASSOC 41 a ) a)
)
(ENTMOD a)
)
```

Рис. 8.5. Листинг программы PWO

После загрузки вышеописанной команды в ответ на запрос *Выберите объект*: необходимо растянуть секущую рамку на любом сегменте полилинии и подтвердить выбор. Ширина выбранной полилинии уменьшится до нулевой.

Когда функция ***ENTMOD*** применяется для модификации подпримитива (вершины ломаной или атрибутов блока), информация в базе данных изменяется, а для изменения изображения на экране необходимо использовать функцию ***ENTUPD***.

Функция ***ENTUPD*** обновляет изображения на экране. Предназначена для обновления на экране модифицированных ломаной или блока, однако может быть вызвана для любого примитива; при этом примитив на экране всегда будет регенерироваться: *(ENTUPD <имя примитива>).*

Функция ***ENTMAKE*** создает новый примитив в базе данных рисунка: *(ENTMAKE [<список примитива>]).*

Если примитив создан, эта функция возвращает список данных, определяющих примитив, а в противном случае – NIL. Перед созданием нового примитива ***ENTMAKE*** проверяет корректность имен слоя, типа линий и цвета. Если указанное имя слоя отличается от тех, что хранятся в базе данных рисунка, создается новый слой под этим названием. Проверяются имена блоков, размерных стилей, гарнитур текста и форм.

Пример

(ENTMAKE ' ((0 . "CIRCLE") (62 . 1) (10 1.0 4.0 0.0) (40 . 1.0)))

В данном примере необязательные параметры <слой> и <тип линии > опущены – им будет присвоено значение по умолчанию.

8.5. Примеры программ, использующие ГБД AutoCAD

Пример 1

Программа для отрисовки прямоугольника вокруг выбранного текстового примитива. Для решения задачи используется функция

(textbox <список>). Данная функция измеряет заданный текстовый примитив и возвращает координаты диагонали описывающего текст прямоугольника. Аргумент <список> должен описывать текстовый примитив. В случае успешного завершения функция возвращает список двух точек, в противном случае – *nil*. Минимальный список, который можно передать *(textbox)*, это сама текстовая строка. Точки, возвращаемые функцией *(textbox)*, задают описывающий текстовый примитив Прямоугольник так, как если бы этот примитив был вставлен в точку с координатами (0, 0, 0) с углом поворота, равным 0. Независимо от ориентации строки и гарнитуры текста точки, возвращаемые функцией (textbox), вычисляются относительно точки вставки текста (код группы 10), непосредственно преобразованной в объектную систему координат. На эту точку следует ссылаться при преобразовании координат, возвращаемых функцией (textbox), для определения действительных границ текста (рис. 8.6).

```
 (defun win  (/ textent tb ll ur ul lr)
(setvar "osmode" 0)
  (setq textent (entsel "\ n Выберите текст: "))
(command "_.UCS" "íî" "î"  textent)
(setq tb  (textbox (list (cons -1 (car textent))))
ll (car tb)
ur (cadr tb)
ul (list (car ll) (cadr ur))
lr (list (car ur) (cadr ll))
)
(command "_.pline" ll lr ur ul "_close")
(command "_.UCS" "_p")
(princ)
)
```

Рис. 8.6. Листинг программы WIN

Пример 2

Данная программа копирует дуги на другой слой с другим цветом и со сдвигом (рис. 8.7).

```
; ПРОГРАММА
;-------------------------------------------------------------------------
(defun get_arcs ( / i aa aa2 el_t rad cen st_a en_a)
; подбор примитивов - дуг
 (setq aa (ssget "X" '((0 . "ARC") (8 . "1")) ))
; обработка окружностей - копирование на другой слой
 (setq i 0)
```

Рис. 8.7. Листинг программы GET_ARCS
(1-й фрагмент; окончание см. на с. 88)

```
; проверка простая - действия выполняются, если список aa существует
 (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)); радиус
    cen (cadr (assoc 10 el_t)); центр
    st_a(cdr (assoc 50 el_t)); стартовый угол
    en_a(cdr (assoc 51 el_t)); конечный угол
   )
; 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")
 )
)
```

Рис. 8.7. Окончание (начало см. на с. 87)

Пример 3

Рассмотрим еще один пример – программу, которая позволяет изменить высоту символов, ранее введенного текста (рис. 8.8).

```
(defun chtext (/ a ts n index b1 b c d b2)
(setq a (ssget)); выбор объектов
(setq ts (getreal "Введите новую высоту текста"))
(setq n (sslength a))
(setq index 0)
(repeat n (setq b1 (entget (ssname a index)))
(setq index (1+ index)) (setq b (assoc 0 b1))
(if (= "TEXT" (cdr b))
(progn
(setq c (assoc 40 b1))
(setq d (cons (car c) ts))
(setq b2 (subst d c b1))
(entmod b2))
)))
```

Рис. 8.8. Листинг программы CHTEXT

В программе определены только локальные переменные. С помощью функции (*ssget*) пользователю предоставляется сделать выбор текста стандартным образом – с помощью опций "_WINDOW(РАМКА)" или "_CROSSING (СЕКРАМКА)". Размер задается с клавиатуры. Количество выбранных примитивов в наборе определяется с помощью функции *(sslength)*. В выбранные примитивы могут входить не только текст, но и другие примитивы. Далее производится цикл по всем выбранным примитивам. Извлекается очередной примитив. В переменную *b* записывается подсписок, включающий код группы *"0"* – имя примитива. Если обрабатываемый примитив – текстовый, в переменную *c* назначается подсписок, определяющий высоту текста (код группы *40*). С помощью функций (*cons*) и (*car*) конструируется подсписок с новой высотой текста. Следующая строка программы производит замену старого подсписка *c* на новый *d* в списке *b1*. Затем новое значение списка *b1* присваивается *b2*. С помощью функции *(entmod)* новый список *b2* записывается в графическую базу данных.

Возможна следующая модернизация программы:

– вместо: *(setq a (ssget))* можно вставить: *(setq a (ssget "X" ((0. "TEXT"))))*. В этом случае, программа будет изменять высоту всех текстовых примитивов. Так как теперь в наборе остались одни текстовые примитивы, таким образом, нет смысла извлекать имя примитива и делать проверку на его соответствие строке "TEXT".

```
(setq b (assoc 0 b1))
(if (= "TEXT" (cdr b)))
```

Тема 9. РАЗРАБОТКА ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ИНТЕРФЕЙСА

9.1. Файлы меню AutoCAD

AutoCAD содержит средства объектно-ориентированного пользовательского интерфейса, который обеспечивает комфортный диалог пользователя с компьютером, ускоряет разработку конструкторских документов и решение объектно-ориентированных задач.

Файлы меню AutoCAD представляют собой простые текстовые документы. Чтобы создать собственное меню, необходимо модифицировать исходный файл меню и перекомпилировать его. Для этих целей подойдет любой текстовый редактор, например WordPad. Исходные ASCII – файлы меню называются acad.mnu и acad.mns. Файл acad.mns имеет такую же структуру, что и acad.mnu, но, в отличие от него, не содержит комментариев и специального форматирования. При первой загрузке файла acad.mnu AutoCAD компилирует его под именем acad.mnc в двоичный файл, содержащий строки команд, описание функций, внешнего вида меню и других элементов интерфейса. Во время каждой компиляции acad.mnc генерирует файл ресурсов acad. mnr, который представляет собой двоичный файл, содержащий растровые изображения, используемые меню и другими элементами интерфейса.

При загрузке файла меню acad.mnu автоматически загружается в память файл acad.mnl, где хранится программный код AutoLISP и содержатся LISP-выражения, которые могут использоваться файлом меню.

Все изменения, которые будут рассмотрены ниже, следует сохранить в файле acad.mnu: это позволит сохранить настройку панелей инструментов при редактировании системы падающих меню.

Структура файла позволяет разбить его на разделы, каждый из которых относится к некоторому устройству, поддерживающему работу с меню, и содержит командные строки, предназначенные для этого устройства. Разделы идентифицируются с помощью специальных меток. Заголовок раздела образуется из трех символов «звездочка» (***) и имени раздела:

– *** MENUGROUP – задает имя группы файла меню;

– ***SCREEN – управляет экранным меню и подменю, которые появляются в правой части графического экрана. В данном разделе все допустимые команды AutoCAD;

– ***BUTTONSn – позволяет присваивать каждой кнопке мыши (или другого манипулятора) различные функции. К их числу относятся управляющие коды, в частности, переключатели режимов и команды AutoCAD. Кнопкой выбора всегда считается левая кнопка мыши – ее функция не может быть изменена;

– ***AUXn – позволяет адаптировать конфигурацию кнопок системного устройства указания;

– ***POPn – создает в верхней части экрана строку падающих меню, обеспечивает их содержание и форматирование, а также описывает назначение пунктов контекстного меню (раздел POP0);

– ***IMAGE – формирует ряд графических меню, которые позволяют выбирать команды или опции из поля списка, поля слайдов или соответствующего им поля выбора на графическом экране;

– ***TABLETn – обеспечивает работу с планшетным меню, которая сводится к простому указанию на цифровом планшете. При этом команда может быть представлена графической пиктограммой на шаблоне (template) планшета;

– ***TOOLBARS – описывает панели инструментов, позволяя задавать их имена, статус (падающая или закрепленная, видимая или невидимая) и положение на экране. Кроме того описываются все кнопки инструментов и их свойства;

– ***HELPSTRINGS – содержит текст, выводимый в строке состояния, когда выделен тот или иной пункт падающего или контекстного меню, а также, когда указатель мыши наведен на некоторую кнопку панели инструментов;

– ***ACCELERATORS – отражает назначение комбинаций клавиш быстрого вызова определенных макросов меню.

Метки показывают, что соответствующие пункты меню до метки следующего раздела или до конца файла относятся к конкретному устройству, работающему с меню.

В файлах с меню используются следующие управляющие последовательности символов:

*** – указывает заголовок раздела;

** – указывает метку раздела в подменю;

[] – ограничивает заголовки для экранного , контекстного, падающих и графических меню, пунктов меню, для имен слайдов или текста;

[ ] – пробел между элементами последовательностей команд в пунктах меню – аналог нажатие клавиши пробел;

? или ^M – имитирует нажатие клавиши Enter;

^I – имитирует нажатие клавиши Tab;
\ – выдает паузу для ввода пользователем информации (не может применяться в разделе ***ACCELERATORS);
- – переводит следующую за этим символом команду или ключевое слово AutoCAD используемой версии, например, на русский язык;
+ – продолжает макрос на следующей строке, если является в ней последним символом;
=* – выводит текущую пиктограмму, падающее, графическое или контекстное меню на экран;
$ – специальный символьный код, предназначенный для загрузки раздела меню или ввода условного макровыражения на языке DIESEL;
*^C^C – префикс для повтора пункта.

С целью включения или отключения установок AutoCAD в файле меню могут использоваться следующие управляющие коды:
^B – переключатель режима SNAP (ШАГ);
^C – отмена выполнения команды (Esc);
^D – переключатель отображения координат (Ctrl+D);
^E – циклический переключатель плоскости изометрии(Ctrl+E);
^G – переключатель режима GRID (СЕТКА);
^O – переключатель режима ORTHO (*ОРТО);
^P – переключатель эха команд AutoCAD на подсказку Command;
^T – переключатель режима Планшет (Ctrl+T);
^H – имитация клавиши Backspace;
^Q – параллельный вывод подсказок, сообщений и вводимых пользователем данных на принтер (Ctrl+Q);
^Z – нуль-символ, подавляющий автоматическое добавление пробела в конец пункта меню;
^V – переключатель текущего видового экрана.

9.2. Экранное меню

Раздел ***SCREEN файла меню acad.mnu управляет экранным меню, которые появляются в правой части графического экрана. При запуске команды AutoCAD в области экранного меню возникает соответствующее ей подменю – группа пунктов в разделе меню. Подменю временно замещает все текущее меню или его часть.

Ниже приведен фрагмент файла русскоязычного меню, содержащего раздел верхнего уровня экранного меню:

```
***SCREEN
**S
[AutoCAD ] ^C^C^P [ai_rootmenus] ^P
[*  *  * *  * ] $S=ACAD.OSNAP
[ФАЙЛ     ] $S=ACAD.01_FILE
[ПРАВКА  ] $S=ACAD.02_EDIT
[ВИД   1   ] $S=ACAD.03_VIEW1
[ВИД   2   ]S=ACAD.04_VIEW2
[ВСТАВКА] $S=ACAD.05_INSERT
[ФОРМАТ ]$S=ACAD.06_FORMAT
[СЕРВИС 1] $S=ACAD.07_TOOLS1
[СЕРВИС 2] $S=ACAD.08_TOOLS2
[РИСУЙ  1 ] $S=ACAD.09_DRAW1
[РИСУЙ 2  ] $S=ACAD.10_DRAW2
[РАЗМЕРЫ] $S=ACAD.11_DIMENSION
[РЕДАКТ 1] $S=ACAD.12_MODIFY1
[РЕДАКТ 2] $S=ACAD.13_MODIFY2
[СПРАВКА] $S=ACAD.14_HELP
```

Каждая строка раздела ***SCREEN содержит ряд пунктов из перечисленных ниже:

[ТЕКСТ] – отображает сообщение;

$S – вызывает подменю и отображает его на экране;

^C^C – отменяет любую текущую команду;

_NAME – запускает команду AutoCAD.

Каждое подменю в разделе ***SCREEN имеет формат **NAME n,

где NAME – имя подменю;

n – номер строки экранного меню, с которой должно начаться подменю.

Например, подменю раздела DRAW1 в русскоязычной версии выглядит так:

```
** 09_DRAW1 3
[Отрезок ] ^C^C_line
[Луч       ] ^C^C_ray
[Прямая  ] ^C^C_xline
[Млиния ] ^C^C_mline
[Плиния ] ^C^C_pline
[3-Поли  ] ^C^C_dpoly
[Мн-угол] ^C^C_polygon
```

```
[Прямоуг] ^C^C_rectang
[Дуга     ] ^C^C_arc
[Круг     ] ^C^C_circle
[Кольцо   ] ^C^C_donut
[Сплайн   ] ^C^C_spline
[Эллипс   ] ^C^C_ellipse
```

Пример следующего уровня подменю команды CIRCLE в русскоязычной версии:

```
**CIRCLE 3
[Круг:       ] ^C^C_circle
[Ц, Рад      ] \\
[Ц, Диам     ] \_d
[2 точки     ]_2p \
[3 Рточки    ]_3p \\
[ККР         ]_ttr
[ККК         ]_3p _tan \_tan \_tan \
[Коп рад:    ] ^C^C_circlerad _cal rad
```

Когда требуется с клавиатуры или при помощи мыши ввести информацию в середине пункта меню, то используется символ \ (обратная косая черта).

Обычно символы, считанные из пункта меню, отображаются в зоне подсказок экрана, как при вводе с клавиатуры, и запросы высвечиваются даже в том случае, когда пункт меню включает ответы. Эту информацию можно подавить с помощью системной переменной MENUECHO. Если эхо-вывод отключен, управляющая последовательность ^P в пункте меню включает его, и наоборот, если эхо-команд включен, то отключает.

9.3. Падающие меню

Разделы от ***POP1 до ***POP499 содержат падающие меню, в каждое падающее меню может входить до 999 пунктов. Разделы ***POP0 и от ***POP500 до ***POP999 – контекстные, максимальное ограничение для контекстных меню – 499 пунктов. Оба предельных значения учитывают все подменю в иерархии. Если количество пунктов какого-то меню превышает предел, AutoCAD игнорирует лишнее. Если длина падающего или контекстного меню слишком велика и оно не помещается на экране, лишние пункты усекаются.

POP1 содержит падающее меню File (Файл), POP2 – падающее меню Edit (Правка), POP3 – падающее меню View (Вид) и т. д. Ниже приведен фрагмент файла acad.mnu в русскоязычной версии.

```
***POP0
**SNAP
            [&Контекстное меню привязки]
ID_Tracking [T&очка отслеживания]_tt
ID_ From  [&Смещение]_from
ID_ MnPointFi [->Координатные &фильтры]
ID_ PointFilx   [.X].X
ID_ PointFily   [.Y].Y
ID_ PointFilz   [.Z].Z
                [-]
ID_ PointFixy   [.XY].XY
ID_ PointFixz   [.XZ].XZ
ID_ PointFiyz   [.YZ].YZ
                [--]
…
***POP1
**FILE
ID_MnFile             [&Файл]
ID_New                [Соз&дать… Ctrl+N] ^C^C_new
ID_Open               [&Открыть… Ctrl+O] ^C^C_open
ID_DWG_ CLOSE    [&Закрыть] ^C^C_close
ID_PartialOp          [$ (if, $ (eq, $ (getvar, fullopen), 0) ,, -) &Частичная
                      загрузка]^C^C_partialload
            [--]
ID_Save               [&Сохранить Ctrl+S]^C^C_qsave
ID_Saveas             [Сохранить &как… Ctrl+Shift+S]^C^C_saveas
ID_ ETransmit [C&формировать комплект…]^C^C_etransmit
ID_Publish[Пу&бликация в Интернете…]^C^C_publishtoweb
ID_Export [&Экспорт…]^C^C_export
                [--]
…
***POP3
**VIEW
ID_MnView                [&Вид]
ID_Redrawall              [&Освежить]’_redrawall
ID_Regen                 [&Регенерировать]^C^C_regen
ID_Regenall              [Ре&генерировать все]^C^C_regenall
                       [--]
…
```

Первая строка в описании падающего меню POPn указывает его позицию, например ***POP1 – для меню File (Файл). Строка ID_MnFile [&File] указывает имя меню. При этом идентификатор ID_MnFile является уникальным в файле acad.mnu и не должен повторяться. Запись в квадратных скобках [&File] определяет заголовок меню.

Каждая строка описания меню POP содержит часть пунктов из следующего списка:

[ТЕКСТ] – заголовок, который не работает как команда, а служит в данном меню названием, появляющимся в строке падающих меню;

[…] (многоточие) – по принятому в AutoCAD соглашению это поле указывает, что выбор пункта меню вызывает появление другого меню или диалогового окна;

[--] – разделитель частей меню. Если используется без других символов, то пространство между пунктами расширяется и пересекается разделительной линией. При выдаче падающего меню эта строка увеличивается во всю его ширину;

[->] – отмечает начало каскадного меню. Текст, заключенный в квадратные скобки, появляется в меню вместе со стрелкой, указывающей вправо. Выбор такого пункта приводит к появлению следующего (подчиненного) меню справа от выбранного пункта исходного меню;

[<-] – указывает, что данный пункт является последним пунктом каскадного меню;

\ – указывает на необходимость ввести в данном месте некоторое значение.

В команды меню, содержащие этот знак, могут быть включены варианты ответов на команду.

$I – приводит к выводу поименованного графического меню;

+ – переводит макрос на следующую строку, если является последним символом в строке;

$(– включает в заголовке строковый макрос языка DIESEL;

– отключает (высвечивает серым) данный пункт меню;

& – поставленный перед некоторой буквой в названии пункта меню, означает, что она будет отображаться в меню подчеркнутой, и активизировать данный пункт можно будет нажатием клавиши с этой буквой;

!. – отмечает пункт меню специальным маркером (галочкой).

Любая строка в файле acad.mnu, начинающаяся с символов \\, является комментарием и игнорируется программой AutoCAD.

Заголовки пунктов внутри разделов ***POPn могут иметь произвольную длину. Ширина каждого падающего меню определяется самым длинным заголовком.

При создании нового падающего меню и добавлении к существующему меню AutoCAD рекомендуется сначала сделать копию файла acad.mnu с произвольным именем (например, acad_temp.mnu) и только затем модифицировать оригинал.

Файл меню acad.mnu, загруженный в текстовый редактор, необходимо прокрутить до начала раздела, озаглавленного ***Toolbars(Панели), который следует за разделом POP11, где описывается падающее меню Help (Помощь). Сразу после раздела POP11 можно вставить новый фрагмент, содержащий пользовательское меню. Например:

```
***POP12
**PP1
ID_myMenu [Новое падающее меню]
[--]
    [-> Уровень 1, строка 1]
          [-> Уровень 2, в строке 1, кнопка 1]
               [Уровень 3, в строке  1, кнопка 1 ]
               [->Уровень 3,в  строке  1, кнопка 2 ]
               [<-Уровень 3, в строке  1, кнопка 3 ]
          [ Уровень 2, в строке 1, кнопка 2]
          [<-Уровень 2, в строке 1, кнопка 3]
[--]
[-3
    [-> Уровень 1, строка 2]
          [Уровень 2,  в строке 2, кнопка 1]
          [-> Уровень 2,  в строке 2, кнопка 2]
               [Уровень 3,  в строке  2, кнопка 1 ]
               [Уровень 3,  в строке  2, кнопка 2 ]
               [<-Уровень 3,  в строке  2, кнопка 3 ]
          [<-Уровень 2, в строке 2, кнопка 3]
[--]
    [-> Уровень 1, строка 3]
          [Уровень 2, в строке 3, кнопка 1]
          [<- Уровень 2, в строке 3, кнопка 2]
 [--]
[--]
```

```
[--]
   [-> Уровень 1, строка 4]
         [Уровень 2, строке 4, кнопка 1]
         [Уровень 2, строке 4, кнопка 2]

         [<- Уровень 2, строке 4, кнопка 3]
```

Изменения, внесенные в файл acad.mnu следует сохранить. Далее измененный файл необходимо перекомпилировать. Для этого в командной строке AutoCAD нужно ввести команду ***_MENU (МЕНЮ)***, а затем в появившемся диалоговом окне Select Menu File (выбор файла меню) выбрать модифицированный файл acad.mnu.

9.4. Графическое меню

Графическое меню (рис. 9.1) позволяют выбирать команды или опции путем вывода на экран графического образа – слайда (или пиктограммы). Для графических образов используются файлы и библиотеки слайдов AutoCAD.

Графическое меню содержится в разделе ***IMAGE файла меню. Определяется оно так же, как и экранное меню: каждый пункт состоит из заголовка и последовательности команд, которая должна выполняться при выборе данного пункта. Так же, как и в разделах падающих меню, первая строка – это заголовок, отображаемый над группой пиктограмм, входящих в графическое меню и не используется в качестве возможности выбора. Ниже приведен фрагмент файла acad.mnu русскоязычной версии, соответствующий диалоговому графическому окну.

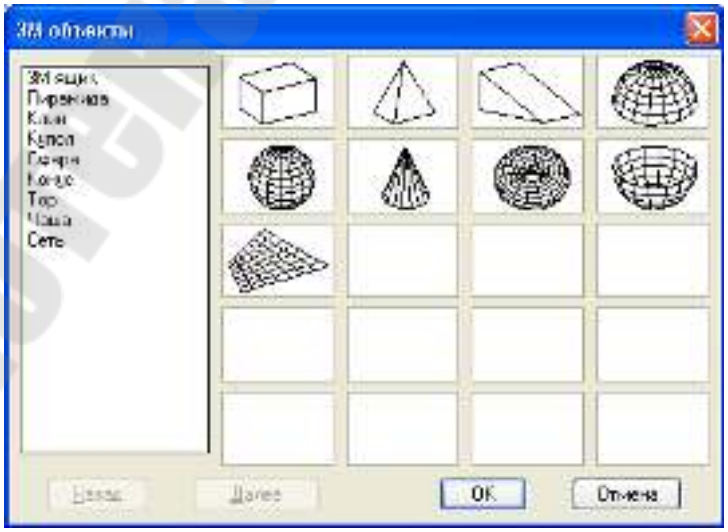


Рис. 9.1. Графическое меню
(вызывается Рисование/Поверхности/3М поверхности)

```
***IMAGE
**IMAGE_3DOBJECTS
[3D Objects]
[acad (Box3d, 3M ящик)] ^C^C_ai_box
[acad (Pyramid, Пирамида)] ^C^C_ai_pyramid
[acad (Wedge, Клин)] ^C^C_ai_wedge
[acad (Dome, Купол)] ^C^C_ai_dome
[acad (Sphere, Сфера)] ^C^C_ai_sphere
[acad (Cone, Конус)] ^C^C_ai_cone
[acad (Torus, Тор)] ^C^C_ai_torus
[acad (Dish, Чаша)] ^C^C_ai_dish
[acad (Mesh, Сеть)] ^C^C_ai_mesh
```

Графические меню могут быть полностью или частично заменены подменю, точно так же, как и другие разделы меню. Команда «$i» адресуется к графическому меню. Между подменю должна стоять пустая строка для очистки пунктов предыдущего меню. Графические меню могут содержать максимально 20 вариантов выбора (заметим, что заголовок не является вариантом выбора, следовательно, максимум 21 строка может быть в определении графического меню). Любые дополнительные строки игнорируются. Пиктограммы отображаются вместе с прокручиваемым полем списка, содержащим связанные с ним имена слайдов или другой текст до 17 символов длиной.

Допустимы следующие виды заголовков пунктов графического меню:

– [имя слайда] – в поле со списком выводится имя слайда, который отображается в виде пиктограммы;

– [имя слайда, заголовок] – в поле со списком выводится заголовок, а слайд отображается в виде пиктограммы;

– [библиотека (имя слайда)] – в поле со списком выводится имя слайда из библиотеки, а слайд отображается в виде пиктограммы;

– [библиотека (имя слайда, заголовок)] – в поле со списком выводится заголовок, а слайд из библиотеки отображается в виде пиктограммы;

– [blank] – в поле со списком в качестве заголовка используется разделительная строка, и никакой слайд не отображается;

– [текст] – если первым символом является пробел, то в поле со списком выводится заданный текст, и никакой слайд не отображается. Это позволяет установить в меню ссылки на родственные команды, а также создавать пункты без изготовления специальных слайдов.

Все пространство в диалоговом графическом окне разделено на ряд полей. Кнопки Previous (Назад), Next (Далее), OK (Да), Cancel (Отмена) добавляются к графическому меню автоматически. Если пунктов меню больше, чем может быть отображено в выводимом окне, доступ к пунктам, оставшимся вне зоны видимости, осуществляется при использовании соответствующих кнопок графического меню или полос прокрутки в поле со списком.

Выбор из графического меню позволяет делать специальная команда меню «**$**i =*», которая отображает графическое меню на экране. Эта команда может быть помещена в любой раздел меню, но не может быть введена с клавиатуры. Графическое меню отображается в диалоговой рамке с заголовком из первой строки, выровненной по центру в верхней части рамки.

После выбора пункта в графическом меню определенный в строке меню для данного текст логически заменяет команду «**$=***», которая активизировала меню. Это позволяет включать в графическое меню произвольное число команд. Представленный текст считывается программой управления меню и может содержать команды меню, в том числе «$i=». Таким образом можно конструировать иерархические графические меню, где выбор пункта приводит к отображению следующего графического меню и т. д. Вызов таких меню осуществляется последовательно.

Чтобы создать новое графическое меню, следует добавить новый пункт в падающее меню:

```
***POP12
**PP1
ID_myMenu [Новое падающее меню]
[--]
    [-> Уровень 1, строка 1]
            [-> Уровень 2, в строке 1, кнопка 1]
                [Диалоговое окно 1] ^C^C$i=Окно1 $i=*
      [-Уровень 3,в  строке  1, кнопка 2 ] ^C^C$i=Окно2 $i=*
           [<-Диалоговое окно 3 ] ^C^C$i=Окно3 $i=*
           [Уровень 2, в строке 1, кнопка 2]
           [<-Уровень 2, в строке 1, кнопка 3]
[--]
[--]
    [-> Уровень 1, строка 2]
           [Уровень 2,  в строке 2, кнопка 1]
```

```
            [-> Уровень 2,  в строке 2, кнопка 2]
        [Уровень 3,  в строке  2, кнопка 1 ] ^C^C$i=Окно4 $i= *
        [Уровень 3,  в строке  2, кнопка 2 ] ^C^C$i=Окно5 $i= *
      [<-Уровень 3,  в строке  2, кнопка 3] ^C^C$i=Окно1 $i= *
            [<-Уровень 2, в строке 2, кнопка 3]
[--]
    [-> Уровень 1, строка 3]
            [Уровень 2, в строке 3, кнопка 1]
            [<- Уровень 2, в строке 3, кнопка 2]
 [--]
[--]
[--]
    [-> Уровень 1, строка 4]
            [Уровень 2, строке 4, кнопка 1]
            [Уровень 2, строке 4, кнопка 2]
            [<- Уровень 2, строке 4, кнопка 3]
```

Затем следует описать само графическое меню.

Наиболее целесообразно использовать графическое меню для вставки блоков. В этом случае с окном графического меню необходимо связать команду INSERT (ВСТАВИТЬ). Фрагменты разрабатываемого графического меню следует размещать в файле acad.mnu после раздела ***IMAGE. Последний подраздел графического меню начинается с **IMAGE_VPORTI – именно после него и нужно разместить новый фрагмент. Например:

```
**IMAGE_VPORTI
[Tiled Viewport Layout]
……
**ОКНО1
[Заголовок диалогового окна]
[s11,Название элемента 1]^C^C(command "_.insert" "Элемент1" '(0 0) 1
1 0.0) [s12,Название элемента 2]^C^C(command "_.insert" "Элемент2"
'(0 0) 1 1 pause)
[s13,Название элемента 3]^C^C(command "_.insert" "*Элемент3"  '(0 0)
1 pause)
[s14,Название элемента 4]^C^C(command "_.insert"  "Элемент4" pause
1 1 pause)
[s15,Название элемента 5]^C^C_-insert "Элемент6";\  ;
[s16,Erase2]^C^C(c:erase2);\  ;
```

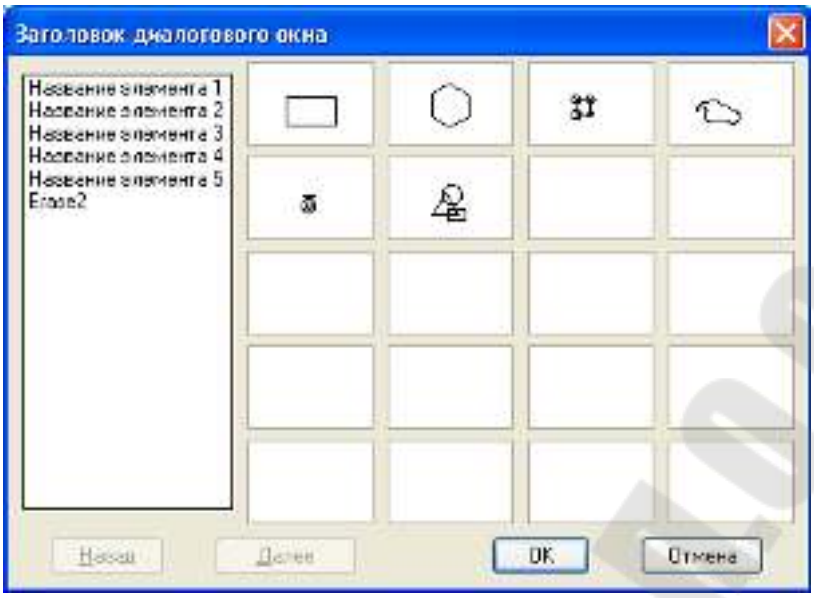


Рис. 9.2. Заголовок диалогового окна

Первая строка **Окно1 содержит имя нового диалогового графического окна. Запись [Заголовок диалогового окна] определяет тематический заголовок, который появляется в верхней части этого диалогового окна.

В следующих строках сообщается, какие слайды нужно отобразить и какие команды выполнить при выборе соответствующих окон графического меню. Например, в третьей строке:
[s11,Название элемента 1]^C^C(command "_.insert" "Элемент1" '(0 0) 1 1 0.0) сообщается, что слайд s11 отобразится в первом неперекрывающемся окне, а в поле со списком будет выведен заголовок Название элемента 1. Затем следует отмена любых других команд (^C^C) и команда INSERT (ВСТАВИТЬ), выполняющая вставку блока Элемент1. Все параметры команды INSERT (ВСТАВИТЬ) определены, поэтому блок будет вставлен в точку, имеющую координаты 0, 0 с коэффициентами масштабирования по оси X и Y, равными единице и углом поворота 0°.

В четвертой строке

[s12, Название элемента 2]^C^C (command “_.insert” “Элемент2 ” ‘(0 0) 1 1 pause)

точка вставки блока и коэффициенты масштабирования определены, а угол поворота объекта предлагаются ввести пользователю либо с клавиатуры, либо с помощью мыши.

В пятой строке

[s13, Название элемента 3]^C^C (command “_.insert” “*Элемент3” ‘(0 0) 1 pause)

устанавливается взорванный, т. е. расчлененный блок (на это указывает звездочка *, помещенная перед его именем), а, следовательно,

коэффициент масштабирования указывается одним параметром, сразу для осей X и Y. При этом во время установки блок на экране отображаться не будет.

Шестая строка

[s14, Название элемента 4]^C^C (command “insert” “Элемент4 ” pause 1 1 pause)

сообщает о том, что при установке блока необходимо указать точку вставки либо с клавиатуры, либо с помощью мыши, а также угол поворота. Поскольку в командной строке блок не взорван, то на экране компьютера будет отображаться динамическое положение устанавливаемого блока.

Седьмая строка

[s15,Название элемента 5]^C^C_-insert "Элемент6";\ ;

сообщает о том, что при установке блока необходимо указать точку вставки либо с клавиатуры, либо с помощью мыши.

Восьмая строка

[s16,Erase2]^C^C(c:erase2);\ ;

Для загрузки из диалогового графического окна команды, разработанной пользователем на языке AutoLISP, строка меню должна быть представлена в следующем виде:

[slide, Заголовок]^C^C(c:namefunction)

После компиляции модифицированного файла acad.mnu при последовательном выборе пунктов *Новое падающее меню ⇒Уровень 1, строка 1⇒ Уровень 2 в строке 1, кнопка* 1 на экране появится новое диалоговое графическое *Окно1*.

При создании диалоговых окон, обеспечивающих доступ к элементам информационной базы объектно-ориентированных систем автоматизации проектирования, конструктору удобно предоставить изображение деталей, расположенных в проекционной связи. Ниже предлагается фрагмент меню.

```
** Окно3
[*** деталь ***]
[blank]
[s21, Вид снизу]^C^C (Command “_.Insert” “Деталь1” pause 1 1 )
[blank]
[blank]
[s22, Вид справа]^C^C (Command “_.Insert” “Деталь2” pause 1 1 )
[s23, Вид спереди]^C^C (Command “_.Insert” “Деталь3” pause 1 1 )
[s24, Вид  слева]^C^C (Command “_.Insert” “Деталь4” pause 1 1 )
[blank]
[blank]
[s25, Вид сверху]^C^C (Command “_.Insert” “Деталь5” pause 1 1 )
```

Изготовление слайдов для графического меню ничем не отличается от их создания в AutoCAD. Для этого предназначена команда **_MSLIDE** (**ДСЛАЙД**). Однако в процессе работы следует придерживаться некоторых рекомендаций.

Во-первых, нужно по возможности делать слайды простыми, оформлять пиктограммы как упрощенные версии изображений, чтобы легче их идентифицировать.

Во-вторых, желательно рамку пиктограммы целиком заполнять изображением. Если картинка очень вытянута по горизонтали или вертикали, лучше расположить ее по центру экрана. Пиктограммы отображаются с соотношением 3:2 (три единицы в ширину на две единицы в высоту).

В-третьих, следует отказаться от заполнения сплошных тел, т. к. при выводе графического меню эта операция будет проигнорирована.

9.4. Создание собственных меню, отличных от acad.mnu

Необходимо выполнить следующую последовательность действий:

1. Переименовать файл acad.mnu, например, в it_menu. mnu.
2. Удалить практически все, оставив собственные меню:

```
//
//     файл меню AutoCAD - I:\it\lesson\lesson9\222\it_menu.mnu
//

***MENUGROUP=it_menu

***POP1
**PP1
ID_myMenu     [Новое падающее меню]
            [--]
            [-> Уровень 1, строка 1]
              [-> Уровень 2, в строке 1, кнопка 1]
                [Диалоговое окно 1]^C^C$I=it_menu.Окно1 $I=it_menu.*
                [Уровень 3,в  строке  1, кнопка 2 ]^C^C$i=Окно2 $i=*
                [<-Диалоговое окно 3 ]^C^C$i=Окно3 $i=*
              [Уровень 2, в строке 1, кнопка 2]
              [<-Уровень 2, в строке 1, кнопка 3]
            [--]
            [--]
```

```
          [-> Уровень 1, строка 2]
            [Уровень 2,  в строке 2, кнопка 1]
            [-> Уровень 2,  в строке 2, кнопка 2]
              [Уровень 3,  в строке  2, кнопка 1 ]^C^C$i=Окно4 $i=*
              [Уровень 3,  в строке  2, кнопка 2 ]^C^C$i=Окно5 $i=*
              [<-Уровень 3,  в строке  2, кнопка 3]^C^C$i=Окно1 $i=*
            [<-Уровень 2, в строке 2, кнопка 3]
          [--]
          [-> Уровень 1, строка 3]
            [Уровень 2, в строке 3, кнопка 1]
            [<- Уровень 2, в строке 3, кнопка 2]
          [--]
          [--]
          [--]
          [-> Уровень 1, строка 4]
            [Уровень 2, строке 4, кнопка 1]
            [Уровень 2, строке 4, кнопка 2]
            [<- Уровень 2, строке 4, кнопка 3]

***TOOLBARS

***IMAGE
**ОКНО1
[Заголовок диалогового окна]
[s11,Название элемента 1]^C^C(command "_.insert"  "Элемент1"  '(0 0)
1 1 0.0);\  ;
[s12,Название элемента 2]^C^C(command "_.insert"  "Элемент2" '(0 0)
1 1 pause )
[s13,Название элемента 3]^C^C(command "_.insert" "*Элемент3"  '(0 0)
1 pause)
[s14,Название элемента 4]^C^C(command "_.insert"  "Элемент4" pause
1 1 pause)
[s15,Название элемента 5]^C^C_-insert "Элемент6";\  ;
[s16,Erase2]^C^C(c:erase2);\  ;
//
//   Конец файла меню AutoCAD - I:\it\lesson\lesson9\222\it_menu.mnu
```

Как загружать и выгружать собственные меню?

Для этих целей необходимо воспользоваться ***Сервис/Адаптация/Меню***. Нажать кнопку *Обзор*, найти нужный файл меню. Кнопка *Загрузить* становится активной, загрузить. Перейти на

вкладку *Строка меню*, из списка *Группы меню*, выбрать свое меню (например, it_menu).

В списке *Активные меню* выбрать строку, перед которой будет вставлено новое меню, нажать кнопку *Добавит*ь.

Аналогично, выбрав ***Сервис/Адаптация/Меню***, можно выгрузить любое меню.

Обязательно необходимо указать путь, где находятся слайды и рисунки:

Сервис/Настройка/Файлы/Путь доступа к вспомогательным файлам

Нажать кнопку *Обзор*, а затем – кнопку *Добавить.*

9.5. Загрузка из диалогового графического окна команды, разработанной пользователем на языке AutoLISP

Для загрузки из диалогового графического окна команды, разработанной пользователем на языке AutoLISP, строка меню должна быть представлена в следующем виде:

[slide, Заголовок]^C^C(c:namefunction)

Автоматическая загрузка программ

Программы на AutoLISP будут загружаться автоматически при загрузке AutoCAD, если их поместить в файл под названием ACAD.LSP.

Например, создать файл acad.lsp, который выполняет две функции:

```
; Программа 10: Удаление предпоследнего элемента чертежа
(DEFUN C:ERASE2 ( )
(SETQ secondlast (ENTDEL (ENTLAST)))
(ENTDEL (ENTLAST))
(ENTDEL secondlast)
)
(defun  C:unline ( /  a)
(setq a (SSGET))
(IF (> (SSLENGTH a ) 1 )
(COMMAND "_pedit" a "_y"  "_j" a "" "_x" )
)
```

Необходимо указать путь доступа к этому файлу:

Выбрать ***Сервис/Настройка/Файлы/Путь доступа к вспомогательным файлам***

Нажать кнопку ***Обзор***, а затем кнопку ***Добавить.***

Чтобы файл acad.lsp загружался при открытии каждого рисунка, необходимо системной переменной ACADLSPASDOC присвоить значение 1.

Литература

1. Бугрименко, Г. А. Автолисп – язык графического программирования в системе AutoCAD / Г. А. Бугрименко. – Москва : Машиностроение, 1992. – 144 с.

2. Кудрявцев, Е. М. AutoLISP. Основы программирования в AutoCAD 2000 / Е. М. Кудрявцев. – Москва : ДМК Пресс, 2000. – 416 с.

3. Хювенен, Э. Мир Лиспа. В 2 т. Т. 2 / Э. Хювенен, Й. Сеппанен. – Москва : Мир, 1985.

4. Соколова, Т. Ю. AutoCAD 2004. Англоязычная и русская версии / Т. Ю. Соколова. – Москва : ДМК Пресс, 2004. – 600 с.

5. Мурашко, В. С. Использование языка AutoLISP для автоматизированного проектирования : лаб. практикум по курсу «Основы автоматизированного проектирования» для студентов специальностей 1-36 01 01 «Технология машиностроения» и 1-36 01 03 «Технологическое оборудование машиностроительного производства» днев. и заоч. форм обучения / В. С. Мурашко. – Гомель : ГГТУ им. П. О. Сухого, 2007. – 35 с.

Содержание

Учебное электронное издание комбинированного распространения

Учебное издание

Мурашко Валентина Семеновна

СИСТЕМЫ КОМПЬЮТЕРНОЙ ГРАФИКИ В АВТОМАТИЗИРОВАННОМ ПРОЕКТИРОВАНИИ

Курс лекций по одноименной дисциплине для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной формы обучения В двух частях Часть 1

ЯЗЫК AUTOLISP

Электронный аналог печатного издания

Редактор *Н. И. Жукова*
Компьютерная верстка *М. В. Аникеенко*

Подписано в печать 18.05.09.
Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 6,51. Уч.-изд. л. 6,08.
Изд. № 150.
E-mail: ic@gstu.gomel.by
http://www.gstu.gomel.by

Издатель и полиграфическое исполнение:
Издательский центр учреждения образования
«Гомельский государственный технический университет
имени П. О. Сухого».
ЛИ № 02330/0549424 от 08.04.2009 г.
246746, г. Гомель, пр. Октября, 48.