Федеральное агентство по образованию
Государственное образовательное учреждение
высшего профессионального образования
«Омский государственный технический университет»

Ф. Н. Притыкин

ПАРАМЕТРИЧЕСКИЕ ИЗОБРАЖЕНИЯ ОБЪЕКТОВ ПРОЕКТИРОВАНИЯ НА ОСНОВЕ ИСПОЛЬЗОВАНИЯ ЯЗЫКА АВТОЛИСП В СРЕДЕ АВТОКАД

Учебное пособие

Омск
Издательство ОмГТУ
2008

УДК 004.43 (075)
ББК 32.973.26–018.1я73
П77

Рецензенты:
Д. В. Сакара, канд. техн. наук, доцент, зав. каф. «Детали машин и инженерная графика» ОмГАУ
Ю. Ф. Савельев, канд. техн. наук, доцент, зав. каф. «Инженерная графика» ОмГУПС

Притыкин, Ф. Н.
П 77 **Параметрические изображения объектов проектирования на основе использования языка АВТОЛИСП в среде АВТОКАД**: учеб. пособие / Ф. Н. Притыкин. – Омск: Изд-во ОмГТУ, 2008. – 112 с.

ISBN 978-5-8149-0527-7

В учебном пособии описаны встроенные функции алгоритмического языка программирования АВТОЛИСП (более 70), а также функции STRIX, SAPKA, ACAD и др., разработанные автором совместно с канд. техн. наук, доцентом Ляшковым А. А. Приведены примеры применения указанных функций в решении некоторых практических задач.

Учебное пособие предназначено для студентов всех форм обучения специальностей 220401 «Мехатроника», 220200 «Автоматизация и управление» и 552800 «Информатика и вычислительная техника», изучающих дисциплину «Компьютерная графика», с целью приобретения практических навыков программирования на языке Автолисп при решении задач проектирования изделий и технологических процессов. Также пособие может быть рекомендовано специалистам, работающим в области разработки САПР различного назначения, и конструкторам-проектировщикам.

Печатается по решению редакционно-издательского совета ОмГТУ.

УДК 004.43 (075)
ББК 32.973.26–018.1я73

ISBN 978-5-8149-0527-7

ОГЛАВЛЕНИЕ

1. ВВЕДЕНИЕ В КОМПЬЮТЕРНОЕ ПРОЕКТИРОВАНИЕ ИЗДЕЛИЙ И ТЕХНОЛОГИЧЕСКИХ ПРОЦЕССОВ

1.1. Общий алгоритм проектирования изделий

Любой процесс конструирования изделий состоит из ряда последовательных шагов-действий, начиная от разработки замысла изделия до создания конструкторской документации на его изготовление. Укрупненно такой процесс можно представить следующим образом:

1) формирование внешнего вида изделия (эскизирование);

2) анализ прочностных характеристик материалов и элементов конструкций;

3) оптимизация конструкции с учетом результатов по первым двум пунктам;

4) технологическая проработка конструкции изделия;

5) создание экспериментальных образцов и натурные испытания.

Из приведенного перечисления шагов создания нового изделия следует, что процесс проектирования имеет сложный итерационный характер. Для его реализации свойственно неоднократное возвращение к началу проекта или к отдельным его составным частям. Поэтому вопрос автоматизации процессов проектирования всегда актуален для предприятий и конструкторских бюро. В связи с этим во многих конструкторских бюро внедряют автоматизированные системы проектирования САПР или CAD/CAE-системы. Без такого внедрения успешное развитие предприятий в настоящее время уже невозможно.

САПР (CAD/CAE) *– это система, позволяющая на базе ЭВМ автоматизировать определенные функции, выполняемые человеком с целью повышения темпов и качества проектирования.*

CAD *– это комплекс прикладных программ, обеспечивающих проектирование, черчение, трехмерное геометрическое моделирование деталей и сложных конструкций. Дополнительно применение CAD позволяет выполнять проектирование с элементами анимации (движения), визуализации и управление базами данных.* ***CAE*** *– компьютерное технологическое проектирование.*

Для функционирования САПР необходимо наличие:

1– технических средств – компьютеров, устройств ввода графической информации, плоттеров и т. п.;

2 – программных средств (пакетов программ графических систем и языков высокого уровня);

3– квалифицированных специалистов-проектировщиков, владеющих знаниями по работе с прикладными программами различного назначения.

1.2. Структура САПР. Организация информационных потоков в САПР

Решение задач конструирования и технологической подготовки производства с применением ЭВМ предполагает переход от реального технического объекта к его кодированному описанию в памяти вычислительной машины. Реальному объекту ставится в соответствие модель, над которой могут быть осуществлены преобразования с использованием вычислительной техники.

Внутримашинное представление проектируемого объекта задают в виде массива численных значений параметров и кодов, описывающих структуру объекта. На рис. 1.1 представлена схема организации информационных потоков при использовании компьютерного проектирования изделий.

Процесс проектирования начинают с выявления и описания потребностей заказчика и постановки задачи. Выявление потребностей предполагает установление кем-либо существования проблемы, в соответствии с которой должно быть предпринято то или иное корректирующее воздействие. Такой проблемой может быть выявление некоторого дефекта в конструкции эксплуатируемой машины. Постановка задачи включает в себя детальное описание изделия, подлежащего проектированию. Это описание должно содержать информацию о физических и функциональных характеристиках объекта проектирования. В соответствии с этим первым и вторым блоком схемы (рис. 1.1) должны быть блок входных данных о проектируемом изделии и блок обработки информации из блока 1. Впоследст-

вии эти данные постоянно будут использоваться в процессе конструирования.

Для создания информационной модели о проектируемом изделии необходимо построить внутримашинное представление об этой модели (блок 3). Информационная модель изделия состоит из совокупностей информационных моделей отдельных частей и деталей.

Информационная модель детали *– это совокупность сведений, однозначно задающих форму детали и другие данные, необходимые для её изготовления.* Информационная модель детали может быть задана системами уравнений линий и поверхностей, алгебраическими соотношениями, графиками, списками и таблицами. Информационная модель детали представляется элементами, отраженными на рис. 1.2. Как видно из рисунка, основными элементами при задании информационной модели детали служат точки, контуры, поверхности и элементарные тела.

На рис. 1.3а представлен граф, задающий информационную модель о детали, изображенной на рис. 1.3б. Следующими шагами проектирования являются синтез проектного решения, его анализ и оптимизация. Данные шаги тесно связаны друг с другом и многократно повторяются в процессе проектирования. На данном этапе происходят конструирование и геометрические вычисления – блок 4, с использованием различных баз данных о материалах, приспособлениях, инструментах и т. п. – блок 9. После конструирования необходимы осмысление и оценка решений – блок 5. Если спроектированное изделие не удовлетворяет заданным требованиям, то вновь изменяются параметры проектируемого изделия – блок 7. Проектируемое изделие, как правило, состоит из совокупности геометрических объектов. Поэтому для наглядной оценки принимаемых конструктором решений используют графический диалог конструктора с проектируемым изделием – блок 6. Графический диалог пользователя с проектируемым изделием возможен благодаря использованию средств ***компьютерной графики*** (рис. 1.1).

Компьютерная графика *– одна из подсистем САПР, необходимая для получения графической информации, её обработки и получения готовой документации в виде чертежей.*

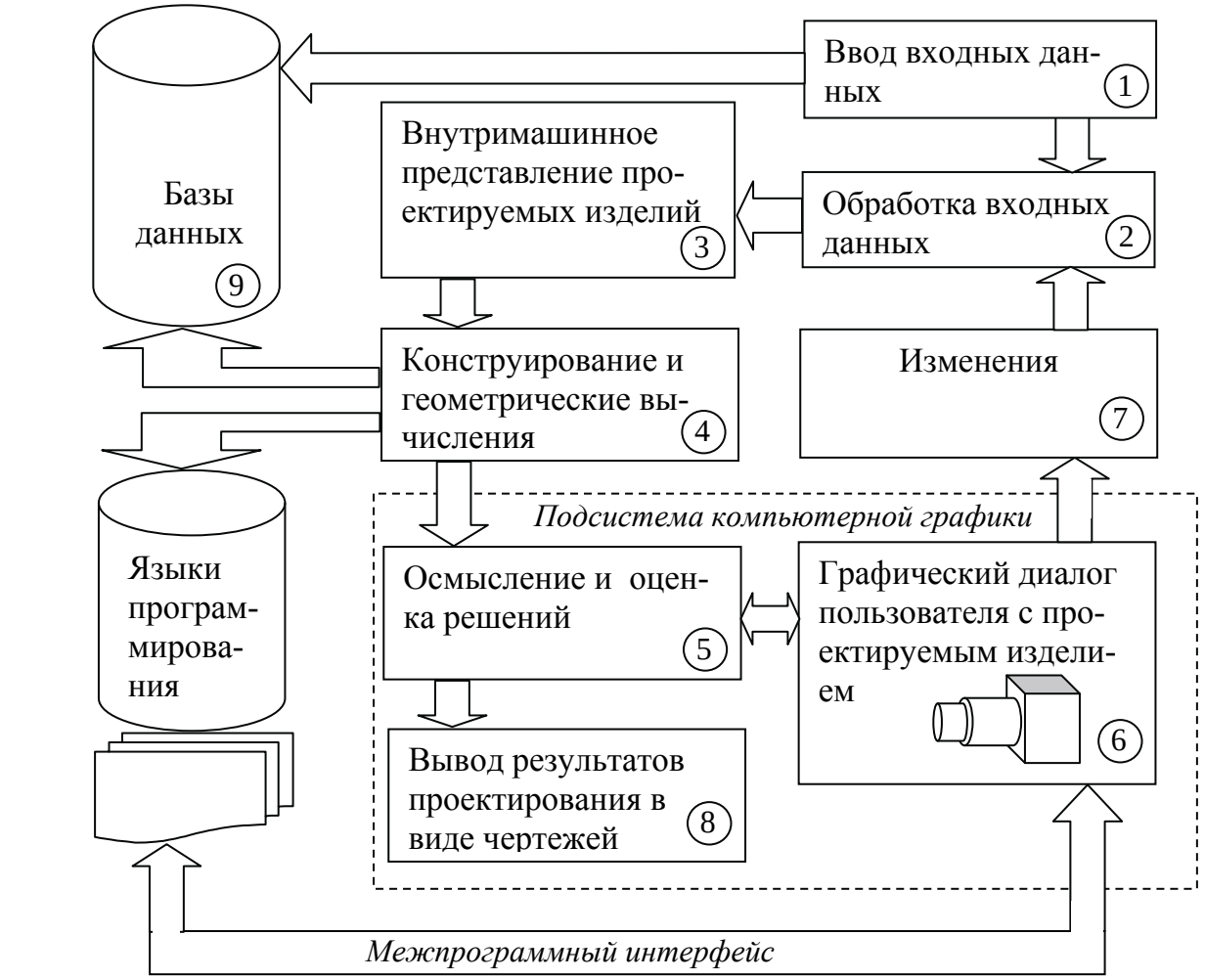


Рис. 1.1. Структура САПР

В процессе проектирования изделий одновременно с графическими системами (ACAD, КОМПАС, T-flex, Solid-Works и т. п.) могут быть использованы и алгоритмические языки программирования. Между ними программист или конструктор организует межпрограммный интерфейс.

Под ***графическими системами*** *будем понимать комплексы прикладных программ, имеющие специальные аппаратные средства для выполнения плоской двумерной графики и трехмерного геометрического моделирования.*

Межпрограммный интерфейс *– это реализованная на компьютере возможность организации обмена данными между различными автономно функционирующими пакетами прикладных программ.*

Интерактивный характер этапов конструирования и геометрических вычислений, а также осмысления и оценки решений проявляется в том, что вначале проектировщик определяет концептуальную основу конкретного компонента или узла создаваемого изделия, затем эта концепция подвергается анализу и усовершенствованию на основе базовых проектных решений.

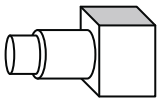
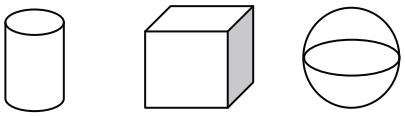
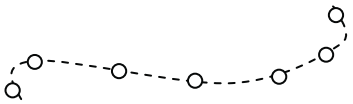
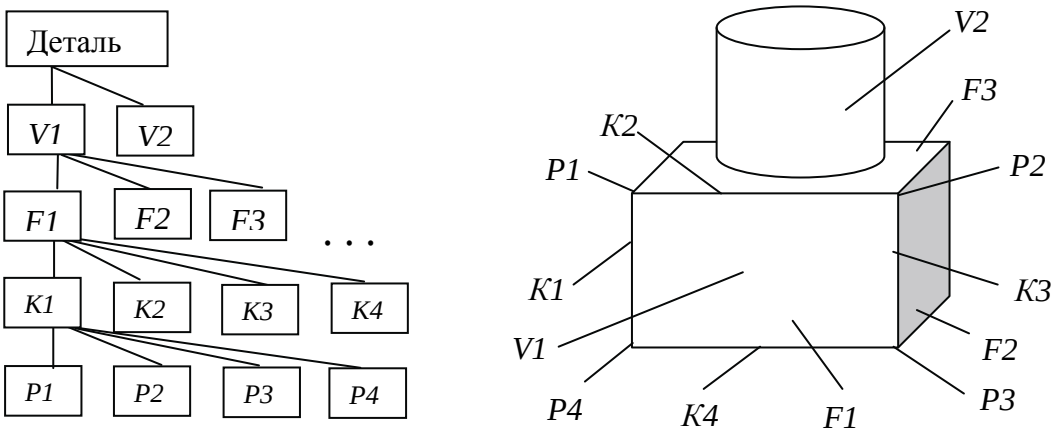
Группы элементов	
Элементарные тела – *V1, V2, …*	
Поверхности – *F1, F2, …*	
Контуры – *K1, K2, …*	
Точки – *P1,P2, …*	

Рис. 1.2. Составные элементы информационной модели детали

Этот цикл повторяют до тех пор, пока не будет получено требуемое оптимальное решение с учетом заданных проектных ограничений. Спроектированные компоненты и подсистемы синтезируют в рамках проектного решения, т. е. по всему изделию в целом с использованием аналогичных интерактивных методов.



а) б)

Рис. 1.3. Представление информационной модели детали:
а – граф, характеризующий информационную модель детали;
б – обозначение элементов модели

Заметим, что этап оценки связан с изменением проектных характеристик конкретного варианта и сопоставлением их с требованиями, установленными заранее на этапе постановки задачи. Заключительная фаза процесса проектирования – это представление результатов, которое предполагает документирование проекта с помощью чертежей, спецификаций, схем и т. п. (блок 8). Проект любой механической конструкции требует чертежей деталей, сборочных чертежей, которые получают в автоматизированном режиме с помощью средств компьютерной графики.

1.3. Преимущества разработки технической документации средствами компьютерной графики

Применение компьютерных графических систем для разработки технической документации даёт конструктору практически неограниченные возможности:

– разрабатывать чертежи-аналоги по чертежам-прототипам;

– создавать библиотеки изображений стандартных элементов (изображения крепежных элементов – болтов, шпилек, гаек и т. п.);

– моделировать трехмерные геометрические объекты с помощью объемных примитивов и операций, выполняемых с ними;

– адаптировать графическую систему к решаемым задачам пользователя путем расширения графической системы разработкой собственного меню пользователя и внедрения в систему языков программирования высокого уровня.

1.4. Анализ основных возможностей различных графических систем

В настоящее время рынок САПР представлен целым рядом программных продуктов. Наиболее распространённые из них – КОМПАС 3D-V9, ACAD2007, T-flex, Solid-Works и др. – работают в среде Windows, имеют открытый интерфейс [1]. Однако необходимо отметить, что наиболее универсальным инструментом для решения задач проектирования и технологической подготовки производства сегодня можно признать пакет программ САПР АВТОКАД. Это объясняется тем, что в постоянно развиваемой системе АВТОКАД не возникает больших затруднений использовать язык высокого уровня АВТОЛИСП [2–4]. Применение этого языка значительно ускоряет процесс разработки проектной документации и позволяет создавать специализированные меню в среде Автокад, новые команды графического редактора, осуществлять доступ к графической базе данных и моделировать её. Автолисп позволяет разрабатывать функции для решения самых разнообразных задач, а также создавать эффективные системы и подсистемы САПР, связанные с обработкой и анализом графической информации, получением готовой документации в виде чертежей. Помимо системы САПР АВТОКАД, использование алгоритмического языка программирования возможно в графической системе КОМПАС-3D-V8. Однако сочетающийся с КОМПАСом язык программирования ПИТОН не обладает столь разнообразными функциями, позволяющими выполнять работу с примитивами и обработкой различного рода графической информации.

Дополнительно заметим, что алгоритмический язык программирования Автолисп хорошо сочетается с машинной графикой.

Автолисп *– это подмножество языка Common Lisp, дополненное некоторыми функциями, отражающими специфику Автокада. Он позволяет выполнять не только расчеты, находясь в графическом редакторе, но*

и создавать подпрограммы для создания собственной необходимой и удобной среды проектирования. Программы на данном языке дают возможность в автоматизированном режиме получать параметрические изображения.

Параметрические изображения *– это изображения, состоящие из совокупностей примитивов, заданных узловыми точками, координаты которых могут быть рассчитаны в соответствии с наперед заданными геометрическими параметрами.* На рис. 1.4 приведены примеры параметрических изображений крановых блоков для различных диаметров канатов [5]. Данные изображения получены с использованием алгоритмического языка программирования АВТОЛИСП в среде АВТОКАД [5].

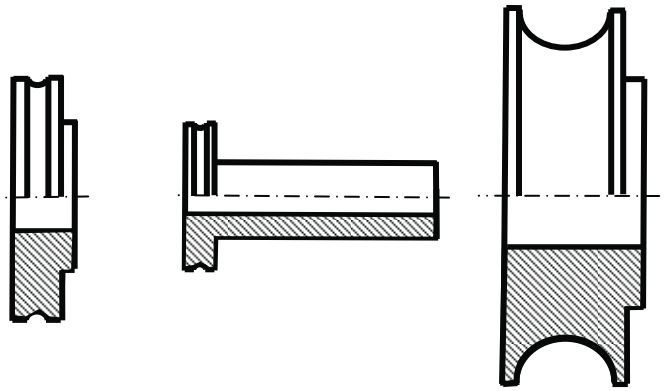


Рис. 1.4. Параметрические изображения, полученные на основе текстовой программы, описанной на языке АВТОЛИСП

1.5. Вопросы для самопроверки

1. Из каких основных логических блоков состоит процесс проектирования изделий различного назначения?

2. Как выстраивается структурно САПР изделий машиностроения?

3. С какой целью используют подсистему САПР-машинная графика?

4. Какими составными элементами представляют информационную модель деталей машиностроения?

5. Что такое межпрограммный интерфейс?

6. В чем преимущества получения технической документации средствами компьютерной графики?

7. С какой целью используют алгоритмический язык программирования АВТОЛИСП?

8. Какие изображения чертежей называют параметрическими?

2. ФУНКЦИИ, ПРЕДНАЗНАЧЕННЫЕ ДЛЯ ВЫЧИСЛЕНИЯ КООРДИНАТ ТОЧЕК ПАРАМЕТРИЧЕСКИХ ИЗОБРАЖЕНИЙ

2.1. Типы данных в АВТОЛИСП

Под данными понимают сведения, характеризующие объект, его состояние и операции, которые необходимо выполнять над этим объектом. Данные либо известны перед началом работы с объектом, либо их задают (вводят) в процессе работы.

Данные могут иметь различный вид и различное назначение. Им могут присваивать определенные значения постоянных или переменных величин. Для различения данных в Автолиспе используется понятие «***тип данных***». Под ***типом данных*** будем понимать возможные вид и значения, которые могут принимать переменные и константы алгоритмического языка программирования Автолисп. Для обозначения структурного элемента типа данных введем произвольно символ ai (i = 1, 2, ..., n) как некоторую переменную, которой будем присваивать определенные значения (табл. 2.1).

Таблица 2.1

Перечислим основные типы данных в Автолисп с примерами их значений:

№	Типы данных	Имя-символ	Пример задания конкретных значений
1	Целые переменные	a1	10;
2	Действительные переменные	a2	2.25 или 0.736;
3	Списки	a3	10 20 30 40 50;
4	Строковые константы	a4	"сталь Г13л";
5	Имена примитивов	a5	код примитива (60000048);
6	Встроенные функции	a6	(sl1) ;
7	Наборы примитивов	a7	список кодов примитивов;
8	Дескрипторы файлов	a8	(open «bolt»);

Для образования типов данных в Автолиспе применяют **имена-символы, числа** и **логические знаки** True (истина) и Fa1se (ложь). Они есть (суть) простейшие объекты Автолиспа.

Имена-символы состоят из букв, цифр и специальных символов. Имена-символы могут иметь обозначения a1, a7, c4 и т. п. Имена-символы не могут начинаться с цифры. Они могут обозначать различные лисповские объекты, другие символы или числа.

Имена-символы, задающие переменные и постоянные объекты Автолиспа, числа и логические знаки называют ***атомами.*** Из определения атома получаем возможные его виды: 1, 7, a1, True, 36, Fa1se и т. п. Из атомов создают другие структуры, в том числе и различные типы данных. Если имя-символ – переменная, то для установления, является ли переменная атомом или нет, применяют функцию atom:

(atom <переменная >). Если переменная – атом, то функция возвращает значение T(true), если нет, то – nil.

Для образования типов данных кроме атомов применяют ***«списки».*** ***Список*** *– это такой тип данных, который содержит набор разделённых пробелами атомов и вложенных (но не обязательно) в список подсписков.* Чтобы отличить список от атома в АВТОЛИСПЕ, применяют функцию listp. Функция в виде решения возвращает ответ либо T(true), если запись – список, и nil – в остальных случаях. Например, если a1 = (10 20), (listp a1) → True, если a1 = 10, (listp a1) → nil. Подсписок – это список, заключенный в круглые скобки и введенный в список как обособленный набор атомов. Примеры:

(x y z) – список состоит из трех символов-атомов.

(a (b c) d) – список состоит из двух символов-атомов и одного подсписка, состоящего из двух атомов. Список, в котором нет ни одного атома, называют пустым списком. Его обозначают () или NIL. Символ NIL имеет двойственный характер, т. к. он может быть как атомом, так и списком, не содержащим ни одного элемента.

Другой тип данных – ***строковые константы***. ***Строковые константы*** *определяют как значения переменных, задающих тексты.* Строковые константы состоят из символов, слов, обозначений, предложений и т. п. Форма строковых констант представляет последовательность символов-атомов, взятых в кавычки. Например: «Сталь 45».

Имена примитивов описаны ниже в п. 4.2.

Встроенные функции, например – (sl1). Их описание и назначение приведены в п. 3.3.

Наборы примитивов, состоящие из СП исков кодов примитивов, рассмотрены в п. 4.2.

Дескрипторы файлов *– это переменные, применяемые для открытия файлов с целью ввода или вывода информации,* рассмотрены в п. 5.4.

Для установления типа данных применяют функции:

int – целые числа;

real – числа с плавающей запятой;

str – строковые константы;

syn – символы;

subs – встроенные функции Автолиспа;

fname – имя примитива Автокада;

pckset – наборы примитивов Автокада.

Для определения типа данных переменной применяют функцию **type**: *(type <переменная>).*

2.2. Функции в АВТОЛИСП, назначение и типовое образование

АВТОЛИСП – это функциональный язык программирования. При его использовании все вычисления, преобразования и управления выполняют как с помощью встроенных функций, так и функций, разработанных пользователем при написании программы. В результате программа в целом представляет собой суперпозиции некоторых функций. В свою очередь, эта программа может быть использована как функция другими программами. К основным встроенным функциям можно отнести setq, list, defun**,** ... . К этим функциям можно обращаться непосредственно с командной строки Автокада. Когда вводится какая-либо группа знаков, интерпретатор командной строки ищет вводимое слово в списке команд Автокада. Если введенного слова нет в списке команд, то выдается сообщение об ошибке. Если же первым символом будет введена круглая открывающаяся скобка, то интерпретатор командной строки переходит в режим ввода выражения Автолиспа. Выход из этого режима осуществляется при вводе скобки, закрывающей вводимое выражение.

Встроенные функции Автолиспа вызываются как списки, у которых первый элемент – имя функции, а остальные – элементы, если они есть – аргументы.

Любую функцию в АВТОЛИСП записывают в круглых скобках. Запись выражения в скобках при этом есть элемент программы.

Например, в записи (list a b c d …) list – название функции, a b c d … – аргументы функции.

Функция, позволяющая присвоить переменной p1 значения координат $x = 10$, $y = 20$ при задании точки на плоскости, запишется так: (setq p1 (list 10 20)).

Для некоторых функций число аргументов строго определено, например (sin U), (cos U). Для других оно может быть произвольным (+ a b c d …). В последней функции минимальное количество аргументов для сложения не может быть меньше двух. Поэтому аргументы c и d функции (+ a b c d …) называют факультативными, то есть они могут как присутствовать в функции, так и нет.

Факультативные аргументы *– это аргументы, которые могут как присутствовать, так и нет в списке аргументов функции.* Условимся в дальнейшем при описании функций факультативные аргументы помещать в квадратные скобки, т. е. (+ a b [c] [d]). В данном примере факультативными аргументами являются c и d.

Задать имя функции или определить новую функцию можно с помощью функции defun АВТОЛИСПА. Она определяет функции посредством создания списка операторов программы. Список операторов в этом случае представляет собой замкнутую область локальных переменных. При вызове функции в эту область вначале передаются данные и выполняются операторы программы, после чего осуществляется передача данных обратно в среду АВТОЛИСПА – АВТОКАД.

Форма ее записи имеет вид

(**defun** <имя> [<список аргументов>] <тело функции>)

Defun определяет функции с именем <имя>, за которым в круглых скобках указывается <список аргументов>. Этот список состоит из аргументов, передаваемых в заданную функцию из других функций. Заме-

тим, этот список может быть пустым. В этом случае, если в функции не указывают ни аргументы, ни символы, после имени функции ставят пустые скобки.

Пример задания функции:

```
;функция преобразования углов в радианы, заданных в градусах
(defun dtr (u)
    (* pi (/ u 180))
)
```

В данном примере начало записи ;функция преобразования … есть комментарий функции (описание назначения функции); dtr – название функции; (u) – аргумент функции и строка (* pi (/ u 180)) – тело функции. Функция выполняет пересчет углов, заданных в градусах, в радианы, где pi – константа, равная 3,14159. Строка, записанная перед первой скобкой функции и начинавшаяся с точки с запятой, не исполняется и служит комментарием.

2.3. Функции присвоения

Функции языка Автолисп позволяют присваивать переменным их значения, другие выражения, задавать и определять значения системных переменных САПР Автокад.

Системные переменные *– это переменные, используемые для настройки и управления параметрами графической системы, каждая из которых представляется определенным типом данных.* Задание системных переменных осуществляют с помощью специальных функций (см. дальше п. 3.1). Рассмотрим основные функции присвоения.

- (**setq** <переменная1> <выражение1> <переменная2> <выражение2> ...) означает, что данная функция устанавливает в <переменую1> значение <выражения1>, в <переменную2> – значение <выражения2> и т.д. Переменные могут называться как угодно, при условии, что первый символ имени является буквой. Присвоив некоторой переменной f значение, можно в любое время посмотреть ее значение в командной строке Автокада. Задав восклицательный знак и переменную "!f", интерпретатор командной строки передаст имя переменной f АВТОЛИСПУ, который возвращает обратно значение переменной с этим именем. Так, запись (setq a 10) присваивает переменной <a> значение 10

и возвращает 10, а запись (setq b "R") переменной <b> присваивает значение строковой константы "R" и возвращает "R", т. е. b = "R".

• (**list** <выражение>...) применяют для присвоения некоторой переменной значений списка (например, координат двух- и трехмерных точек). Функция list, используя любое количество выражений, формирует из них строку и возвращает список. Пример: (list 11.2 23) – возвращает результат (11.2 23).

(setq p1 (list 12 28)) – присваивает переменной p1, определяющей точку в плоскости, значения ее координат: x = 12 и y = 28. Если в командной строке набрать: Command: !p1, то после этого в командной строке появится (12, 28, 0).

• (**setvar** <имя переменной> <значение>) используют для присвоения системной переменной Автокада указанное <значение>. При этом в записи функции setvar <имя переменной> заключают в кавычки.

Примеры: запись (setvar "DIMTXT" 5) – устанавливает размер шрифта размерных чисел, равный 5 (см. п. 3.1). "DIMTXT" – обозначение системной переменной, 5 – значение системной переменной;

Запись (setvar "FILLETRAD" 5) – устанавливает величину радиуса сопряжения, равную 5 ед.

• Для определения значения системной переменной используют функцию getvar: (**getvar** <имя переменной>).

Пример: пусть последний раз радиус сопряжения был задан величиной, равной 5, тогда функция (getvar "FILLETRAD") возвращает 5.

• Для извлечения координат x, y и z из списка координат точки, заданного функцией list, используют функции **car, cadr** и **caddr**.

Примеры: пусть точка p1 задана координатами: (setq p1 (list 10 20 30)). Тогда (car P1) – возвращает значение координаты $x = 10$, (cadr P1) – возвращает значение координаты $y = 20$, (caddr P1) – возвращает значение координаты $z = 30$.

• (**nth** <номер элемента списка> <список>) используют для извлечения заданного элемента из заданного списка. Пример:

(nth 10 r) – возвращает десятый элемент списка, заданного переменной r.

2.4. Функции, выполняющие арифметические операции

Ниже приведено описание функций, позволяющих выполнять различные арифметические операции:

- Функцию (**+** <число> <число> . . .) используют для вычисления суммы всех <чисел>, которые могут быть как целыми, так и действительными. Если все числа целые, то и результат представляет целое число.

Пример: (+ 4 7 234) – возвращает результат 245.

- Функцию (**-** <число1> <число2> . .) применяют для вычитания из первого <числа1> сумму остальных. Если задано одно число, то оно вычитается из 0.

Примеры:

(- 234 34 12) – возвращает результат 188,

(- 30.2) – возвращает результат: –30.2 .

- Функцию (***** <число> <число> . . .) используют для перемножения всех <чисел>.

Пример:

(* 2 3 12) – возвращает результат 72.

- Функцию (**/** <число> <число> . . .) применяют для деления первого числа на произведение остальных. Если чисел два, то функция делит первое <число> на второе и возвращает частное. Примеры:

(/ 32.4 4) – возвращает результат 8.1;

(/ 100 5 2) – возвращает результат 10.

- Функция (**max** <число> <число> . . .) предназначена для определения наибольшего числа из заданных чисел.

Примеры:

(max 32.4 4 19.4) – возвращает результат 32.4;

(max 100 5 2 -6) – возвращает результат 100.

- (**min** <число> <число> . . .) – функция определения наименьшего числа из заданных чисел.

Пример:

(min 132.1 4 -190.4) – возвращает результат -190.4 .

- (**sqrt** <число>) – функция для извлечения квадратного корня из <числа> и представления его как действительного числа. Пример:

(sqrt 64) – возвращает результат 8,0.

- (**exp** <число>) – функция для вычисления e в степени <число> ($E = e^x$).

Пример: (exp 1) – возвращает результат 2.71428, а (exp 2.2) – возвращает результат 9.025013.

- (**expt** <основание> <степень>). Эту функцию применяют для вычисления <основания>, возведенного в указанную <степень>. Результат вычисления – целое число, если обе переменные не целые. В других случаях – действительное число.

Пример: (expt 2.0 3) – возвращает результат 8.0, а (expt 2.32 2) – возвращает результат 5.382.

- (**log** <число>) – функция для вычисления натурального логарифма <числа>. Пример: (log 16.5) – возвращает результат 2.80336 .
- (**fix** <число>) используют для преобразования <числа> в целое. Дробная часть действительного числа отбрасывается.

Примеры: (fix 32.7) – возвращает результат 32.
(fix 100) – возвращает результат 100.

- (**float** <число>). Такую функцию используют для преобразования заданного <числа> в действительное. Примеры: (float 32.7) – возвращает результат 32.7.

(float 100) – возвращает результат 100.0 .

- (**cos** <угол>) – функция для вычисления косинуса <угла>, заданного в радианах. Пример: (cos 0.5) – возвращает результат 0.877543. Аналогично функция:
- (**sin** <угол>) используют для вычисления синуса <угла>, заданного в радианах. Пример: (sin 0.5) – возвращает результат 0.479426.
- (**atan** <число1> <число2>) используют для вычисления arctg (<число 1> / <число 2>) в радианах. Если <число2> не указано, то вычисляется arctg <числа1>. При этом область допустимых значений от - pi до + pi.

Пример: (atan 1) – возвращает результат 0.785398, а (atan 1 2) – возвращает результат 0.4621.

- (**abs** < число >) – функция вычисления абсолютного значения <числа>. Пример: (abs -123.34) – возвращает результат 123.34.

2.5. Функции определения параметров геометрических объектов

Проектируемое изделие, как правило, состоит из совокупности геометрических объектов. В процессе проектирования изделий необходимо определять параметры указанных объектов (длину, углы, точки пересечения и т. п.). Для этого в АВТОЛИСПЕ существуют следующие встроенные функции:

• Функция (**distance** <точка 1> <точка 2>) вычисляет расстояние между заданными точками и возвращает его величину.

Пример: (distance '(2.0 3.0) '(2 8)) – возвращает результат 5.0, где список '(2.0 3.0) определяет координаты x = 2.0 и y =3.0 первой точки и список '(2 8) определяет координаты x = 2 и y = 8 второй точки. Если координаты точек заданы переменными p1 и p2, каждая из которых есть список, состоящий из двух атомов, то расстояние l определится по следующей записи функций: (setq l (distance p1 p2)) (рис. 2.1).

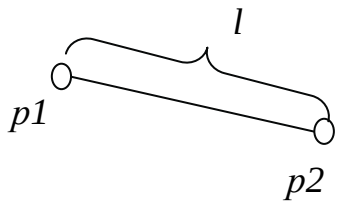


Рис. 2.1. Вычисление расстояния между точками

• Функция (**polar** <точка> <угол> <расстояние>) вычисляет координаты новой точки, находящейся от заданной <точки> на расстоянии <расстояние> под углом <угол>, который задается в радианах и измеряется против часовой стрелки относительно координатной оси x.

Пример:

(setq p2 (polar p1 u l)) – вычисляется положение точки p2 на плоскости в полярных координатах относительно точки p1 (рис. 2.2). Угол u измеряется от прямой d, принадлежащей точке p1 и параллельной оси x.

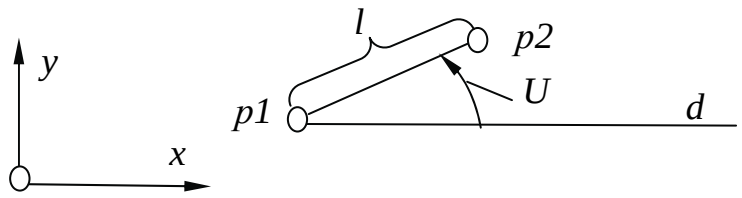


Рис. 2.2. Вычисление координат точки в полярных координатах

• (**angle** <точка1> <точка2>) – функция вычисления угла (в радианах), образованного вектором ***V***, направленным из <точки1> в <точку2> , и осью *x* (рис. 2.3). Этот угол измеряется против часовой стрелки.
Пример: (angle '(10 10) '(20 30)) – возвращает результат 1.10715. (setq u (angle p1 p2)) – если точки заданы переменными p1 и p2 (рис. 2.3). Точки p1 и p2 определяют направление вектора ***V.***

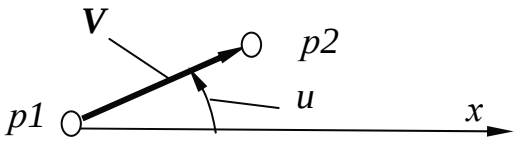
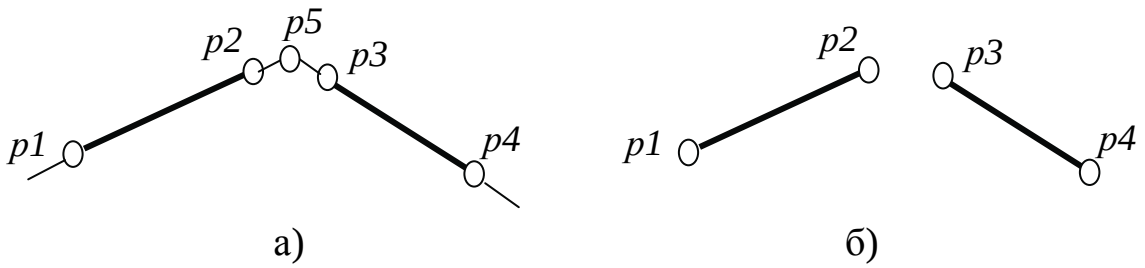


Рис. 2.3. Вычисление значения угла между направлением луча, заданного вектором ***V,*** и направлением оси *x*

• Функцию (**inters** <точка1> <точка2> <точка3> <точка4> [<C>]) применяют для вычисления точки пересечения двух отрезков, заданных точками: <точка 1> , <точка2> – отрезок1 и <точка3>, <точка4> – соответственно второй отрезок. Если факультативный аргумент <C> присутствует и равен nil, то отрезки воспринимают как отрезки бесконечной длины и координаты точки пересечения будут возвращаться всегда. В случае, если <C> отсутствует или не nil, то точка должна находиться на этих отрезках или функция возвращает nil.

Примеры: (inters '(10 10) '(20 30) '(20 20) '(30 60) (nil)) – возвращает результат (13.3333 16.6667), где списки '(10 10) '(20 30) '(20 20) '(30 60) задают координаты четырех точек. Если точки, определяющие прямые, заданы переменными p1, p2, p3 и p4, то точка p5 пересечения этих прямых вычисляется по следующей записи функции: (setq p5 (inters p1 p2 p3 p4 nil)) (рис. 2.4а). Переменной p5 будет возвращено значение координат

точки пересечения бесконечных прямых, проходящих через точки p1, p2 и p3, p4. Если запись имеет вид (setq p5 (inters p1 p2 p3 p4)), то переменной p5 будет возвращено значение nil, так как отрезки, заданные точками p1, p2 и p3, p4, не пересекаются (рис. 2.4б).



а) б)

Рис. 2.4. Вычисление координат точки пересечения:
а – двух прямых, б – двух отрезков прямых

2.6. Логические и специальные функции в АВТОЛИСПЕ

Структура Автолиспа содержит логические функции в виде условных выражений. Данные функции позволяют выполнять некоторые действия над зависимостями при выполнении заданного условия, если оно истинно T(true). Это достигается с помощью функции if.

- Функцию (**if** <условие> <выражение - тогда> [<выражение - иначе>]) применяют для проверки условия и если оно True (истина), то выполняется действие <выражение - тогда>, в противном случае – <выражение - иначе>. В общем случае выражение <выражение - иначе> может отсутствовать.

Если при составлении текста выражений 1 или 2 используют более одной функции, то необходимо ввести функцию – progn. Пример применения функции progn:

```
(if (< k 0)(progn  (setq x 10 y 20) (command "отрезок"  P1 P2 ""  )
                )
)
```

В данном примере, если выполняется условие (< k 0), то выполняется следующая последовательность: вначале функция setq присвоит значения переменным x = 10, y = 20, а затем будет построен отрезок, проходящий через точки p1 и p2.

В условных выражениях используют ряд специальных функций, которые рассмотрены ниже. Назначение этих функций – выполнять своеобразный контроль каких-либо объектов АВТОЛИСПА путем сравнения их.

• Функция (**minusp** <элемент>) возвращает T(True), если <элемент> действительное или целое число, имеющее отрицательное значение. Если число положительное, то функция возвращает nil.

Примеры:

(minusp 5) – возвращает nil,

(minusp -5) – возвращает T(True).

• Функция (**numberp** <элемент>) возвращает T(True) для элемента, являющегося целым или действительным числом, в противном случае – nil. Примеры:

(numberp 5) – возвращает T,

(numberp "R") – возвращает nil.

• Функция (**zerop** <элемент>) возвращает T(True), если <элемент> равен нулю, в противном случае – nil.

• Функцию (**=** <атом1> <атом2> . . .) применяют для проверки на условие равенства все <атомы>, и если атомы эквивалентны, то возвращается T(True), в противном случае – nil.

Примеры:

(= 5 5) – возвращает T(True), (= 5 25) – возвращает nil.

• Обратная этой функции (**/=** < атом1 > < атом2 > . . .) – возвращает T(True) если <атом1> не эквивалентен <атому2>, в противном случае – nil.

• Функция (**<** <атом1> <атом2> . . .) выполняет сравнение "меньше чем", и если каждый предыдущий <атом> меньше последующего, то она возвращает T(True).

Примеры:

(< 5 15 19) – возвращает T(True), (< 125 23) – возвращает nil.

• Функция (**<=** <атом 1> <атом2> . . .) возвращает T(True), если каждый предыдущий <атом> меньше или равен последующему, в противном случае – nil.

• Функция (**>** <атом 1> <атом 2> . . .) выполняет сравнение "больше чем", и если каждый предыдущий <атом> больше последующего, то она возвращает T(True), иначе – nil.

Примеры: (> 25 13 15) – возвращает T, (> 25 225) – возвращает nil.

• Функцию (**>=** <атом1> <атом 2> . . .) вводят для сравнения "больше чем". Она возвращает T(True), если каждый предыдущий элемент больше или равен последующему, в противном случае функция возвращает nil.

• (**not** <элемент>). Если <элемент> nil, то такая функция возвращает T(True), иначе - nil.

• Функцию (**and** <выражение> . . .) используют для выполнения логического И над списком выражений.

Примеры:

1 (setq x 10 y 20) – переменным *x* и *y* присвоены значения 10 и 20 соответственно, (if (and (= x 10) (= y 20)) (setq p1 (list x y))) – возвращает список (10 20), состоящий из значений переменных *x* и *y*, т. к. заданное условие истинно (T(True)).

2 (setq x 10 y 20)
(if (and (= x 20) (= y 20)) (setq z (list x y)) (setq z (+ x y))) возвращает число 30, т. к. заданное условие F (False).

В первом примере в функции (if (условие) (выражение-тогда)) отсутствует факультативный аргумент. Во втором примере (if (условие) (выражение-тогда) (выражение-иначе)) факультативный аргумент присутствует и так как заданное условие возвращает F(False), выполняется выражение-иначе.

• Функцию (**eq** <выражение1> <выражение2>) применяют для определения идентичности <выражение1> и <выражение2>. Она возвращает T, если оба выражения идентичны, иначе – nil. С её помощью устанавливают, являются ли два списка одним или нет. Пример:

(eq 'x (car '(x y)) – возвращает T(True).

(eq 'x (cadr '(x y)) – возвращает nil. Смотрим пример использования функций car и cadr в п. 2.3.

• Функция (**equal** <выражение1> <выражение2> [<допуск>]) проверяет условие равенства значений заданных выражений. Факультативный

аргумент <допуск> задает максимальную точность, на которую могут отличаться <выражение1> и <выражение2>. Примеры:
Допустим, что (setq a1 7.323) (setq a2 7.320). Тогда
(equal al a2) – возвращает nil,
(equal a1 a2 0.001) – возвращает T(True).

- Функция (**while** < тест - выражение1 > < выражение2 > . . .) вычисляет <выражение2> и другие до тех пор, пока <тест-выражение1> не станет nil. Функция while возвращает последнее значение последнего <выражения>.

Пример: пусть требуется вычислить сумму целых чисел от 1 до N.

Решение: Зададим начальное значение суммы и величину первого числа: (setq sum 0 M 1). Тогда (while (<= M N) (setq sum (+ sum M) (setq M (+ M 1)))
переменной sum будет присвоено значение, являющееся суммой последовательности из N целых чисел (значение N также должно быть предварительно задано).

- (**repeat** <число> <выражение> ...) – применяют для повторения вычислений. Функция выполняет <выражение> заданное <число> раз и возвращает результат последнего выражения. В данной функции может быть несколько выражений, а параметр <число> представляется любой положительной величиной.

Пример: пусть (setq b 100), тогда выражение
(setq sum (repeat 4 (setq b (+ b 10))) – возвращает 140.

2.7. Вопросы и задачи для самопроверки

1. Назовите основные типы данных языка программирования Автолисп.

2. Сформулируйте методику записи и составления функций Автолиспа.

3. Назовите функции, позволяющие проводить операции присваивания.

4. Назовите основные функции, позволяющие выполнять арифметические операции.

5. Назовите геометрические функции, используемые при расчете координат узловых точек параметрических изображений.

6. Как оформляются логические функции в Автолиспе?

7. Назовите основные функции, позволяющие проводить контроль объектов Автолиспа путем сравнения.

Задача 2.1

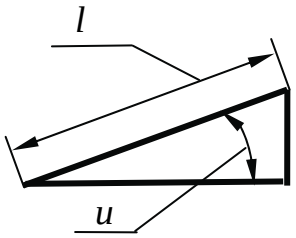


Рис. 2.5

Составьте фрагмент текста программы для вычисления периметра прямоугольного треугольника, если длину гипотенузы и один из углов определяют переменные *l* и *u*. Длины катетов имеют обозначения k1 и k2, а периметр per (рис. 2.5).

Задача 2.2

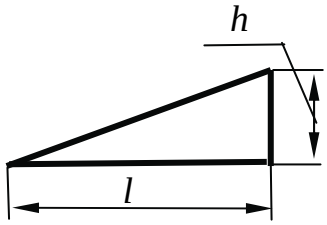


Рис. 2.6

Составьте текст программы, позволяющей вычислять гипотенузу прямоугольного треугольника (имеющего обозначение *gp),* размеры катетов которого задают переменные *h* и *l* (рис. 2.6).

Задача 2.3

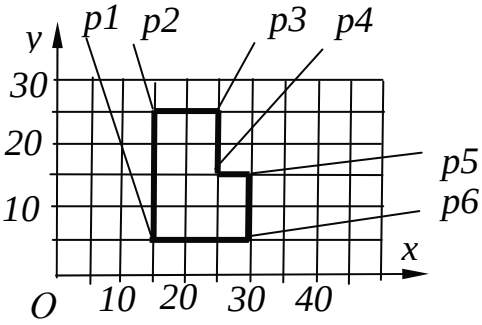


Рис. 2.7

Составьте фрагмент текста программы, позволяющий задать значения переменным p1, p2, p3, p4, p5 и p6, обозначающим точки плоского контура. Указанным переменным задать значения координат x и y с помощью функций list, используя координатную сетку, представленную на рис. 2.7.

Задача 2.4

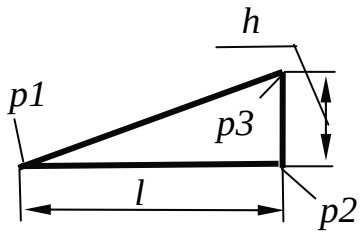


Рис. 2.8

Составьте фрагмент текста программы, осуществляющей расчет узловых точек p1, p2 и p3 параметрического изображения прямоугольного треугольника, при различных значениях переменных *l*, *h*. Геометрический смысл переменных ясен из рис. 2.8. Точка p1 имеет координаты x = 50, y = 50. Для расчета точек p2 и p3 использовать функцию polar.

Задача 2.5

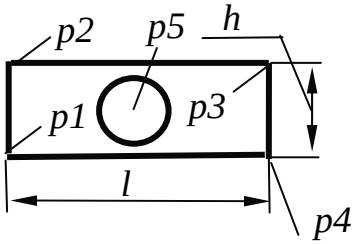


Рис. 2.9

Составьте фрагмент текста программы с использованием логической функции, позволяющей расчет координат центра окружности с радиусом 10 мм. Центр окружности располагается внутри прямоугольника рис. 2.9. Окружность может иметь минимальное удаление от сторон прямоугольника, равное 2 мм. Если значения переменных *l* и *h* не обеспечивают заданное минимальное расстояние до сторон прямоугольника, то координаты центра окружности точки p5 принимают значения nil. Вершины прямоугольника задают переменные p1, p2, p3 и p4. Точка p1 имеет координаты x = 50, y = 50.

3. ФУНКЦИИ, ОБЕСПЕЧИВАЮЩИЕ РЕАЛИЗАЦИИ КОМАНД СИСТЕМЫ АВТОКАД

3.1. Настройка среды черчения и оформление размерных стилей чертежей с использованием текстовых программ

Для доступа из АВТОЛИСПА к командам АВТОКАДА используют функцию command. Форма записи для ее вызова имеет вид

• (**command** <аргумент> "..." . . .), где <аргумент> – имя вызываемой из Автокада команды, а "..." здесь и далее показывают, что функция может иметь дополнительные аргументы в виде числовых значений или опций этой команды. Имена команд и опции представляют как строковые константы, а точки (если присутствуют) – как списки из двух или трех чисел. Если после функции command не указывается никакой информации, это равносильно нажатию пробела на клавиатуре. Если в качестве аргумента команды, которая вызывается функцией command, встречается ключевое слово PAUSE, то функция command приостановит свое действие, чтобы пользователь ввёл необходимое значение. Пауза длится до момента окончания ввода допустимого аргумента.

При построении изображений чертежей вначале необходимо задать совокупность параметров, определяющих среду черчения. Данные параметры определяют стиль текста, стиль размеров, совокупность слоев и др. Данные параметры могут быть установлены с помощью специальной функции, которую может создать пользователь. Эту функцию условно назовем SAPKA. Ниже приведены фрагменты указанного файла, отражающие примеры задания стиля текста и слоев, используемых при получении изображений чертежей и др. [1]. Стиль текста определяет тип, угол наклона и высоту шрифта.

• ;задание стиля текста

```
(command "style" "russ"  " russ " "0" "1" "15" "" "" )
```

1 1 2 3 4

1 – задание файла шрифта [1]; 2 – задание высоты текста (см. рис. 3.6);

3 – задание коэффициента сжатия текста; 4 – задание угла наклона текста;

"" – пустые кавычки соответствуют нажатию клавиши ENTER.

Под слоем понимают прозрачные плоскости, содержащие определённую информацию, из которой состоит чертеж. На отдельном слое можно размещать информацию, объединенную по какому-либо признаку (цвет линий, тип линий).

- ;задание размеров чертежа формата А3, имеющего горизонтальную
  ;ориентацию
  (command "Limits" "0,0" "420,297")

1 – задание координат левого нижнего угла формата;

2 – задание координат верхнего правого угла формата;

- ;удаление изображений примитивов, находящихся
  ;на графической зоне

(command "ERASE" "WINDOW" (getvar "VSmin") (getvar "VSmax") "" "redraw")

1 – задание опции WINDOW (Окно); 2 – задание левого нижнего угла «виртуального экрана» текущего видового экрана; 3 – задание правого верхнего угла «виртуального экрана» текущего видового экрана; 4 – ввод команды "redraw" (перерисовать).

- ;установление масштаба изображения, позволяющего поместить
  ;весь чертеж на графической зоне
  (command "Zoom" "all")
- ;задание нового слоя, имеющего имя 1 и цвет линий - синий
  (command "LAYER" "M" "1" "C" "5" "1" "")

1 – задание опции Make (Создай) [1]; 2 – задание имени слоя;

3 – задание опции Color (Цвет); 4 – задание номера цвета
(5 – синий цвет); 5 – указание имени слоя.

- ;задание нового слоя, имеющего имя 2, цвет синий и имеющего
  ;тип линии "hidden" (для построения изображений штриховых линий).

```
(command "LAYER" "M" "2"  "C" "5" "2" "LTYPE"  "hidden"  "2" "")
                                  1        2     3
```

1 – задается опция Ltype (Типлинии) [1]; 2– название типа линии (указанный тип линии hidden используется для получения изображений штриховой линии, т. е. линий невидимого контура; 3 – имя слоя.

Другие примеры создания слоев с различными типами линий и цветами

- ;задание нового слоя, имеющего имя 3, цвет синий, имеющий
  ;тип линии "dashdot" (для построения изображений штрихпунктирных
  ;линий).
  (command "LAYER" "M" "3" "C" "5" "3" "LTYPE" "dashdot" "3" "")
- ;задание нового слоя, имеющего имя 4 и цвет зеленый
  ;(для построения изображений размерных линий)
  (command "LAYER" "M" "4" "C" "3" "4" "")
- ;создание функций для установки слоёв
  (defun sl0()(command "LAYER" "S" "0" "")) ;изображение основных линий

```
                1   2
```

1 – задание опции Set (Установи); 2 – задание имени слоя.

```
(defun sl1()(command "LAYER" "S" "1" ""));функция для установки слоя 1
;изображающего тонкие линии
(defun sl2()(command "LAYER" "S" "2" "")); функция для установки слоя 2
 ;изображающего штриховые линии
(defun sl3()(command "LAYER" "S" "3" "")); функция для установки слоя 3
 ;изображающего штрихпунктирные линии
```

Оформление размерных стилей в графической системе АВТОКАД осуществляют с помощью системных размерных переменных, которые задают размерные стили (рис. 3.1, 3.2).

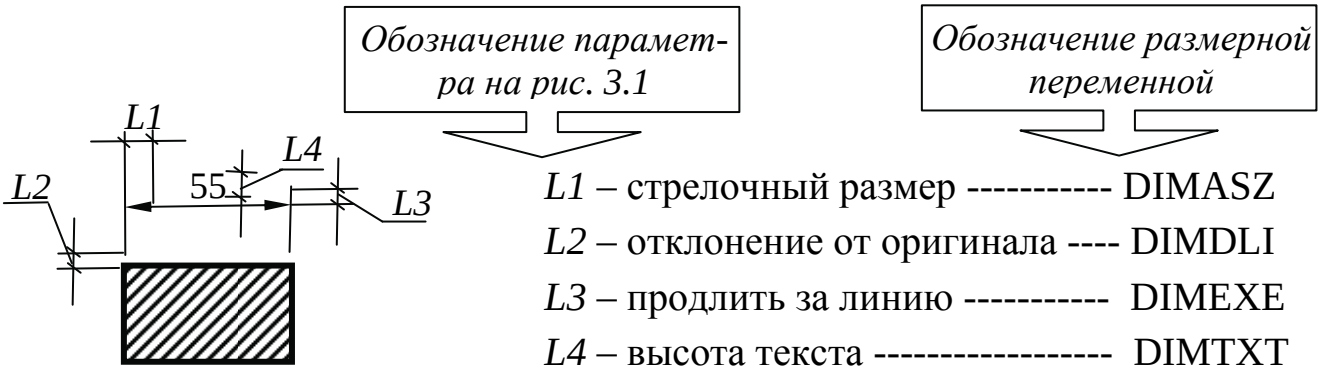


Рис. 3.1. Графическое представление некоторых системных размерных переменных

Системная размерная переменная DIMTAD → 1 позволяет изображать текст в разрыве или над размерной линией (рис. 3.2а). Размерную переменную DIMTIH используют для задания ориентации текста относительно размерной линии (рис. 3.2б). Если DIMTIH принимает значение Off (откл), положение текста определяется ориентацией размерной линии, в противном случае, если значение On – текст имеет всегда горизонтальную ориентацию.

а) б)

Рис. 3.2. Графическое представление системных размерных переменных DIMTAD и DIMTIH

```
• ;Пример задания значений системных размерных переменных
  ;в текстовой программе на языке AutoLISP:
(SETVAR  "DIMASZ" 5.)(SETVAR "DIMTXT" 5.)
(SETVAR  "DIMEXE" 2.)(SETVAR "DIMDLI" 0.)
(command "DIM" "DIMTAD" 1 "DIMTIH" "OFF" "EXIT")
```

3.2. Реализация команд системы АВТОКАД с помощью программ на языке АВТОЛИСП

При получении параметрических изображений вначале выполняют расчет узловых точек данных изображений, которые в примерах ниже имеют обозначения p1, p2, … и т. п. Далее по расчетным узловым точкам необходимо построить в среде АВТОКАДА параметрические изображе-

ния видов, разрезов, сечений деталей и конструкций. Для этого необходима реализация команд построения примитивов АВТОКАДА с помощью программ, описанных на языке АВТОЛИСП.

Примитив *– это любой графический элемент, например отрезок, окружность, дуга и т.п. , построенный с помощью реализации команд АВТОКАДА* [1].

Для того, чтобы самостоятельно записать строку программы позволяющую реализовать команду построения нового примитива, надо войти в графический редактор Автокада и средствами указания подробно отработать последовательность запросов графической системы по этой команде. Например, при вводе команды hatch (штрих) графическая система вначале запрашивает тип штриховки, масштаб, угол штриховки и примитивы, задающие замкнутый контур, который заштриховывается. В соответствии с данными запросами записывается строка программы в Автолиспе реализующая эту команду.

Ниже приведены примеры такой реализации по наиболее часто встречающимся командам при построении изображений чертежей с записью в файлах с расширением .dwg.

- Пример 1. (command "line" p1 p2 p3 "" "line" p4 p5 "") – построение отрезков прямых, где переменные p1, p2, p3, p4 и p5 определяют точки отрезков (рис. 3.3).

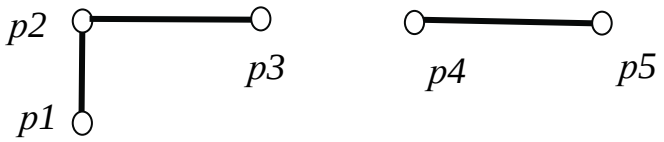


Рис. 3.3. Реализация команды line

- Пример 2. (command "pline" p1 "w" "5" "" p2 p3 "") – построение **ломаной,** состоящей из отрезков прямых заданной толщины. Переменные p1, p2, и p3 определяют точки отрезков, "w" – задает опцию width (ширина). В данном примере ширина равна 5мм (рис. 3.4).

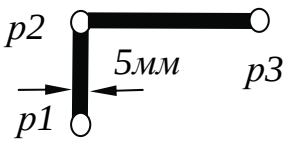


Рис. 3.4. Реализация команды pline

Внимание! – все приведенные команды **действуют лишь в английской** версии АВТОКАДА, чтобы работало на любой версии, следует к команде добавить слева "_.", а к параметру "_". Например "line" → "_.line" "all" → "_all".

• Пример 3. (command "circle" p1 r2 "circle" p2 (/ d 2)) – построение окружностей, где p1 и p2 определяют центры окружностей, r2 и выражение (/ d 2) задают радиусы окружностей (рис. 3.5).

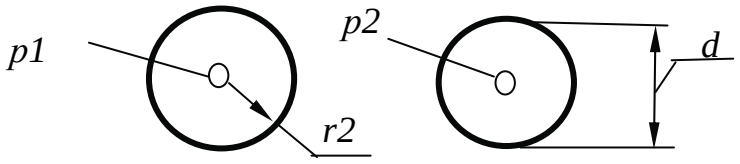


Рис. 3.5. Реализация команды circle

• Пример 4. (command "arc" "c" p1 p2 p3) – построение дуги окружности по центру и двум точкам, где "c" – задает опцию центр, p1 – центр дуги окружности, p2 и p3 – начальная и конечная точки дуги (рис. 3.6а).
(command "arc" p4 p5 p6) – построение дуги окружности по трем точкам (рис. 3.6б).

а) б)

Рис. 3.6. Реализация команды arc

• Пример 5. По записи команды (command "text" p1 "10" "0" "Неуказанные радиусы – 3мм") – выполняется построение изображения текста, где p1 – базовая точка текста, "10" – высота текста, "0" – угол наклона текста. Изображаемый текст "Неуказанные радиусы – 3мм" приведен на рис. 3.7.

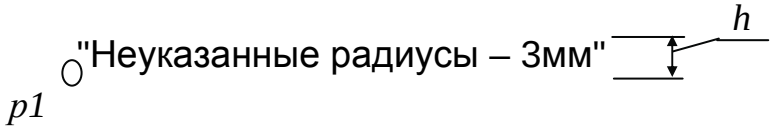


Рис. 3.7. Реализация команды text

- Пример 6. (command "fillet" "r" r1) – задание размера радиуса сопряжения. Значение радиуса сопряжения принимает значение переменной, имеющей обозначение r1. (command "fillet" p1 p2) – построение сопряжения между двумя отрезками прямых, точки p1 и p2 задают произвольные точки, принадлежащие сопрягаемым прямым (рис. 3.8).

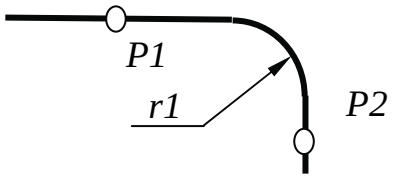
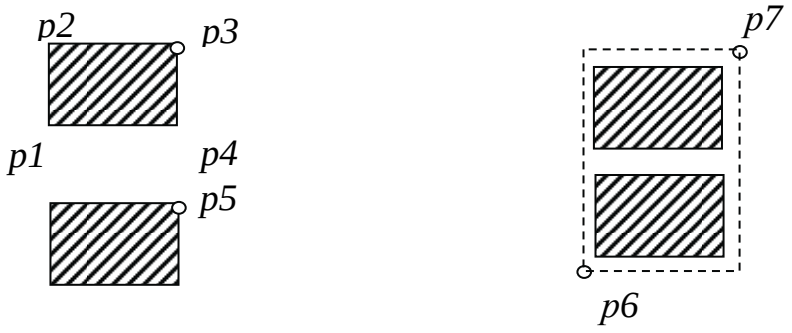


Рис. 3.8. Реализация команды fillet

- Пример 7. Перед выполнением операции штриховки необходимо выполнить изображение замкнутого контура с помощью команды pline. Например: (command "pline" p1 p2 p3 p4 p1 ""). В данном случае выбор замкнутого контура, подлежащего штриховке, делается с помощью указания одной точки.

  (command "hatch" "jis_wood" "5" "0" p3 p5 "") – построение штриховки с выбором замкнутых контуров, определяемых точками, принадлежащими этим контурам (рис. 3.9а). "jis_wood " – определяет тип штриховки, "5" – масштаб штриховки, "0" – угол наклона штриховки, p3, p5 – точки, необходимые для выбора объектов, определяющих замкнутые контуры.

  (command "hatch" "jis_wood" "5" "0" "w" p6 p7 "") – построение штриховки с выбором замкнутого контура с использованием рамки (рис. 3.9б). "w" – задание опции выбора объектов с помощью рамки; p4, p5 – точки, задающие вершины рамки по диагонали.



а б

Рис. 3.9. Реализация команды hatch

При составлении программы, позволяющей наносить размеры, необходимо вначале задать значения системных размерных переменных. Нанося размеры на чертеже, необходимо рассчитывать вспомогательные точки, определяющие положение размерных линий и примитивов точки – *p3, p4* и *p5* (рис. 3.10 а, г).

Ниже даны примеры формирования изображений размеров на чертеже.

• Пример 8. (command "dim" "horiz" p1 p2 p3 "" "exit") – нанесение горизонтального размера. Значение размерного текста определяется длиной между точками p1 и p2 рис. 3.10а.

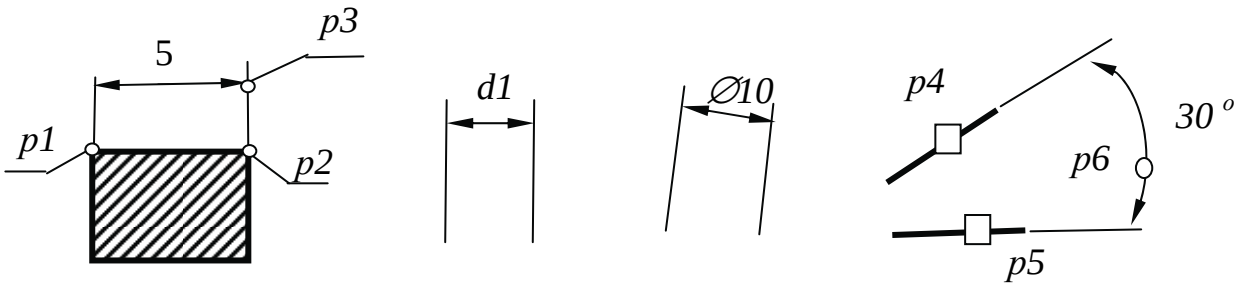


Рис. 3.10. Реализация команд нанесения размеров: а – горизонтальный размер, б – горизонтальный размер со значением строковой константы, в – параллельный размер со знаком диаметра

• Пример 9. (command "dim" "horiz" p1 p2 p3 "d1" "exit") – нанесение горизонтального размера. В качестве размерного текста выступает строковая константа "*d1*" рис. 3.10 б.

• Пример 10. (command "dim" "aligned " p1 p2 p3 (strcat "%%c" (rtos h 2 0)) "exit") – нанесение параллельного размера со знаком диаметра рис. 3.10 в.

Функция strcat предназначена для соединения двух строковых констант. Описание данной функции приведено в п.4.5; "%%c" – запись кода знака диаметра в файле шрифта. Функция rtos позволяет выполнять преобразование переменной h, имеющей тип данных действительного числа в значение строковой константы. Описание данной функции приведено в п. 4.5.

- Пример 11. (command "dim" "vertical" p1 p2 p3 "" "exit") – нанесение вертикального размера.
- Пример 12. (command "dim" "angular" p4 p5 p6 "30" "exit") – нанесение углового размера, точки p4, p5 указывают две прямые, между которыми проставляется угловой размер (данные прямые должны быть изображены заранее); p6 – указывает положение размерной линии; 30 – задание текста, рис. 3.10г.
- Пример 13. (command "qleader" p1 p2 p3 "" "R1" "" "") – нанесение размера радиуса *R1*. Точки p1, p2 и p3 задают положение полки размера (рис. 3.11);

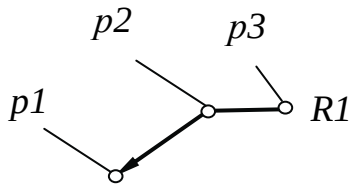


Рис. 3.11. Реализация команды qleader

- (**redraw** <имя примитива>) функция перерисовывает текущий видовой экран с примитивами, изображенными на нём. Если функция выполняется с аргументом, то будет перерисовываться только выбранный примитив. Определение и задание имен примитивов будет подробно рассмотрено в п. 4.2.

3.3. Пример составления программы построения параметрических изображений детали

Ниже приведен текст программы, позволяющий выполнять построение параметрических изображений детали, представленной на рис. 3.12.

Пусть необходимо построить параметрические изображения главного вида и разреза, представленных на рис. 3.12 при различных значениях

размеров *l, l1, h, h1, …* и т. д. Для составления теста программы, позволяющей решить указанную задачу, необходимо сделать следующее.

1. Обозначить узловые и вспомогательные точки p1, p2, p3 и т. п. на изображениях детали для формирования линий видимого контура, размеров, изображения штриховки и др.

2. Составить текст программы расчета координат указанных точек с использованием функций, представленных в пп. 2.3–2.5.

3. Составить тексты программ формирования среды черчения и получения изображений видов, размеров с использованием команд Автокада. Примеры создания среды черчения и реализации команд Автокада представлены в пп. 3.1, 3.2;

С целью облегчения работы с программой и анализа её функций (где начало и конец) рекомендуется скобки размещать, как показано стрелками на рис. 3.12. В тексте программы используют следующие функции:

(sl0) – встроенная функция, устанавливает слой – 0 (см. п. 3.1);

(sl1) – встроенная функция, устанавливает слой – 1 (см. п. 3.1);

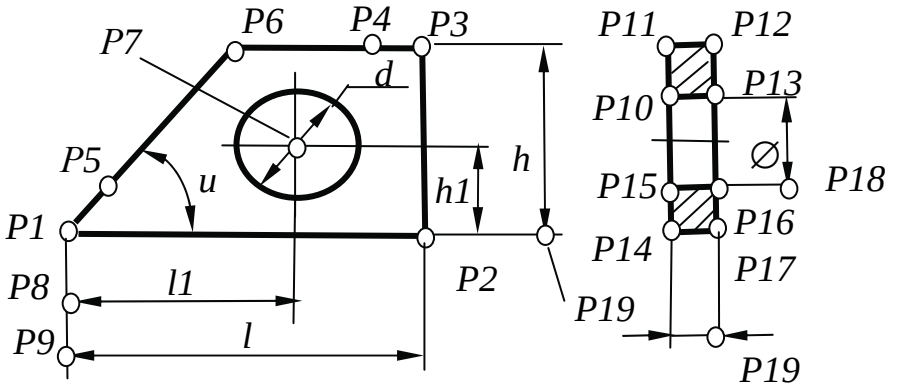
kon – имя функции для её загрузки.

Файлы текстовых программ (рис. 3.12), описанных на языке AutoLISP, имеют расширение .lsp. Для загрузки файла с текстом программы необходимо набрать на клавиатуре:

Command : (load "Ivanov") далее нажать клавишу Enter → где Ivanov имя файла,

Command: (kontur) далее нажать клавишу Enter → где kontur имя функции.

Загрузка файла "Ivanov" возможна с помощью интегрированной системы Visual LISP.




```
(defun kontur () ;функция построения параметрических изображений
                 ;видов детали
   (setq     ;расчет точек главного вида ******************************
         x   50   y  50  l  100  l1  50    h1  25   h  50  d  20 u 0.78
         p1  (list x y)                    p2  ( polar p1  0  l)
         p3  (polar p2  1.57  h)           p4 (polar p3 3.14  5)
         p5  (polar p1 u 5)                p6 (inters p1 p5 p3 p4 nil)
         p7  (list  (+ x  l1) (+ y h1))    p8 (polar p1 4.71 10)
         p9 (polar p1 4.71 20)
    ) ; окончание функции - setq
    ; формирование изображения главного вида**********************
   (command "line" p1 p2 p3 p6 p1 ""  "circle" p7 (/ d 2) )
    ; формирование изображения разреза
          (command "pline" p10 p11 p12 p13 p10 "")
          (command "pline" p14 p15 p16 p17 p14 "")
     ;нанесение изображения штриховки  на разрезе
   (command "hatch"  "jis_wood" "10" "0" p10  p14 "")
   (command "line" p15 p10  ""   "line" p16 p13  "" )
     ; нанесение размеров на чертеже
   (command "dim" "horiz" P1 P2 P9 ""   "horiz" P1 P7 P8 ""
                          "vertical"  P3 P2 P19  ""
                 " vertical " p13 p16 p18 (strcat "%%c" (rtos d 2 0)) "exit")
); окончание функции –   kontur
```

Рис. 3.12. Пример составления текста программы для получения параметрических изображений детали

3.4. Структура и запуск Visual LISP

Visual LISP *– это интегрированная среда разработки программ на языке программирования АВТОЛИСП в системе АВТОКАД. Она значительно облегчает процесс создания программы, её изменения, тестирования и отладки*. Visual LISP имеет собственный набор окон и меню, который отличается от соответствующего набора AutoCAD. Однако запуск интегрированной среды Visual LISP производится из системы AutoCAD. Поэтому для работы с Visual LISP система AutoCAD должна быть загружена.

Для запуска системы Visual LISP необходимо щелкнуть в падающем меню

Tools (средства) → AutoLISP → Редактор Visual LISP

После этого появится окно интегрированной среды Visual LISP for AutoCAD (рис. 3.13). Для создания файла необходимо выбрать пиктограмму  – New File (Новый файл). Если файл уже ранее был создан, необходимо выбрать пиктограмму  – Open File (Открыть файл). При составлении текста программы необходимо использовать пиктограммы копирования выделенного фрагмента в буфер и вставку содержимого буфера в текущий текст. Здесь предполагается, что студент уже знаком с основами информатики. Главное меню представляет собой систему, которая обеспечивает доступ ко всем средствам Visual LISP. Главное меню – это основной управляющий центр интегрированной среды Visual LISP. Роль дополнительных центров играют панели инструментов и отдельные кнопки, за которыми закреплены команды, используемые наиболее часто. Для облегчения работы со средой Visual LISP можно вывести на экран инструментальные панели. Вывести или отключить требуемые панели инструментов можно с помощью пункта Toolbars (Панели инструментов) падающего меню View (Просмотр), который вызывает диалоговое окно выбора панелей инструментов Toolbars. Диалоговое окно обеспечивает управление отображением на экране следующих панелей инструментов:

1 Standart (Стандартная);
2 Searh (Поиск);
3 Tools (Инструменты);
4 Debug (Отладка);
5 View (Просмотр).

Для того чтобы вывести ту или иную панель на экран, необходимо щелчком мыши включить/выключить соответствующие флажки, после чего нажать кнопку Apply (Применить).

При открытии файла с программой на языке AutoLISP автоматически вызывается окно текстового редактора. Следует помнить, что одновременно можно работать только с одним файлом в одном окне текстового редактора. Каждый раз при открытии файла, с которым мы работаем в интегрированной среде, Visual LISP отображает файл в новом окне текстового редактора. Текстовый редактор Visual LISP обеспечивает цветное изображение программы. Кроме того, он выполняет синтаксический анализ программы на языке AutoLISP и назначает соответствующие цвета различным константам, именам функций. Это помогает выявлять синтаксические ошибки до выполнения компиляции. Программы на языке AutoLISP содержат большое количество круглых скобок. Текстовый редактор помогает найти ближайшую закрывающуюся круглую скобку, которая соответствует указанной открывающей скобке.

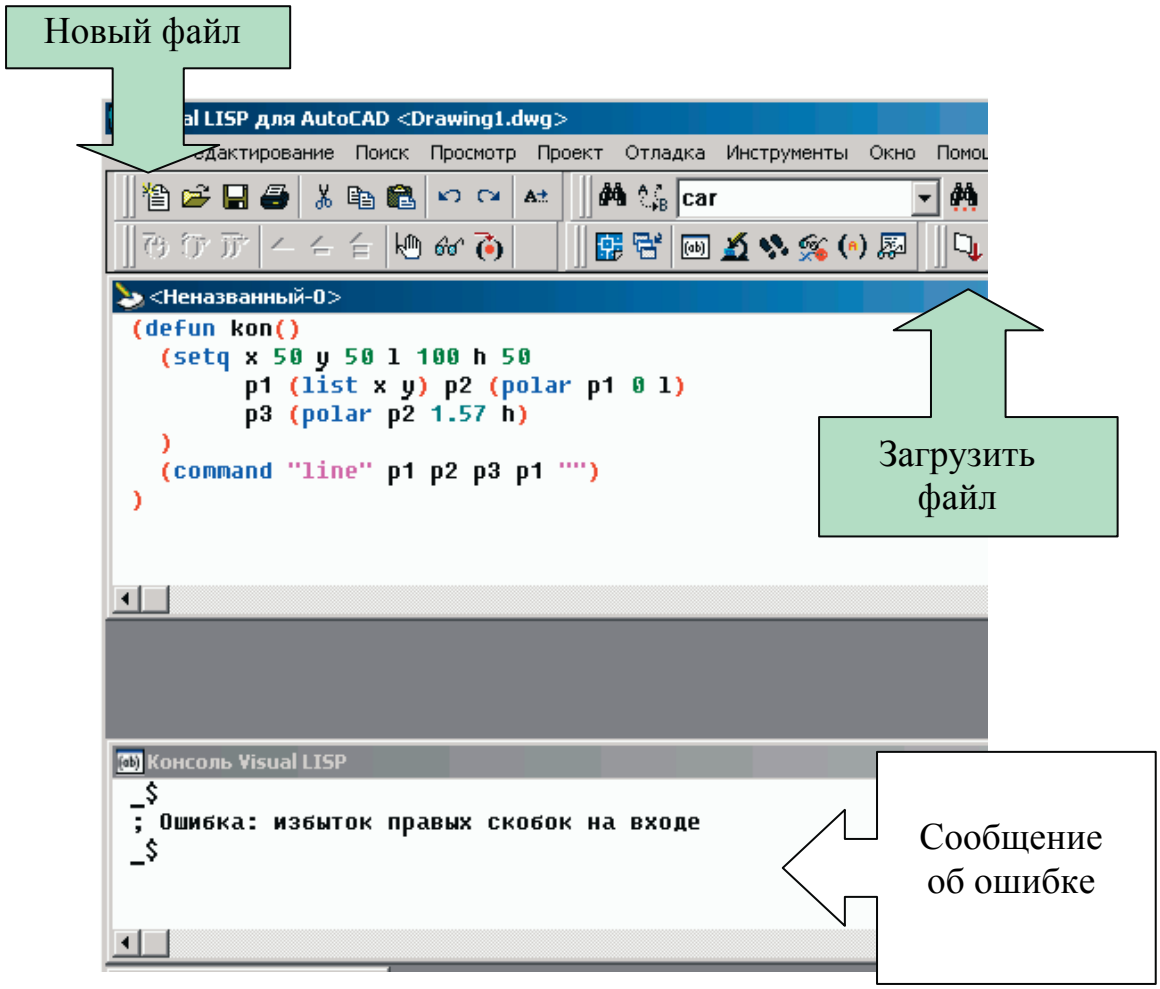


Рис. 3.13. Изображение окна Visual LISP

3.5. Возможные ошибки при запуске и отладке программ на языке Автолисп

При запуске программы на языке AutoLISP в окне консоли Visual LISP могут появляться следующие сообщения (рис. 3.13):

- Ошибка в правильной расстановке скобок. Надпись об этом будет содержать текст "Malformed list".
- Неизвестная функция.

Данная ошибка появляется, когда при наборе названия функции допущена орфографическая ошибка. Например (setg … Здесь допущена ошибка: вместо символа q набран символ g. Должно быть → setq).

- Ошибка в расстановке кавычек - Malformed string. Данная ошибка появляется, когда при наборе текста программы или существует лишний символ ", или данного символа не хватает. Например: (command "line" p1 p2 p3 ").

В этом примере есть ошибка: после переменной p3 находится одна двойная кавычка, а должно быть две двойные кавычки, т. е. (command "line" p1 p2 p3 "").

- Неверный тип аргумента.

error: bad argument type – неверный тип аргумента, передаваемого в функцию.

Данная ошибка появляется, когда функции неправильно передано значение аргумента. Пусть выполняется расчет точки (setq p22 (polar P21 U l)). При возникновении указанной ошибки на экране в командной строке появляется сообщение

(polar p21 u l)
;Ошибка неверный тип аргумента
(polar p21 u l) – указывается функция, куда неверно передано значение аргумента.

Для определения ошибки необходимо определить, какой из аргументов функции p21, u или l принимает значение nil или значение не соответствует функции. Значение переменной выясняют набором в командной строке имени переменной, перед которой указывают символ ! . Например: Command: !p21 – Enter. Если значение переменной p21 ранее рассчитано, то в скобках появляется значение координат точки p21 → (l00, 50). Если значение, например, переменной u не вычислено или ранее не задано, то появляется сообщение Command: !u → (nil).

Сообщение о данной ошибке появляется в следующих случаях:

– значение аргумента функции ранее не было рассчитано или введено;

– значение переменной при расчетах становится – nil (например, при использовании функции inters, когда отрезки являются параллельными);

– неправильное количество элементов списка в каком-либо аргументе функции, например в функции polar угол u имеет значение списка !u - (10,20);

– вместо действительного числа аргумент функции имеет значение строковой константы !u - " T1".

• Функции передано неверное количество аргументов.

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

error: Too few arguments – мало аргументов. Сообщение проявляется, когда мало аргументов указано для функции.

Схематика вывода информации об ошибках:

Запускаем команду:

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

В команде заведомо указан неверный тип данных, являющихся строковой константой. Второй аргумент должен быть типа REAL, а не строка "стол".

В результате получим ошибку:

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

Эта надпись обозначает следующее:

– сперва выводится тип ошибки – error: bad argument type;

– потом пишется выражение, в котором произошла ошибка – (RTOS DEFLT);

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

3.6. Вопросы и задачи для самопроверки

1. В чем заключается методика создания слоев в системе АВТОКАД с использованием текстовых программ на языке АВТОЛИСП?

2. Как задаются значения системных размерных переменных с помощью текстовых программ на языке АВТОЛИСП?

3. Каким образом осуществляется реализация команд системы АВТОКАДА, позволяющих получать изображения простых и сложных примитивов с использованием текстовых программ АВТОЛИСПА?

4. В чем заключается методика составления текстовой программы, предназначенной для получения параметрического изображения плоского контура?

5. В чем состоит назначение интегрированной системы Visual Lisp в АВТОКАД?

6. Какие возможные ошибки встречаются при запуске и отладке программ на языке Автолисп?

Задача 3.1

Составьте фрагмент текста программы, позволяющей создавать новый слой с именем 6, строящий линии желтого цвета (номер цвета 2) и имеющий тип линии center2.

Задача 3.2

Составьте фрагмент текста программы, позволяющей задавать значения системных размерных переменных: стрелочный размер – 10 мм; высота текста – 7 мм; продлить за линию (выносную) – 4 мм; отклонение от оригинала 1мм.

Задача 3.3

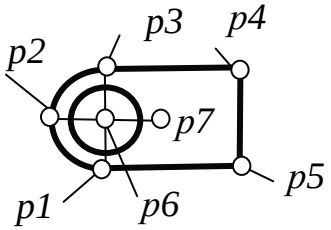


Рис. 3.14. Плоский контур задачи 3.3

Составьте фрагмент текста программы, позволяющей реализовать команды Автокада для построения изображения плоского контура, приведенного на рис. 3.14. Узловые точки имеют обозначения p1, p2, p3, p4, p5 и p6. Радиус окружности определяет переменная *r*. Расчет указанных на рисунке точек не проводить.

Задача 3.4

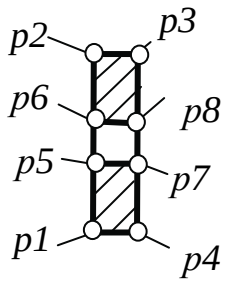


Рис. 3.15. Плоский контур задачи 3.4

Составьте текст программы, позволяющей реализовать команды Автокада с целью получения изображения разреза детали, представленного на рис. 3.15. Тип штриховки определяет файл "jis_wood", масштаб штриховки равен десяти, а угол наклона штриховки равен нулю. Выбор замкнутых контуров, подлежащих штриховке, провести без использования рамки. Расчет узловых точек не выполнять.

Задача 3.5

Составьте фрагмент текста программы, позволяющей реализовать команды Автокада с целью получения изображения втулки с простановкой размеров. Размер диаметр ∅20 задает переменная *d1*, ∅30 задает переменная *d2*. Узловые точки обозначены на рис. 3.16. Расчет узловых точек не выполнять.

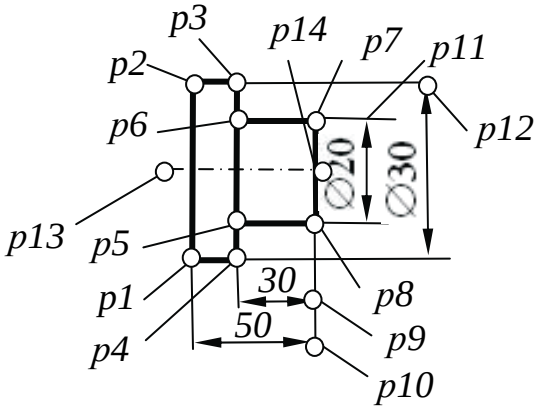


Рис. 3.16. Плоский контур задачи 3.5

4. ФУНКЦИИ, УПРАВЛЯЮЩИЕ СПИСКАМИ И ОБЕСПЕЧИВАЮЩИЕ ДОСТУП К ПРИМИТИВАМ. СТРОКОВЫЕ ФУНКЦИИ

4.1. Функции управления списками

Под управлением списками будем понимать их создание, изменение, их анализ и т. п. Управление осуществляется при помощи специальных функций. Наиболее часто применяют следующие функции:

- (**atom** <элемент списка>) возвращает nil, если элемент являете списком, в противном случае T;
- (**assoc** <элемент списка> <структурированный список>) позволяет устанавливать наличие <элемента списка> в <структурированном списке>. Если данный элемент присутствует, то функция возвращает подсписок с этим элементом. Если <элемента списка> нет, то функция возвращает nil.

Пример: пусть (setq spisok '((x 20) (y 30) (z 40))) – список, состоящий из переменных x, y, z и их значений. Тогда

(assoc 'y spisok) – возвращает (y 30). Эта функция, совместно с функцией subst (см. ниже), удобна для замены величин, которые отыскиваются в <структурированных списках> по заданному ключу с помощью <элемента списка>;

- (**subst** <новый элемент> <старый элемент> <список>) используют для просмотра заданного списка, с целью замены встречающихся <старых элементов> на <новые элементы>. Если <старых элементов> в <списке> нет, то функция исходный список не изменяет.

Пример: пусть задан список (setq spisok '(a (b 90) (c 40) (d a)). Требуется внести изменение в список spisok элемент d заменить новым элементом p. Тогда

(subst 'p 'd spisok) – возвращает новый список (a (h 90) (c 40) (p a)).

Для замены элемента b списка spisok элементом d соответственно необходима следующая запись функции:

(subst 'd 'b spisok) – возвращает новый список (a (d 90) (c 40) (d a));

• (**cond** (<тест1> <результат1> ...) (<тест2> <результат2> ...) ...). Эта функция в качестве аргументов использует списки, число которых может быть любым. Функция cond просматривает поочередно элементы списка, каждый из которых состоит из функции, задающей <тест>, и функции, задающей результат. Если в каком-либо из первых элементов списка функция <тест> окажется не nil, то функция cond вычисляет выражение <результат>, следующее за тестом, и возвращает значение этого выражения в этом подсписке. Например, если <тест1> → nil, <тест2> → nil, <тест3> → не nil , то функция cond возвращает <результат3>. Если не один из тестов не является истинным, то функция возвращает nil. Рекомендуется в качестве последнего теста использовать символ T(true). Тогда, если все предыдущие условия не выполняются, результатом функции будет выражение, соответствующее символу T.

Пример: пусть необходимо, какой-либо переменной ss задать значение 1, если пользователь на запрос системы указывает символ "Y" или "y". В случае, если пользователь указывает символ "N" или "n", переменная ss задает значение 0. Тогда, задавая запрос пользователя в символ s, функция cond проверяет с помощью тестов (= s "Y"), (= s "y"), (= s "N"), (= s "n") значения s и возвращает соответствующий результат переменной ss.

```
(setq ss (cond    ((= s   "Y") 1)
                  (( = s  "y")  1)
                  (( = s  "N") 0)
                  (( = s  "n")  0)
                  (t nil.)
         )

)
```

Если пользователь на запрос системы ответил "N", то результатом первых двух тестов (= s "Y") и (= s "y") двух элементов списков будет nil, поэтому элементы списков результат1 и результат2 не выполняются. Результатом теста3 в третьем элементе списка ((= s "N") 0) будет не nil, поэтому функция cond выполняет результат3 и возвращает переменной ss значение 0.

• Функция (**cons** <новый элемент> <список>) выбирает <новый элемент>, помещает его в начало списка и возвращает новый список с

дополненным элементом. Эта функция является основным конструктором списка.

Пример: пусть необходимо в список (а в) в начало списка поместить элемент с. Тогда (cons c '(а в)) – возвращает список (с а в). Здесь новым элементом является с.

- Функцию **(nth** < i > < список >) используют для извлечения из списка i-го элемента. Для первого элемента i = 0. В случае, если i > (n + 1), где n – число элементов в списке, функция возвращает nil.

Пример: (nth 2 '(7 16 23 4)) – Здесь i = 2, n = 0, 1, 2, 3. Поэтому возвращается третий элемент списка - 23.

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

Пример: пусть из списка (5 (c d) n) требуется извлечь последний элемент n. Тогда (last '(5 (c d) n)) – возвращает атом n.

- (**cdr** <список>) возвращает обновленный <список> без первого элемента. Пример: (cdr '(x y z)) - возвращает (y z), (cdr '()) – возвращает nil.

- (**append** <выражение1> <выражение2>...) Эта функция из заданных выражений, представляющих собой списки, формирует новый список путем слияния исходных. Пример: (append '(x y) (z)) – возвращает (x y z).

- (**length** <список>) определяет число элементов в <списке> и возвращает это число. Примеры: (length '(x y z)) – возвращает 3,
(length '()) – возвращает 0 (пустой список).

- (**mapcar** <функция> <список1> <список2> ...) рассматривает <список1>, <список2> и так далее как аргументы заданной <функции>. Она возвращает результат выполнения функции с указанными аргументами.

Пример: (mapcar 'sqrt (list 9 16 25)) – возвращает (3.0 4.0 5.0). Этот результат эквивалентен следующим: (sqrt 9) = 3.0, (sqrt I6) = 4.0, (sqrt 25) = 5.0.

- (**foreach** <имя> <список> <выражение> ...) присваивает каждому элементу <списка> заданное <имя> и выполняет <выражение> для каж-

дого элемента <списка>. Функция foreach возвращает результат последнего вычисленного выражения. Пример:
(foreach b (25 16 9) (sqrt b)) - возвращает 3.0.

• (**member** <выражение> <список>) возвращает часть списка, начинающегося с первого имеющегося в нём <выражения>. Если указанного <выражения> в заданном <списке> нет, то функция возвращает nil.

Примеры:
(member 'y '(x y z)) – в данном примере в качестве выражения выступает переменная y, поэтому возвращается часть списка (y z);
(member 'x '(y z z1) – возвращает nil, так как выражение или переменная x в списке (y z z1) отсутствует.

• (**reverse** <список>) Результатом выполнения данной функции над заданным <списком> есть список с элементами, расставленными в обратном порядке. Пример: (reverse '(x y z)) – возвращает (z y x).

4.2. Обеспечение доступа к примитивам. Создание наборов примитивов

Примитивы подразделяют на простые и сложные. К простым примитивам относят точку, прямую, окружность, дугу, эллипс, многоугольник и т. д. К сложным примитивам относят Плинию, штриховку, текст, размеры и др. Для работы с примитивами применяют следующие функции. Рассмотрим некоторые из них.

• (**ssget** [<режим>] [<точка1>] [<точка>]) создает набор только основных примитивов, в который не могут быть включены атрибуты и вершины Плиний. Аргумент <режим> задает способ выбора примитива. Этот выбор может выполняться с помощью "рамки", "секущей рамки", соответствовать "последнему" или "текущему" набору примитивов Автокада. С помощью аргументов <точка1> и <точка 2> осуществляется соответствующий выбор. Если в функции ssget не указан ни один аргумент, то Автокад дает сообщение "Выберите объекты". Выбор объектов функция осуществляет в интерактивном режиме. В последнем случае выбранные объекты подсвечиваются. Примеры применения функции ssget:
(ssget) – выбор примитивов выполняется в интерактивном режиме.

Функция (ssget "W" '(10 10) '(20 20)) позволяет выбирать все примитивы, находящиеся в рамке от (10,10) до (20,20);

(ssget "T") – выбирается текущий набор примитивов. Например, если в последний набор примитивов были включены две прямые и окружность, функция (ssget "T") возвращает набор из данных примитивов.

(ssget "П") – выбирается примитив, созданный в рисунке последним;

(ssget "X" <фильтр-список>) – выбираются примитивы в соответствии с <фильтр-списком>. В последнем случае выбор примитивов и создание из них набора осуществляются по определенным критериям. К данным критериям относятся:

1 – принадлежность примитивов определенному слою;

2 – тип примитива;

3 – имя типа линии;

4 – имя блока;

5 – код цвета и некоторые другие.

<Фильтр список> представляет собой список, который состоит из группового кода примитивов и параметров, характеризующих эти примитивы. Фильтр-список указывает, по какой характеристике выбираются примитивы и по каким значениям ведется поиск. Например, запись функции (ssget "X" ((0. "Circle"))) возвращает набор из всех окружностей, имеющихся в рисунке. 0 – задает групповой код примитива. Пусть необходимо создать набор из примитивов, построенных студентом на горизонтальной проекции при решении тестовой задачи, связанной с построением недостающей проекции точки по заданной её фронтальной проекции, если данная точка принадлежит поверхности конуса вращения (рис. 4.1). Тогда, запись функций (setq LL (ssget “W” pr1 pr2)) создает набор примитивов LL, которые построены на горизонтальной проекции. Точки pr1 и pr2 определяют нижний левый и верхний правый угол рамки. Набор LL в этом примере состоит из одной окружности и одного текста.

- (**sslength** <набор>) определяет количество примитивов в <наборе> и возвращает его величину. Если число примитивов в наборе меньше 32767, то оно целое, в противном случае – действительное.

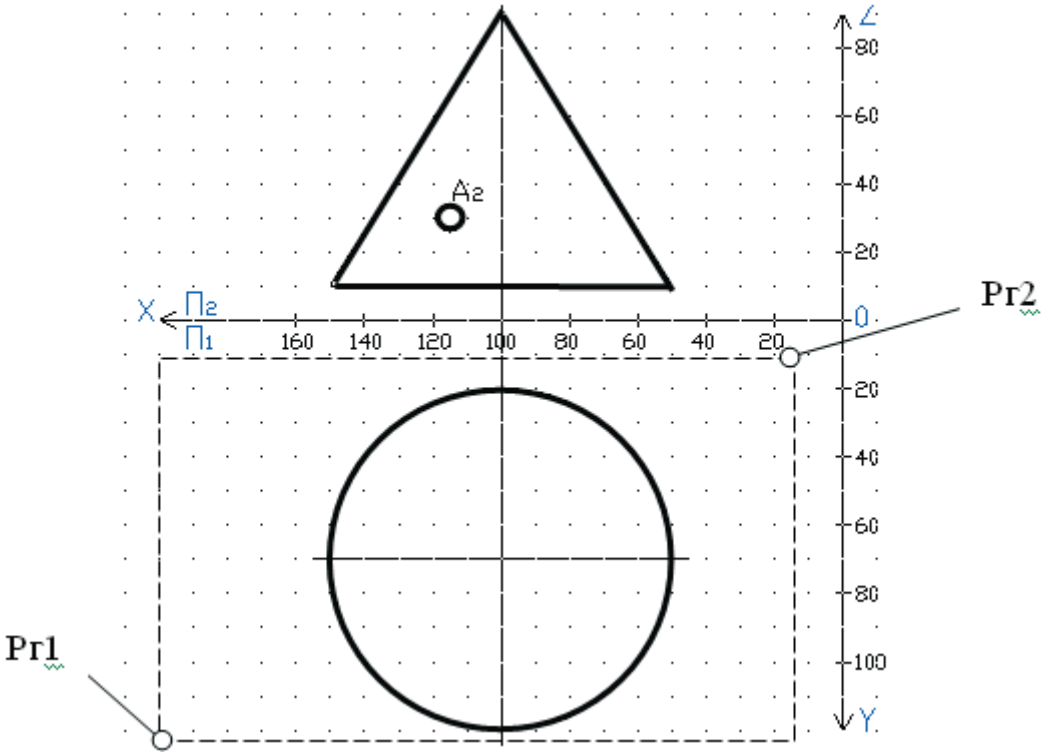


Рис. 4.1. Создание набора примитивов с использованием рамки

Пример: пусть создан набор duga, состоящий из всех дуг окружностей, имеющихся в рисунке, а их число равно 5. Применяя функцию (sslength duga), мы получим результат 5.

• (**ssname** <набор> <индекс>) устанавливает в <наборе> по заданному <индексу> имя примитива и возвращает его. Причем, если устанавливается имя примитива с номером, большим 32767, то <индекс> должен записываться как действительное число. Первому примитиву соответствует индекс 0.

Пример: пусть создан набор nabor, состоящий из всех примитивов, имеющихся на рис. 4.1 и расположенных в некоторой рамке:
(setq nabor (ssget "P" pr1 pr2)), где pr1 и pr2 – координаты нижнего левого и верхнего правого углов рамки соответственно. Тогда
(setq nb1 (ssname nabor 0)) – возвращает имя первого примитива, являющегося окружностью, например, в виде (60000048);
(setq nb2 (ssname nabor 1) – возвращает имя второго примитива, являющегося текстом, например, в виде (60000030).

• (**ssadd** [<имя примитива>] [<набор>]) дополняет заданный <набор> новым примитивом с указанным именем и возвращает измененный набор. Если эта функция записана без аргументов, то создается нулевой набор. При вызове функции с одним аргументом <имя примитива> создается новый набор с одним примитивом. Примеры:

(setq nabor1 (ssadd)) – создается нулевой набор,

(setq prim1 (ssget "П")) – из рисунка выделяется последний примитив и его имя устанавливается в переменную prim1.

(setq nabor2 (ssadd prim1 nabor1)) – возвращает набор nabor2, состоящий из примитива с именем prim1.

• (**ssdel** <имя примитива> <набор>) удаляет из <набора> <имя примитива> и возвращает имя с набором. Если в этом наборе нет задаваемого примитива, то функция возвращает nil. Примеры: пусть в наборе naborl существует примитив с именем prim1. Тогда:

(ssdel prim1 naborl) – возвращает набор naborl без примитива prim1,

(ssdel prim2 naborl) – возвращает nil.

• (**ssmemb** <имя примитива» <набор>) С помощью данной функции осуществляется проверка на принадлежность примитива с заданным именем указанному <набору>. Если этот примитив содержится в наборе, то функция возвращает его имя, в противном случае – nil.

Примеры: пусть в наборе naborl существует примитив с именем prim1. Тогда (ssmemb prim1 naborl) – возвращает имя примитива priml, (ssmemb prim2 naborl) – возвращает nil.

• (**entnext** [<имя примитива>]) возвращает имя первого, имевшегося в рисунке примитива, следующего в базе данных за примитивом с указанным именем. Если функция вызывается без аргумента, то будет возвращено имя первого не удаленного в базе данных примитива.

• (**entlast**) возвращает из базы данных имя последнего примитива.

Пример: (setq prim (entlast)) – присваивает переменной prim имя последнего примитива, добавленного в рисунок, и возвращает его имя.

4.3. Функции для работы с данными примитивов

Приведенные ниже функции, совместно с рассмотренными ранее, позволяют извлекать и изменять данные о примитивах.

• (**entdel** <примитив>). Эта функция позволяет в течение одного сеанса редактировать, удалять и восстанавливать <примитив> в рисунке. Окончательно <примитив> удаляется из рисунка только после выхода из режима редактирования. Примеры:

(setq prim1 (entlast)) – устанавливает в prim1 имя последнего примитива, добавленного в рисунок,

(entdel prim1) – удаляет примитив priml,

(entdel prim1) – восстанавливает удаленный примитив prim1.

• (**entget** <примитив>) позволяет выбирать <примитив> из базы данных. Функция возвращает список, который состоит из параметров, определяющих этот примитив. Такой список состоит из двух частей. Первая часть включает в себя код DXF (флаг), а вторая – данные. Конкретные данные о примитиве могут быть извлечены из этого списка с помощью функции assoc. При необходимости эти данные могут быть изменены, а следовательно, измены свойства примитива.

Пример: пусть последним в рисунок добавлен отрезок, проходящий через точки A(10 10) и B(20 20). Для получения данных об этом отрезке запишем выражение, состоящее из следующих функций:

(setq otr (entget (entlast))).

В результате переменной otr будет соответствовать список следующего вида:

```
( ( -1 . <Имя примитива: 60000018>);
  ( 0 . "LINE")   – указывает – тип примитива  "отрезок";
  ( 8 . "0" )   –   примитив изображен на слое "0";
  (10 10.0 10.0 0.0) – указывает координаты начальной точки отрезка;
  (11 20.0 20.0 0.0) – указывает координаты конечной точки отрезка;
)
```

Числа 0. 8, 10, 11 в начале подсписков являются флагами соответствующих групп данных о примитиве.

- (**entmod** <список>) используют для изменения информации в базе данных о примитиве, указанном в <списке> после флага – 1. Она также преобразовывает <список> в формат, который в дальнейшем возвращается функцией entget. Функции entget, assoc, subst и entmod позволяют получить информацию в виде списка о базе данных примитивов, извлечь необходимые данные, при необходимости изменить некоторые из них (за исключением типа примитива) и в заключение обновить параметры примитива в базе данных.

Пример: пусть последней в рисунке изображена окружность с центром 0 (l00 100), радиусом R = 40 ед. и находится она на слое "1". Тогда
(setq priml (entlast)) – устанавливает в priml имя окружности, добавленной в рисунок,
(setq dprim1 (entget priml)) – устанавливает в dprim1 список, соответствующий базе данных примитива priml и имеющий вид:
((-1 . <Имя примитива: 60000030>),
(0 . "CIRCLE") – указывает тип примитива – "окружность",
(8 . "1") – примитив изображен на слое "1",
(10 100.0 100.0) – указывает координаты центра окружности,
(40 40.0) – указывает величину радиуса окружности.

Пусть теперь требуется изменить координаты центра окружности. Тогда, используя нижепредставленные функции, будет внесено изменение координат центра окружности и будет обновлено соответственно изображение окружности (с новым центром):

```
(setq dprim1 (subst  (cons 10 '(50.0 50.0)) (assoc 10 dprim) dprim1)
 )
(entmod dprim).
```

4.4. Пример составления программы построения изображения детали с использованием наборов примитивов

Пусть необходимо изобразить разрез детали, в котором может присутствовать отверстие заданного диаметра d или нет (рис. 4.2). Реализация команды построения изображения штриховок рассмотрена в п. 3.2.

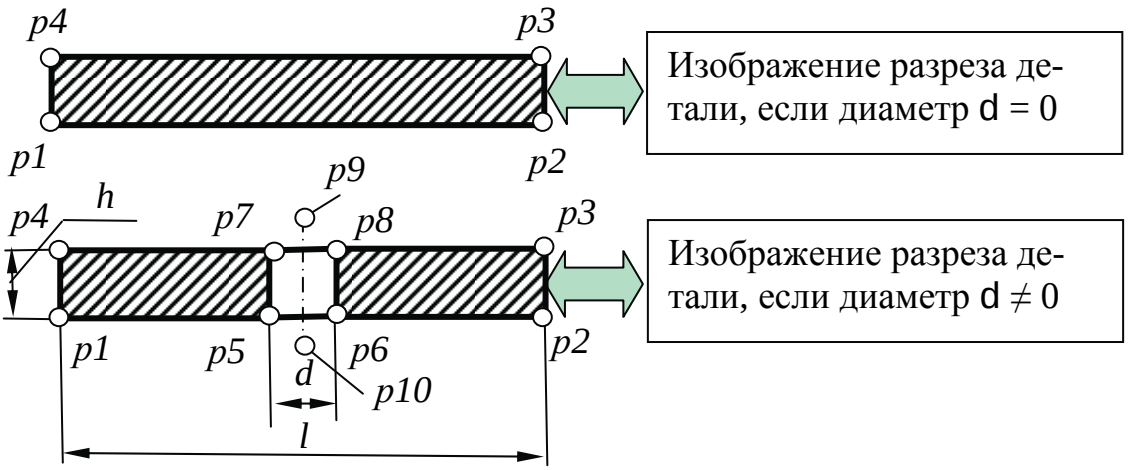


Рис. 4.2. Получение изображения штриховки с использованием функций доступа к примитивам

Приведенный ниже пример текста программы позволяет в зависимости от значения размера диаметра отверстия строить изображение разреза как с отверстием, так и без него. Для этого использованы логические функции. Программа построения изображения детали будет состоять из двух частей. Первая часть будет соответствовать, когда параметр d будет равен нулю, а вторая часть – когда указанный параметр соответственно не равен нулю. Текст программы будет иметь следующий вид:

```
(defun kontur1()
  ;блок ввода значений геометрических параметров
  ;изображения разреза
  (setq h (getreal "Задайте длину–d: ") h (getreal "Задайте высоту–
  h: ")
              p1 (getpoint "введите базовую точку: ")
              d   (getreal "Задайте диаметр отверстия – d: ")
```

```
                P2 … P4;блок программы расчета точек P2 – P4
      )        ; Если диаметр отверстия  d ≠ 0
      (if (= d  0) ()
                  (setq p5…p10) ;блок программы расчета точек
      )
 ; Построение разреза
       (sl0)(command "точка" "0,0") (setq ew entlast))
        (if (= d 0)
            (prong ;построение изображения разреза без отверстия
              (sl0) (command "line" p1 p2 p3 p4 p1  "")(sl1) (strix 45 2)
            )     ; Если диаметр отверстия  d ≠ 0
            (prong ;построение изображения разреза с отверстием
              (sl0) (command "line" p1 p4 p5 p7 p1  "")
                    (command "line" p6 p8 p3 p2 p6  "") (sl1) (stiix 45 2)
              (sl0) (command "line" p7 p8 "" "line" p5 p6  "")
                   (sl3) (command "line" p9 p10 "" )
           )
   ) ; окончание функции – defun
```

Функции и команды, используемые в программе kontur1:

- (command "точка" "0,0") – примитив, необходимый для задания примитива, от которого начинается отсчет примитивов, помещаемых в набор. Этот набор определяет замкнутый контур, подлежащий штриховке. С помощью функций (setq ew entlast)) данному примитиву присваивается имя ew.
- (entlast) – функция присваивает код примитива точки переменной ew, которая используется функцией strix.
- (strix 45 2) – подпрограмма, строящая изображение штриховки замкнутого контура. Данная программа определяет набор из примитивов, который определяет замкнутый контур и заштриховывает его.

  Текст этой программы приведен ниже.
- (sl0) (sl1) (sl3) – функции для установки слоев (см. п. 3.1).

Пример программы с использованием набора примитивов при построении изображения штриховки:

```
(defun strix($u $r); программа построения штриховки замкнутого контура
                    ;определяемого набором примитивов
;$u – параметр определяющий угол наклона штриховки
;$r – параметр определяющий расстояние между штрихами
    (command "штрих" "c" $u $r  ""
       (setq ew1 (entlast)   ss (ssadd) ss
         (while (setq ew (entnext ew)) (setq ss (ssadd ew ss)) )
       ) ""
    ) ; окончание функции - command
) ; окончание функции – defun
```

В программе strix используют следующие переменные и функции для работы с примитивами.

- ew – параметр, определяющий начало отсчета примитивов для определения набора, который определяет замкнутый контур, подлежащий штриховке.
- (ssadd) – функция создает нулевой набор (в данном примере ss).
- (ssadd ew ss) – функция дополняет набор (ss) новым примитивом.
- (entnext ew) – функция возвращает имя примитива, следующего в базе данных за примитивом с указанным именем (ew).
- (while <выражение1> <выражением2>) – функция организует цикл над <выражением2> до тех пор, пока <выражение1> (setq ew (entnext ew)) не примет значение nill.

4.5. Строковые функции

Строковые константы и функции используют для автоматизированного формирования текстовых документов, а также текстов технологических обозначений чертежей деталей и сборочных чертежей. В качестве строковых констант выступают ключевые слова, числа и др. С помощью строковых функций возможно формирование слов, обозначений, предложений и др. Ниже приведены строковые функции:

• Функция (**itoa** <целое число>) возвращает результат преобразований целого числа в строковую константу. Пример:
(itoa 55) возвращает "55".

• Функция (**rtos** <число> [<режим>] [<точность>]) осуществляет преобразование <числа>, являющегося действительной величиной, в строковую константу и возвращает ее. Преобразование выполняется с учетом значений <режима> и <точности>, принимавших целые значения. Так, значению <режима>, равному 1, соответствует десятичный формат представления чисел: равному 2 – десятичный и т. д. Примеры: Пусть переменная aa несет значение числа 123.45. Тогда:
(rtos aa 1 5) – возвращает "1.23450E+02".
В данном примере режим 1 соответствует заданию числа в виде заданной степени. Число 5 определяет число знаков после запятой.
(rtos aa 2 2) – возвращает "123.45".

В данном примере режим 2 позволяет представление числа в виде десятичной дроби. Второе число 2 определяет число знаков после запятой.
(rtos aa 2 0) – возвращает "123".

• Функция (**atof** <строковая константа>) преобразовывает <строковую константу> в действительное число и возвращает его значение.

Пример:
(atof "55.65") – возвращает 55.65.

• Функция (**strcat** <строка1> <строка2>) выполняет соединение строк <строка1> <строка2> и других в одну и возвращает ее. Примеры: Пусть значение диаметра отверстия определяет переменная d, которая равна 20,5. Тогда
(strcat "%%c" (rtos d 2 0)) – возвращает ∅20. В данном примере обозначение "%%c" задает код знака диаметра в файле шрифта [1].
(strcat "R" "56.5") – возвращает "R56.5",
(strcat "56.5" "+ 0.001") – возвращает "56.5 + 0.001".

• Функция (**strlen** <строка>) возвращает целое число, представляющее собой длину в символах заданной <строки>.

Примеры:
(strlen "R50.5") – возвращает 5, т. к. число символов в данной записи равно пяти, включая символ точку.

(strlen "56.5+0.001") – возвращает 10, т. к. число символов в данной записи равно десяти, включая символы точек и знак плюс.

• Функция (**substr** <строка> <начало> [<длина>]) выделяет из заданной <строки> подстроку, начинающуюся с <начального> символа и имеющую указанную <длину>. В случае, если аргумент <строка> не указан, то функция возвращает подстроку до конца строки.

Примеры:

(substr ”абвг” 2*)* – возвращает "бвг". В данном примере число 2 определяет начальный символ б подстроки. Так как факультативный аргумент <длина> отсутствует, то подстрока возвращается до конца.

(substr "абвг" 1 2) – возвращает "аб". В данном примере длина подстроки равна двум.

• (**strcase** <строка> [<признак>]) возвращает копию заданной <строки>, переведя все ее символы в верхний или нижний регистр, в зависимости от аргумента <признак>. В случае, если <признак> nil или опущен, то все символы в <строке> будут переведены в нижний регистр.

Примеры:

(strcase "Поверхность") – возвращает "ПОВЕРХНОСТЬ",

(strcase "Поверхность" T) – возвращает "поверхность".

4.6. Пример составления программы с использованием строковых функций, позволяющих формировать тексты технологических обозначений

При составлении программы формирования чертежей деталей и сборочных единиц часто приходится формировать тексты различных технологических обозначений или требований. Содержание таких текстов зависит от вводимых технологических параметров, используемых при проектировании. В качестве указанных параметров могут выступать обозначение материалов, обозначение неразъемных соединений и др. Пусть необходимо сформировать обозначение сварного шва на сборочном чертеже. На рис. 4.3а показана последовательность этого обозначения [6], а на рис. 4.3б приведен пример обозначения сварного шва. На рис. 4.3в приведено обозначение точек, позволяющих сформировать изображение полки с

надписями. На рис. 4.3г приведена информация, используемая при формировании обозначения сварного шва.

Ниже приведен текст программы, позволяющей осуществлять интерактивный ввод данных, характеризующих сварной шов.

```
(defun svarka(); программа задания технологических параметров сварки
      (setq   a (getstring "введите вид сварного соединения:")
              b (getstring "введите буквенно-цифровое обозначение шва:")
              c (getstring "введите размеры прерывистого вида:")
              d (getstring "введите  способ сварки:"))
); окончание функции – svarka
```

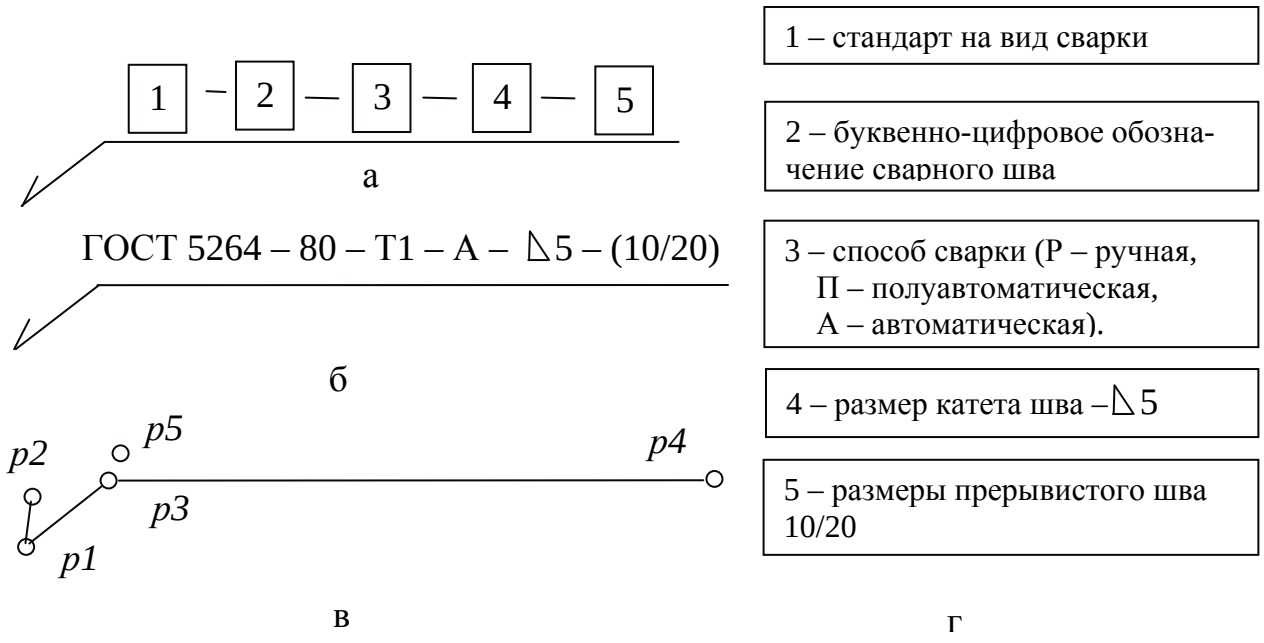


Рис. 4.3. Получение изображения обозначения сварки
с использованием строковых функций

После каждого запроса необходимо с клавиатуры ввести значения строковых констант и после этого нажать клавишу ENTER.

При формировании обозначения сварки размер катета шва может не обозначаться, если разделка кромок не предполагается. Если разделка кромок предусмотрена, то размер катета шва обозначается. Это зависит от того, имеется ли скос кромок у сварных деталей или нет [6]. Блок программы для расчета и построения изображений обозначений сварки на чертеже в зависимости от различных технологических параметров сварки (см. рис. 4.3) представлен ниже.

```
(defun svarka ()
 ; Расчет размера катета шва сварного соединения в зависимости от
 ;толщины свариваемых деталей
 (if (and (= b "T1") (= b "T3")) (setq K 0))  ; в этой строке параметр b
        ;определяет  буквенно-цифровое обозначение сварного шва.
 (if (= K 0) (setq e1 (/ S 3); в этой строке параметр S определяет
                        ;наименьшую толщину свариваемой пластины.
                  e (rtos e1 2 0)
            )
  )
  ;определение строковой константы обозначения сварки на чертеже
  ; с помощью переменных, значения которых введены с помощью
  ;функции svarka
  ; соединение строковых констант, вводимых в интерактивном ре-
жиме.
  (if (= k 0) (setq tt  (strcat  a "-"  b  "-"   c  "-" d))
              (setq tt (strcat  a "-"  b  "-"  "◺" "-" e "-" c "-" d ..))
  )
  ; расчет и формирование изображения выноски (изображение
  ;односторонней стрелки и текстовой полки)
  (setq lt (* 5  (strlen tt))
      p1 (getpoint "Укажите точку обозначения сварки")
      p2 (polar  p1 1.2 5)      p3 (polar p1 0.78 20)  p4 (polar p3 0 lt)
      p5 (polar p3 1.57 2)
  )
 ;формирование  изображения выноски
 (command "line" p2 p1 p3   p4 "")
 ;формирование  изображения текста обозначения сварки
 (command "text" p5  "5" "0" tt )
); окончание функции – defun
```

4.7. Вопросы и задачи для самопроверки

1. Назовите функции, позволяющие управлять списками.

2. Назовите функции, обеспечивающие доступ к примитивам. С какой целью используются данные функции?

3. Назовите функции, позволяющие извлекать информацию о примитивах.

4. Каков порядок составления программы с использованием наборов примитивов при получении фрагментов параметрических изображений разрезов деталей?

5. Назовите строковые функции и примеры их использования при получении текстовых технологических обозначений чертежей.

Задача 4.1

Задан список l1 - (10 20 30 40). Составьте фрагмент текста программы, позволяющий извлекать квадратный корень из четырех элементов списка l1 и возвращающий результат извлечения данного корня.

Задача 4.2

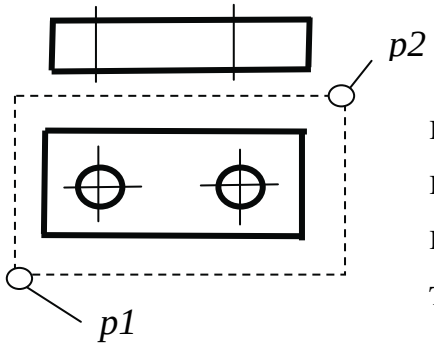


Рис. 4.4. Изображение детали к задаче 4.2

Составьте фрагмент программы, позволяющий создать набор LL1 из примитивов, находящихся в рамке, изображенной штриховой линией на рис. 4.4. Положение рамки определяется точками p1 и p2.

Задача 4.3

Составьте фрагмент текста программы, позволяющий создавать список spisok, задающий координаты центра окружности, которая была построена последней в текущем чертеже.

Задача 4.4

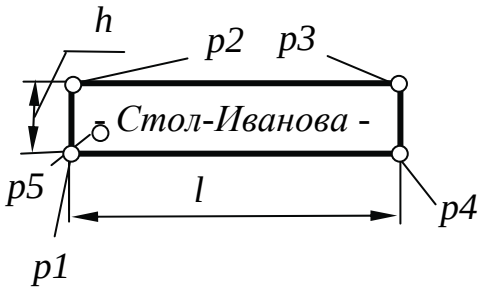


Рис. 4.5. Изображение детали к задаче 4.4

Составьте фрагмент текста программы, позволяющий строить изображение прямоугольника, длина *l* которого зависит от текста, располагающегося внутри его (рис. 4.5). Высота текста равна 5мм. Текст определяется соединением двух переменных t11 и t12. Первая переменная t11 имеет значение строковой константы “Стол”, вторая переменная t12 имеет значение строковой константы, задающей фамилию, которая может иметь произвольное количество символов. Точка p1 имеет координаты x = 80, y = 100. Точка p5 определяет базовую точку текста.

Задача 4.5

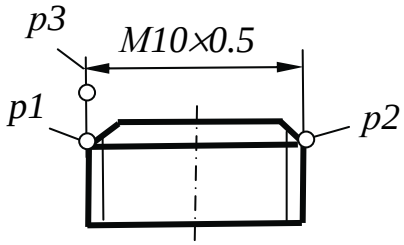


Рис. 4.6. Изображение детали к задаче 4.5

Составьте фрагмент текста программы, позволяющий строить изображения размера резьбы с надписью, указанной на рис. 4.6. Номинальный диаметр резьбы задает переменная dr, шаг резьбы – мелкий и его значение задает переменная ss.

5. ФУНКЦИИ ДЛЯ ОРГАНИЗАЦИИ ГРАФИЧЕСКОГО ДИАЛОГА ПОЛЬЗОВАТЕЛЯ С ПРОЕКТИРУЕМЫМ ИЗДЕЛИЕМ

5.1. Функции для ввода данных в интерактивном режиме

Для ввода данных в интерактивном режиме в Автолиспе предусмотрен ряд функций, у которых все аргументы являются факультативными. Однако часто в этих функциях содержится строковая константа, содержащая текст запроса или подсказку пользователю о вводе необходимых данных. Все эти функции прерывают выполнение программы до тех пор, пока не будут введены данные с клавиатуры или с помощью устройства указания. В ответ на запрос функций ввода нельзя задавать выражения Автолиспа, так как это приведет к ошибке.

Для задания ограничений на ввод соответствующих параметров используют функцию initget .

- (**initget** [<биты>] [<строка>]) В данной функции <биты> являются факультативным аргументом – целым числом. В зависимости от его величины устанавливаются соответствующие ограничения на ввод параметров. Так, если <бит> равен 1, то запрещен пустой ввод, при равенстве <бита> двум – запрещен ввод нуля, а при равенстве четырем – запрещен ввод отрицательных чисел. Если пользователь при вводе данных не выполняет хотя бы одно из таких условий, Автокад указывает на ошибку и предлагает повторить ввод данных. Другой факультативный аргумент рассматриваемой функции – <строка> представляет собой список ключевых слов, которые проверяются в процессе использования функций ввода.

- (**getangle** [<точка>] [<подсказка>]) создает паузу при выполнении программы для ввода угла. Она возвращает угол в радианах между задаваемым пользователем вектором и положительным направлением координатной оси x в пользовательской системе координат. Начальная точка вектора может быть определена первым аргументом функции, а вторая – задаваться устройством указания. В случае, если факультативный параметр <точка> опущен, то пользователь должен ввести две точки. Примеры:

(setq ugol (getangle)) – создается пауза для задания двух точек,

(setq ugol (getangle "Задайте вектор:")) – создается пауза и высвечивается факультативная подсказка.

• (**getpoint** [<точка>] [<подсказка>]). Эта функция используется для ввода пользователем точки. Аргументы <точка> и <подсказка> являются факультативными. Ввод точки возможен как устройством указания, так и при задании ее координат. Если в записи присутствует аргумент <точка>, то Автокад строит "резиновую" нить от этой точки до положения, в котором находится курсор. Примеры:

(setq p (getpoint)) – создается пауза для задания точки;

(setq p (getpoint "Задайте точку:")) – создается пауза и высвечивается факультативная подсказка Задайте точку:.

• (**getcorner** <точка> [<подсказка>]) возвращает координаты указываемой точки. Отличие от функции getpoint состоит в том, что getcorner строит "резиновую" рамку на экране при передвижении курсора. Аргумент <точка> в данной функции обязателен.

• (**getdist** [<точка>][<подсказка>]) используют для ввода расстояния. Функция возвращает величину расстояния как действительное число. Аргументы <точка> и <подсказка> являются факультативными. Расстояние может быть задано как с клавиатуры, так и вводом двух точек на экране устройством указания. В последнем случае Автокад рисует "резиновую" нить от первой точки до положения, в котором в данный момент находится курсор. С помощью этой функции можно определять расстояния между двумерными и трехмерными точками. В последнем случае с помощью функции ihitget должен быть установлен соответствующий флаг – "трехмерные точки". Примеры:

(setq rast (getdist)) – создается пауза для задания расстояния,

(initget 17) – устанавливается флаг ввода трехмерных точек,

(setq rast (getdist '(10 10 10) "Укажите 2-ю точку:)) – создается пауза и высвечивается факультативная подсказка.

• (**getint** [<подсказка>]) создает паузу для ввода пользователем целого числа и возвращает его. Аргумент <подсказка> является факультативным. Пример:

(setq snak (getint "Задайте целое число:)) – создается пауза и высвечивается факультативная подсказка.

• (**getreal** [<подсказка>]) Данная функция аналогична предыдущей, но служит для ввода действительных чисел. Пример:
(setq snak (getreal "Задайте действительное число:")) – создается пауза и высвечивается факультативная подсказка.

• (**getstring** [<флаг - пробела>] [<подсказка>]) создает паузу для ввода строковой константы и возвращает её. Если присутствует факультативный аргумент <флаг – пробела> и он не равен nil, то вводимая <подсказка>, являющаяся факультативной, может содержать пробелы. В противном случае пробел в строковой константе прервет её ввод. Примеры:
(setq obosn (getstrig)) – создается пауза для ввода строковой константы,
(setq obosn (getstring T "Введите обозначение сечения:")) –создается пауза и высвечивается факультативная подсказка.

• (**getkword** [<подсказка>]) С помощью этой функции осуществляется ввод ключевого слова, запрашиваемого у пользователя. Ключевые слова, представляющие собой строковые константы, задаются с помощью функции initget. Функция initget возвращает строковую константу, соответствующую ключевому слову. Пример:
(initget w "Да" "Нет")
(setq w (getkword "Будете строить вид слева ? (Да или Нет)")) Функция getkword установит в символ W строковую константу "Да" или "Нет" в зависимости от ответа. В случае, если ответ не соответствует ни одному ключевому слову или будет задан пустой ввод, Автокад повторит запрос на ввод ключевого слова.

5.2. Задание геометрических параметров проектируемых изделий с использованием изображений слайдов

С целью установления смысла значений геометрических параметров проектируемых изделий, при вводе могут быть использованы изображения слайдов. При этом на графической зоне может появляться изображение детали с обозначением вводимых размеров и переменных (рис. 5.1). После этого пользователь может оценить изображение слайда и ввести нужные значения параметров в интерактивном режиме. ***Слайд*** *является мгновенным снимком чертежа*. Слайд от чертежа отличают по следующим признакам:

• информация о слайде записывается в файл с расширением .sld;

• слайд на порядок занимает большой объем размера памяти, чем файл чертежа с расширением .dwg;

• изображение слайда в отличие от чертежа мгновенно появляется на графической зоне экрана монитора;

• слайд невозможно редактировать. Поэтому копии слайдов необходимо хранить в файлах с расширением .dwg.

Для создания слайда вначале формируют его изображение, а затем используют команду MSLIDE → ДСЛАЙД, при этом система запрашивает имя слайда. Далее указывают директорию и имя файла слайда. Для удаления изображения слайда с графической зоны экрана монитора используют команду redraw «освежи».

Для вывода изображения слайда предусмотрена команда VSLIDE → СЛАЙД. После ввода команды необходимо указать директорию и имя файла с расширением .sld. Для вывода изображения слайда текстовой программой необходимо сформировать следующую запись:

(command "vslide" "kon") – эта команда обеспечивает вызов текстовой программой изображение слайда, хранящегося в файле с именем kon и расширением .sld.

(command "redraw") – удаление текстовой программой изображения слайда.

Ниже приведен фрагмент текста программы для ввода значений геометрических параметров в интерактивном режиме с использованием изображения слайда, представленного на рис. 5.1:

```
(defun sb()
  ; ввод геометрических  параметров проектируемого  изделия
  (command "vslide" "kon"); вывод изображения слайда на экране монитора
  ; задание геометрических параметров в интерактивном режиме
  (setq d   (getreal  "введите – d:")
        d1  (getreal  "введите – l1: ")
        l   (getreal  "введите – l: ")
        p1 (getpoint  "введите  положение базовой точки: ")
        …….
   ) ; окончание функции - setq
(command "redraw") ; удаление изображения слайда с графической зоны
;блок расчета координат узловых точек
  (setq p2 (polar …
```

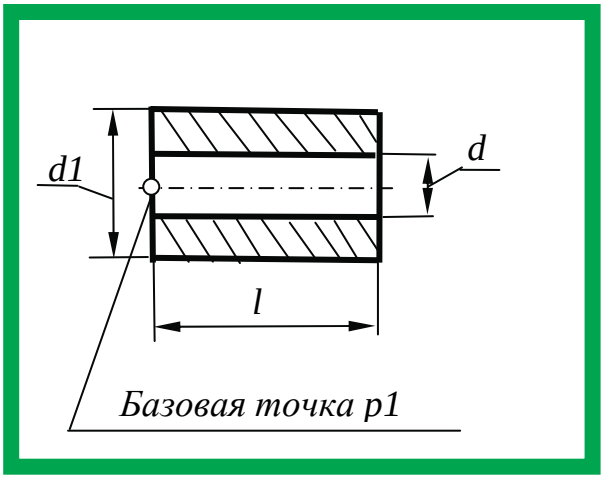


Рис. 5.1. Изображение слайда для ввода геометрических параметров

5.3. Использование меню пользователя при организации интерактивного диалога

Часто при проектировании изделий пользователю необходимо вводить одни и те же данные. В этом случае ввод необходимой информации может быть значительно упрощен с помощью использования меню пользователя. Пусть необходимо ввести значения строковых констант при формировании обозначения сварки, представленного на рис. 4.3. Для упрощения ввода значений параметров сварки используем пользовательское меню. Для этого необходимо составить текст программы (рис. 5.2) с записью в файл с расширением .mnu . Элементы ***POP1, ***POP2, и т. д. программы задают, сколько необходимо использовать элементов падающих меню. Ниже обозначений элементов меню помещаются их названия, например [Вид сварки], [Способ сварки], и т. д. При указании графическим курсором элемента падающего меню ниже появляются опции, например [1 – электродуговая], [2 – газовая] и т. д. При указании данных опций графическим курсором переменной ***a*** (см. ниже) передаются значения строковых констант ГОСТ 5264-80, ГОСТ 11533-75 и т. д.

Меню пользователя может быть установлено в графическом редакторе чертежей Автокада с помощью команды MENU, которая может быть описана текстовой программой на языке AutoLISP. Ниже приведен текст программы, выполняющий интерактивный ввод параметров с помощью пользовательского меню. Заметим, что после запроса значений в интерактивном режиме необходимо графическим курсором указать нужные элементы пользовательского меню, которое появляется вместо стандартного меню системы Автокад.

```
(defun svarka()
  ; программа задания технологических параметров сварки
   (load "menu"  "svarka"); загрузка файл меню с названием “svarka”
  ;блок запроса значений сроковых констант в интерактивном режиме
  (setq      a (getstring "введите вид сварки:")
             b (getstring "введите способ сварки:")
             c (getstring "введите размеры прерывистого вида:")
   )
   (load “menu” “acad”) ;замена пользовательского меню "svarka", меню
                        ; Автокад
); окончание программы  - svarka
```

Если при запросе значения строковой константы b пользователь указывает элемент падающего меню [обозначение шва], а затем выбирает опцию [Т3], то переменной b будет присвоено значение строковой константы "Т3" и т. п.

Текст файла меню пользователя .mnu

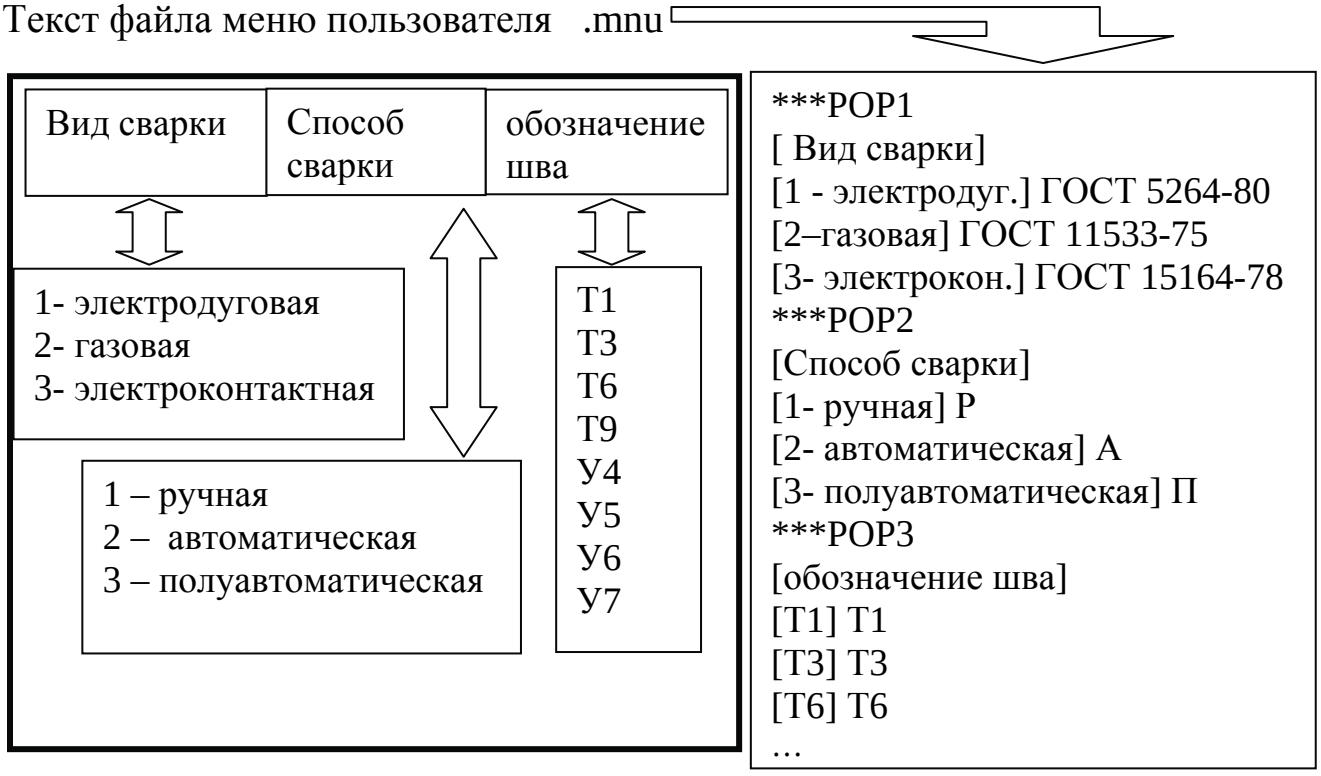


Рис. 5.2. Изображение меню пользователя

Обозначения ***POP1 и т. д. задают номер по порядку элементов падающего меню. С помощью меню пользователя можно осуществлять за-

грузку текстовых программ на языке Автолисп. Ниже приведен пример фрагмента программы меню пользователя, позволяющий проводить загрузку указанных файлов и функций.

```
***POP12
**LOAD
[загрузить]
[создание среды черчения]
[вычертить рамку чертежа](load
"D:/Данные/Студенты/Иванов/ramka.lsp")(ramka)
[задать стиль размеров и слои](load
"D:/Данные/Студенты/Иванов//sapka.lsp")(sapka)
[Загрузка файлов]
[втулка](load "D:/Данные/Студенты/Иванов/vtulka.lsp")(vtulka)
[<-корпус](load "D:/Данные/Студенты/Иванов/korpus.lsp")(korpus)
```

В тексте программы обозначения [загрузить], [создание среды черчения] и т.п. определяют элементы и опции меню пользователя. Запись "D:/Данные/Студенты/Иванов/ramka.lsp" указывает путь к директории, в которой сохраняется искомый файл с расширением .lsp. Записи ramka.lsp, sapka.lsp, … – обозначают имена файлов с расширением .lsp. Записи (ramka), (sapka), … – соответственно задают имена функций.

Создание дополнительных элементов падающего меню графической системы ACAD2006 выполняется по следующей методике (рис. 5.3).

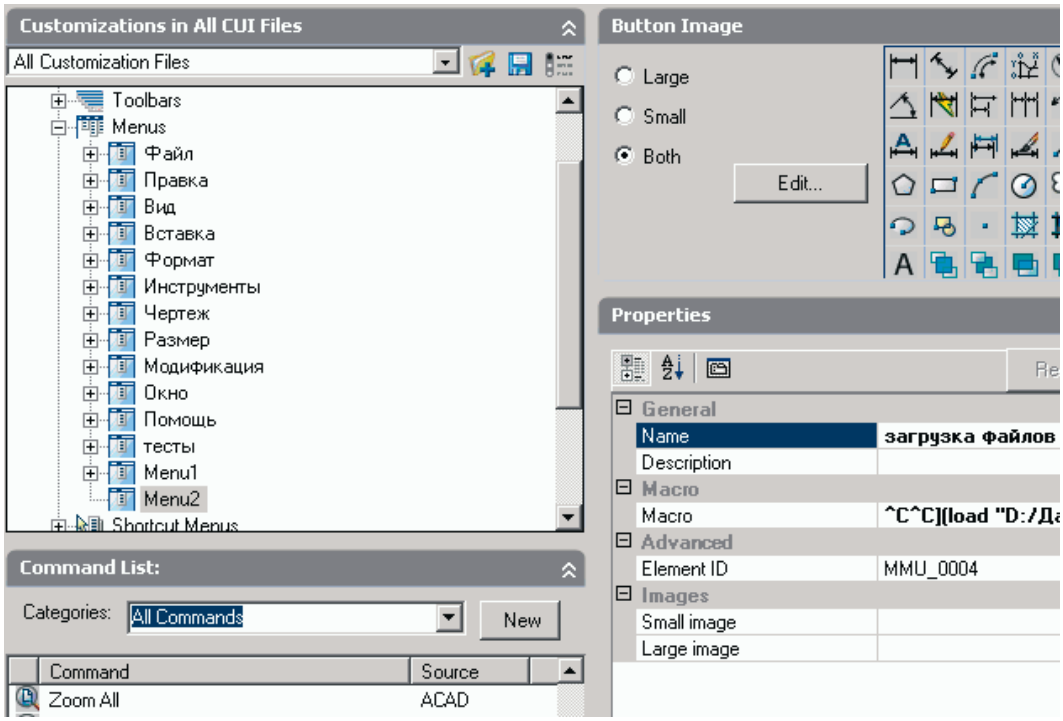


Рис. 5.3. Создание дополнительных элементов падающего меню

1. Щелкнуть по элементам падающего меню в такой последовательности: «Вид» → «Панели инструментов»;

2. В открывшемся окне «Customization Files» выбрать опцию «Menus»;

3. Щелкнуть правой клавишей мыши и выбрать опцию «New»;

4. Задать имя элемента падающего меню:

5. В окне «Command List» щелкнуть по кнопке «New» → «Menu»;

6. В окне «Properties» в поле «Name» задать имя элемента падающего меню;

7. В окне «Properties» в поле «Macro» задать каталог и загружаемый файл.

5.4. Функции ввода данных в файл и извлечение данных из файла

В ряде случаев возникает необходимость в записи некоторой информации в файл, а также в извлечении ее из файла. Автолисп имеет следующие функции, которые решают эти задачи.

- (**open** <имя файла> <режим>). Эта функция, открывая файл, обеспечивает доступ для ввода информации, используя так называемый дескриптор этого файла. ***Дескриптор файла*** не только определяет сам файл, но и содержит информацию о режиме доступа и некоторые системные переменные. Параметр <режим> – это строковая константа. Она состоит из одной буквы "r", "w" или "a" и должна быть набрана на нижнем регистре. При этом, если <режим> задан буквой "r", то файл будет открыт для чтения. Чтобы открыть файл для записи, следует параметр <режим> задать буквой "w". Если <имя файла> не существует, то создается новый файл и он открывается, а если <имя файла> уже существует, то файл будет открыт и в него будут помещены новые данные. Третью букву "a" – применяют для открытия файла и добавления в него информации. Если <имя файла> не существует, то создается новый файл и открывается. Если же <имя файла> уже существует, то новые данные будут помещены в конец записанных ранее.

Пример:

(setq ff (open "new.txt" "w")) – возвращает в виде <файл: #52f4>.

• (**close** <дескриптор файла>). Данная функция закрывает файл, открытый функцией open после ввода в него данных. Если необходимо в этот файл добавить новые данные, то следует вновь использовать функцию open.

Пример: пусть ff – дескриптор открытого файла (см. ваше). Тогда (close ff) закроет файл "new.txt" и возвратит nil.

• (**read-line** <дескриптор файла>). Данная функция считывает данные, вводимые с клавиатуры или из открытого файла, заданного параметром <дескриптор файла>. Она возвращает строку, которая была считана.

Пример:

(setq f (open "mm1.txt" "r")) – создается дескриптор файла с режимом доступа – открыт для чтения.

(setq b1 (atoi (read-line ff)) h1 (atoi (read-line ff)) l1 (atoi (read-line ff)) – переменным b1, h1 и l1 присваиваются значения, заданные в файле mm1.txt

• (**write-line** <строка> <дескриптор файла>). Эта функция используется для записи строковой константы <строка> на экран или в дескриптор файла, заданный параметром <дескриптор файла>. При записи на экране <строка> берётся в кавычки, а при записи в файл кавычки опускаются.

• (**print** <выражение> <дескриптор файла>). При использовании этой функции вводимый текст <выражение> выводится на экран и возвращается в Автолисп. В случае, если присутствует переменная <дескриптор файла> и она является дескриптором открытого для записи файла, то вводимое <выражение> передается на экран и в файл. Это <выражение> печатается без пробелов и без перехода на новую строку.

5.5. Создание параметрических изображений при интерактивном вводе параметров

Отдельные фрагменты текстовой программы, позволяющей получать параметрические изображения разреза и вида слева изделия по рис. 5.4, приведены сразу после рисунка. Данный пример позволяет обобщить представление о структуре программы и назначении её основных блоков.

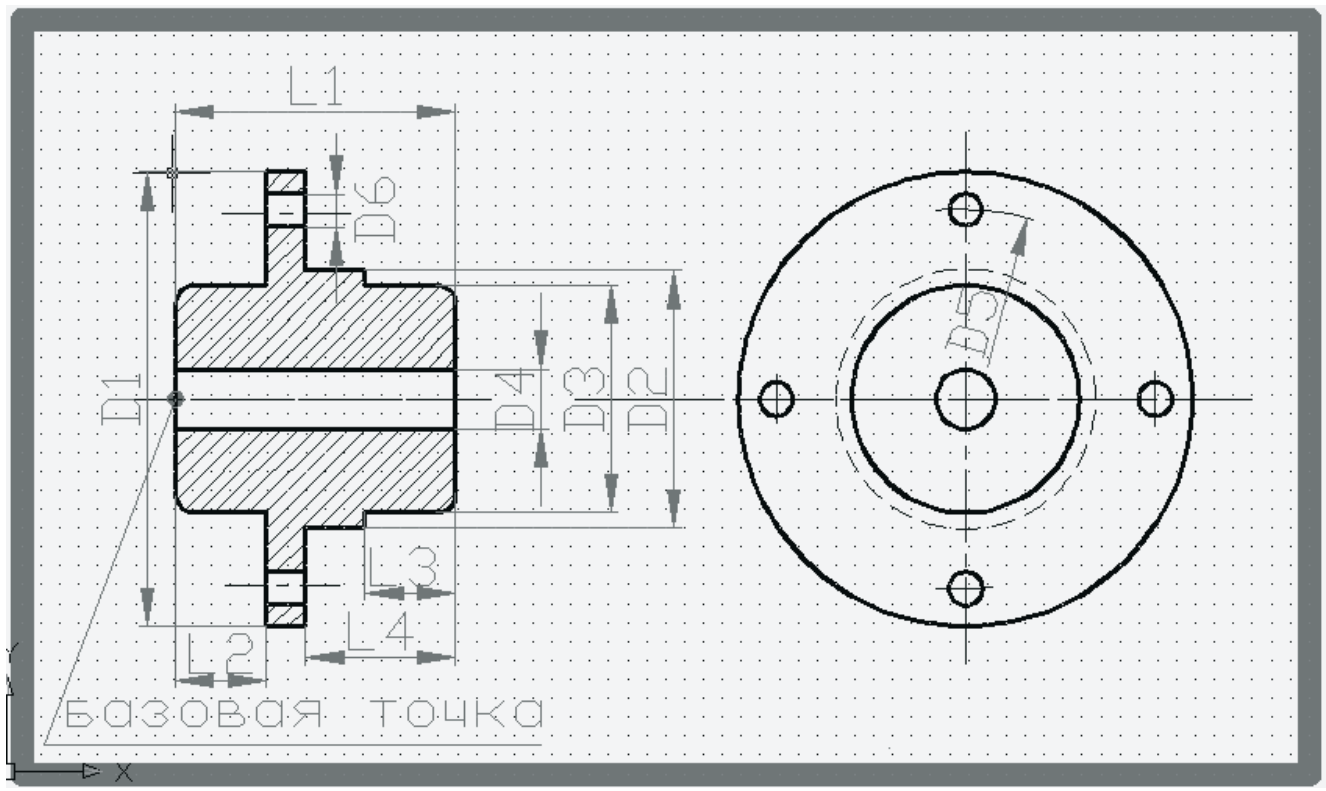


Рис. 5.4. Пример создания слайда текстовой программы

```
(defun st() ;программа позволяет формировать параметрические
            ; изображения видов при различных заданных размерах
            ;детали
 (command "vslide" "st");вывод изображения слайда
 ;блок задание значений переменным ***********************************
      (setq d1 (getreal "Задайте - D1")   d2 (getreal "Задайте - D2")
            d3 (getreal "Задайте - D3")   d4 (getreal "Задайте - D4")
            …
            l3 (getreal "Задайте - L3")     l4 (getreal "Задайте - L4")
      ); окончание ввода значений переменным
  (command "redraw"); удаление изображения слайда
  ;блок создания среды черчения****************************************
     (command "ERASE" "WINDOW" (getvar "VSmin") (getvar "VSmax")
                 "" "redraw") ; удаление изображения с экрана монитора
     (command "Limits" "0,0" "425,297");задание размеров чертежа
     (command "Zoom" "all")(command "SNAP" "off")
     ;задание системных размерных переменных
      (setvar  "DIMASZ" 5.)( setvar "DIMTXT" 5.)( setvar "LUPREC" 0.)
```

```
 (setvar  "DIMEXE" 2.)( setvar "DIMDLI" 0.)
;задание  слоев  и функций установки слоев – sl0, sl1, sl3 и sl4
(command "LAYER" "M" "1"  "C" "5" "1" "" )
(command "LAYER" "M" "2"  "C" "5" "2" "LTYPE" "hidden"  "2" "")
(command "LAYER" "M" "3"  "C" "5" "3" "LTYPE" "dashdot" "3" "")
(command "LAYER" "M" "4"  "C" "3" "4" "")
(defun sl0()(command "LAYER" "S" "0" "")); изобр. основных линий
(defun sl1()(command "LAYER" "S" "1" "")); изобр. тонких линий
(defun sl2()(command "LAYER" "S" "2" "")); изобр. штриховых линий
(defun sl3()(command "LAYER" "S" "3" "")); изобр. штрих-пункт. линий
(defun sl4()(command "LAYER" "S" "4" "")); изобр. размерных линий

(setq    ;ввод параметров на этапе отладки программы***********
 ;x 100 y 100   d1 120 d2 64 d3 60 d4 16 d5 96 d6 10 l1 70 l2 24
l3 24 l4 40
 ; p1 (list x y)
      p1 (getpoint " Задайте базовую точку главного вида")
; блок расчета координат точек главного вида**********************
      p2 (polar p1 1.57 ( / d3 2))          p3 (polar p1 4.71 ( / d3 2))
       …
 ); окончание блока расчета точек главного вида
 ;блок построения изображения главного вида************************
(sl0)(command "line" p20 p2 p3 p27 p31 p6 p7 p8 p9 p21 p20 ""
      "line" p28 p4 p5 p29 p28 ""  "line" p37 p16 p15 p36 p37 "")
      (command "fillet" "r" 5)  (command "fillet" P101 p102
       "fillet" P201 p202  "fillet" P301 p302  "fillet" P401 p402)
(sl1)(command "hatch" "jis_wood" "5" "0" "w" P50 P51 ""
                "hatch" "jis_wood" "5" "0" "w" P50 P52 "")
(sl0)(command "line" p20 p21 p22 p19 p20 ""
     "line" p27 p28 p29 p31 p27 "" "line" p37 p34 p35 p36 p37 "")
(sl3)(command "line" p53 p54 "" "line" p55 p56 ""  "line" p57 p58 ""
     "line" p61 p62 "" "arc" "c" p60 p63 p64 ""  "arc" "c" p60 p66 p65 ""
          "arc" "c" p60 p67 p68 "" "arc" "c" p60 p70 p69 "")
;блок расчета координат точек вида слева**************************
  (setq
```

```
    P73 (polar p1 3.14 (/ d6 2))    p74 (polar p10 0 (+ d1 (+ (/ d1 2) 10)))
      …
  ); окончание блока расчета координат вида слева
 ;блок построения изображения вида слева
    (sl0)(command "circle" p60 (/ d1 2) "circle" p60 (/ d4 2) "circle" p60
(/ d3 2))
    (sl2)(command "circle" p60 (/ d2 2))
    (sl0)(command "circle" p40 (/ d6 2) "circle" p41 (/ d6 2)
                  "circle" p42 (/ d6 2) "circle" p43 (/ d6 2))
    ;Нанесение  размеров на чертеже
    (sl4)(command "dim"
       "vertical" p21 p22 (polar p22 0 10) (strcat "%%c"(rtos d4 2 0)) …
    "exit");окончание нанесения размеров
); окончание функции - defun
```

5.6. Вопросы и задачи для самопроверки

1. Какие функции используют для ввода данных в интерактивном режиме?

2. В чем состоит отличие документов Автокада Слайд и Чертеж, сохраняющихся в файлах с расширениями .sld и .dwg?

3. В чем заключается методика получения слайдов и их использования при вводе данных, характеризующих форму и положение фрагментов параметрических изображений чертежей?

4. В чем заключается методика использования меню пользователя при задании переменным значений строковых констант при автоматизированном формировании текстов технологических обозначений?

5. Назовите функции, позволяющие осуществлять ввод и извлечение данных из файла с программами, написанными на языке Автолисп.

Задача 5.1

Составьте фрагмент текста программы, позволяющей применять метод ввода данных в интерактивном режиме задания значений геометрических параметров l1, l2, h1, h2 и базовую точку, имеющую обозначение p1, с использованием изображения слайда. Информацию о слайде сохраняет файл vtulka.sld.

Задача 5.2

Составьте фрагмент текста программы файла меню с расширением .mnu, позволяющий сформировать меню пользователя, представленного на рис. 5.5.

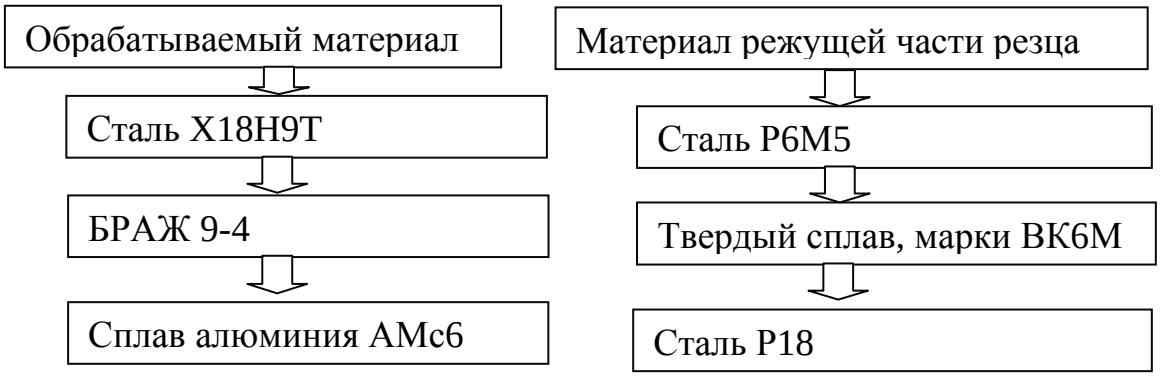


Рис. 5.5. Схема меню пользователя задачи 5.2

Задача 5.3

Составьте фрагмент текста программы файла меню пользователя, позволяющий выполнять автоматизированную загрузку функции (sloi), находящейся в файле sloi1.lsp, и функции (ras), находящейся в файле rasmer.lsp. Номер элемента падающего меню равен десяти и имеет название <среда черчения>. Опциями двенадцатого элемента падающего меню являются <линии чертежа> и <системные размерные переменные №1>. Файлы находятся в директории "D:/Данные/Студенты/"

Задача 5.4

Составьте фрагмент текста программы, позволяющий считывать данные из файла dan.txt с использованием дескриптора файла, имеющего обозначение ff для заданных значений переменных b, h и l.

6. МОДЕЛИРОВАНИЕ ДВИЖЕНИЙ ПРОСТРАНСТВЕННЫХ МЕХАНИЗМОВ РОБОТОВ С ИСПОЛЬЗОВАНИЕМ ТЕКСТОВЫХ ПРОГРАММ НА ЯЗЫКЕ АВТОЛИСП

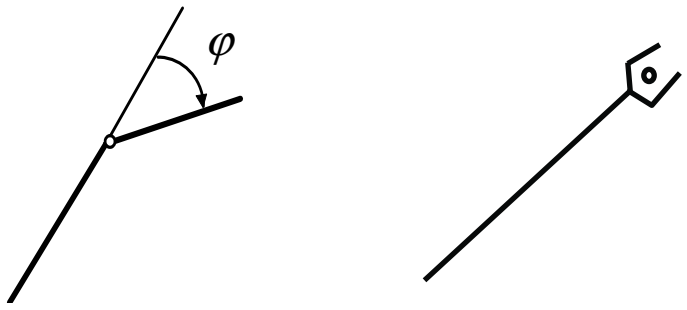
6.1. Основные понятия кинематики и робототехники

Механизм *– подвижно сочлененная совокупность твердых тел (называемых звеньями), предназначенная для преобразования движений.*

Манипулятор *– управляемое устройство робота, представляющее собой пространственный механизм с разомкнутой кинематической цепью, состоящей из нескольких звеньев и кинематических пар, обеспечивающих подвижность этих звеньев.*

Обобщенная координата механизма *– параметр, однозначно определяющий взаимное расположение звеньев, образующих кинематическую пару* (рис. 6.1а). В качестве данного параметра может выступать как угол, так и линейное перемещение. Число обобщенных координат манипулятора равно числу степеней подвижности. Изменение обобщенных координат осуществляется за счет приводов.

Выходное звено (захват) *– конечное звено кинематической цепи манипулятора, движение которого обеспечивает выполнение технологических операций.* На рис. 6.1б показано условное изображение захвата на чертеже.



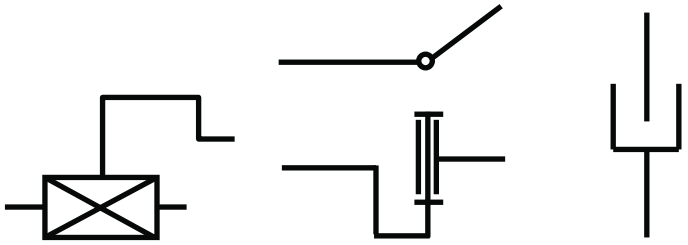
а *б*

Рис. 6.1. Изображение фрагментов кинематической цепи:
а – обобщенная координата; б – выходное звено манипулятора (захват)

Кинематическая пара *– подвижное соединение двух соседних звеньев. Как правило, в робототехнике используются кинематические пары пятого класса (движение определяется одним параметром).* Кинематиче-

ские пары на чертеже принято изображать следующим образом (рис. 6.2): 1 – поступательная кинематическая пара; 2 – вращательная кинематическая пара с осью вращения, перпендикулярной осям стержней звеньев; 3 – вращательная кинематическая пара с осью вращения, совпадающей с осями стержней звеньев.

Конфигурация манипулятора *– положение кинематической цепи, определяемое обобщенными координатами* φ_1 , φ_2 , ... , φ_n *в декартовом пространстве* (рис. 6.3).



1 2 3

Рис. 6.2. Изображения кинематических пар

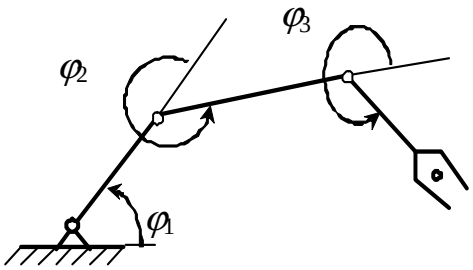


Рис. 6.3. Изображение конфигурации

Кинематическая цепь *– это совокупность звеньев, соединенных в кинематические пары.*

Объект манипулирования *– тело, перемещаемое в неподвижном пространстве манипулятором.*

6.2. Формирование элементов матриц, характеризующих положение подвижных систем координат, которые связаны со звеньями механизмов в неподвижном пространстве

Рассмотрим кинематическую схему четырехзвенного плоского манипулятора, имеющего три вращательные кинематические пары. На рис. 6.4 изображены две конфигурации данного манипулятора. Первая конфигурация соответствует заданным значениям обобщённых координат $\varphi_1 = 0^\circ$, $\varphi_2 = 0^\circ$, $\varphi_3 = 0^\circ$, а вторая – значениям $\varphi_1 = 45^\circ$, $\varphi_2 = -45^\circ$, $\varphi_3 = -45^\circ$. Свяжем с каждым звеном механизма соответственно системы координат $0_0 x_0 y_0 z_0$, $0_1 x_1 y_1 z_1$, $0_2 x_2 y_2 z_2$ и $0_3 x_3 y_3 z_3$. Данные системы, которые для краткости будем обозначать 0_0 , 0_1 , 0_2 , 0_3, определяют положения звеньев механизма. Для удобства описания в матричной форме кинематической

схемы манипулятора следует использовать следующие правила задания систем координат, связанных со звеньями механизма. Если два соседних звена соединены вращательной кинематической парой, то одну из осей координат следует направить по оси вращательной пары, другую – по оси звена, а третью – так, чтобы система координат была правой. Когда звенья образуют поступательную кинематическую пару, одну из координатных осей направляют по линии относительного движения звеньев, а две другие – параллельно соответствующим осям предыдущего звена, обеспечивая при этом также правую систему координат.

Положение выходного звена (захвата) в неподвижном пространстве определяется r параметрами : x_n , y_n , z_n , θ_n , ψ_n , φ_n . Данные параметры однозначно задают положение системы $O_n x_n y_n z_n$, связанной с n-звеном (для краткости систему координат $O_n x_n y_n z_n$ будем обозначать O_n). Для плоского манипулятора число параметров r равно трем. Два параметра (x_n , y_n) определяют положение центра системы O_n и один параметр (α) задаёт поворот данной системы относительно инерциальной системы координат O_0 (рис. 6.5а).

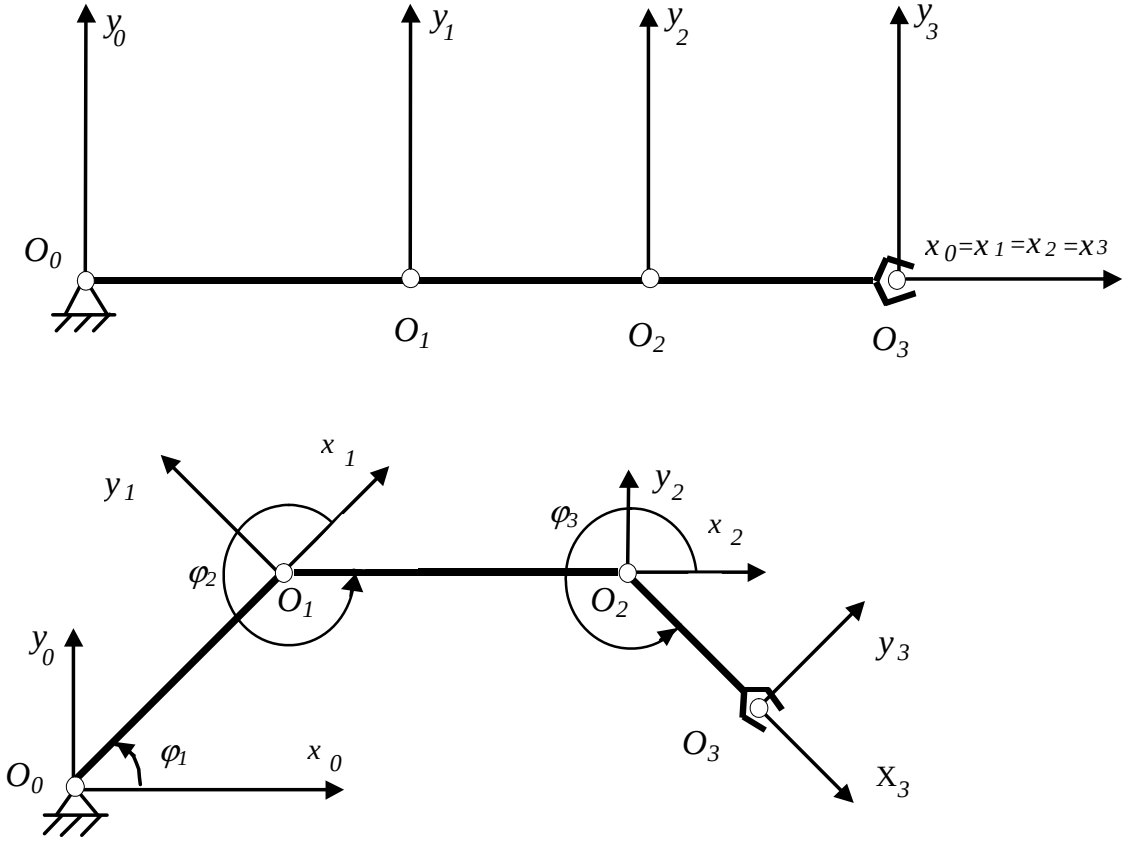


Рис. 6.4. Положение подвижных систем координат, связанных со звеньями механизма

Для пространственного манипулятора $r = 6$.Три параметра (x_n , y_n , z_n) определяют начало координат системы O_n и три параметра (θ_n , ψ_n , φ_n) – три угла Эйлера (рис 6.5б) . Углы Эйлера определяют ориентацию системы O_n в неподвижной системе O_o **.**

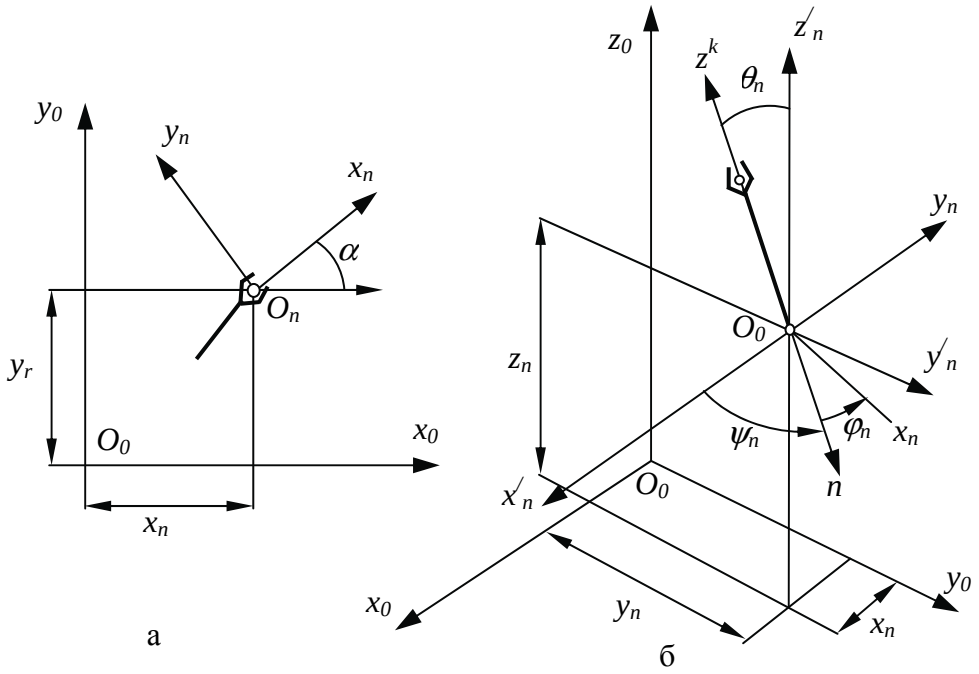


Рис. 6.5. Геометрические параметры, задающие положение выходного звена в неподвижном пространстве

Условимся положение *k*-го звена определять вектором $\boldsymbol{X_k}$ (x_k , y_k , z_k ,θ_k , ψ_k , φ_k), положение захвата – соответственно вектором $\boldsymbol{X_n}$ (x_n, y_n, z_n,θ_n,ψ_n , φ_n) **.** Условимся значения обобщенных координат задавать вектором $\boldsymbol{Q}$ (φ_1 , φ_2 , φ_3). Каждому значению вектора $\boldsymbol{Q}$ будет соответствовать единственная конфигурация или значение вектора $\boldsymbol{X_n}$. При заданной структуре кинематической цепи манипулятора, определяемой исходным набором применяемых кинематических пар, последовательностью их расположения и заданными длинами звеньев механизма, значению вектора $\boldsymbol{L}_n$ (φ_1 , φ_2 , … ,φ_n) соответствует единственное значение вектора $\boldsymbol{X}_n$ (x_n , y_n , z_n , θ_n , ψ_n , φ_n) .

$$
\begin{aligned}
x_n &= f_x(\varphi_1, \varphi_2, \ldots, \varphi_n) \ , \\
&\ldots\ldots \\
\varphi_n &= f_x(\varphi_1, \varphi_2, \ldots, \varphi_n) \ .
\end{aligned}
\qquad (6.1)
$$

Данные функции в аналитическом виде легко определяются матричным произведением [11]:

$$\boldsymbol{M}_{0,n} = \boldsymbol{M}_{0,1} \times \boldsymbol{M}_{0,2} \times \ldots \times \boldsymbol{M}_{k-1,k} \times \ldots \times \boldsymbol{M}_{n-1,n} \quad , \qquad (6.2)$$

где $\boldsymbol{M}_{k-1,k}$ – матрицы, определяющие переход от системы O_k к системе O_{k-1} .

$$\boldsymbol{M}_{k-1,k} = \begin{bmatrix} \cos(x_k \wedge x_{k-1}) & \cos(y_k \wedge x_{k-1}) & \cos(z_k \wedge x_{k-1}) & x_{k,k-1} \\ \cos(x_k \wedge y_{k-1}) & \cos(y_k \wedge x_{k-1}) & \cos(z_k \wedge y_{k-1}) & y_{k,k-1} \\ \cos(x_k \wedge z_{k-1}) & \cos(y_k \wedge x_{k-1}) & \cos(z_k \wedge z_{k-1}) & z_{k,k-1} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad (6.3)$$

где $x_k \wedge x_{k-1}$ – углы, образованные координатными осями системы O_k и O_{k-1} (углы считаются положительными, если поворот системы O_k относительно системы O_{k-1} происходит против часовой стрелки при наблюдении его с положительного направления осей $x_{k,k-1}$ $y_{k,k-1}$, $z_{k,k-1}$, вокруг которой совершается поворот); $x_{k,k-1}$; $y_{k,k-1}$, $z_{k,k-1}$ – координаты начала системы O_k в системе O_{k-1} .

При повороте системы O_k относительно оси x_{k-1} матрица (6.3) принимает следующий вид:

$$\boldsymbol{M}_{k-1,k} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi_{k-1,k} & -\sin\varphi_{k-1,k} & l_x \cos\varphi_{k-1,k} \\ 0 & \sin\varphi_{k-1,k} & \cos\varphi_{k-1,k} & l_x \sin\varphi_{k-1,k} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad (6.4)$$

где l_x – расстояние от начала координат системы O_k до оси вращения $O_{k-1}x_{k-1}$. Данное расстояние, как правило, определяет длина звеньев механизма (перенос центра системы O_{k-1} вдоль одной из её осей до полного совмещения с системой O_k при $\varphi_{k-1,k} = 0$). Перенос вдоль оси вращения отсутствует.

Дальше представлен текст программы на языке Автолисп, позволяющей формировать значения элементов матриц (6.4).

```
;==================== начало - mvx ====================
 (defun mvx (ugol dlina smex)
;программа расчета элементов матриц Мк,к+1 при вращении
;вокруг оси ox
;ugol-обобщенная координата,
;dlina – длина звена,
;smex-смещение вдоль оси вращения.
(setq ss1$ (list 0 1 0 0 smex
          0 (cos (dtr ugol)) (- 0 (sin (dtr ugol)))  (* dlina (cos (dtr ugol)))
          0 (sin (dtr ugol))  (cos (dtr ugol))        (* dlina (sin (dtr ugol)))
          0          0                  0                          1))
);==================== конец - mvx ====================
```

При использовании поступательных кинематических пар, позволяющих осуществлять перенос системы O_k вдоль осей $O_{k-1}x_{k-1}$, $O_{k-1}y_{k-1}$, $O_{k-1}z_{k-1}$, при условии совпадения вектора переноса с направлениями данных осей матрицы преобразования (6.3) примут вид:

$$\boldsymbol{M}_{k-1,k}=\begin{bmatrix} 1 & 0 & 0 & S_{k-1,k} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \boldsymbol{M}_{k-1,k}=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & S_{k-1,k} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{M}_{k-1,k}=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & S_{k-1,k} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.5)$$

Ниже представлен текст программы на языке Автолисп, позволяющей формировать значения элементов матриц (6.5).

```
;================== начало - mpxyz ====================
(defun mpxyz (xp yp zp)
;программа расчета элементов матриц Мк,к+1 при поступательном
;смещении  вдоль  осей x, y и z
;xp – смещение вдоль оси x,
;yp – смещение вдоль оси y,
;zp – смещение вдоль оси  z.
;прог.соз.элем.матриц Мк,к+1 при пос.дв.в доль осей x,y,z
```

```
 (setq mm$  (list   0          1          0.000001    0.000001      xp
                              0.000001     1         0.000001      yp
                              0.000001   0.000001      1           zp
                                0          0           0           1) )
);==================== конец - mpxyz ====================
```

Дальше приведен текст программы, позволяющей определять элементы матриц $\boldsymbol{M}_{k-1,k}$ по заданным значениям обобщенных координат, заданных списком ui, значениям длин звеньев механизмов, заданных списком li, смещениям вдоль осей вращения, заданных списком smi, и кодам кинематических преобразователей, заданных списком kodi. Коды кинематических преобразователей имеют следующие значения:

1 – 3 – с использованием вращательных кинематических пар вокруг осей $O_{k-1}x_{k-1}$, $O_{k-1}y_{k-1}$, $O_{k-1}z_{k-1}$ соответственно;

4 – 6 – поступательных кинематических пар вдоль осей $O_{k-1}x_{k-1}$, $O_{k-1}y_{k-1}$, $O_{k-1}z_{k-1}$ соответственно.

```
;=============== начало - call_coord ======================
(defun calc_mkn (ui li smi kodi)
;программа вычисления элементов матриц Mk,k+1 по заданным
;значениям обобщенных координат и списку кодов кинематических
;преобразователей
;ui – список значений обобщенных координат,
;li – список значений длин звеньев механизма,
;smi – список смещений вдоль осей вращений,
;kodi – список кодов кинематических преобразователей,
;элементы матриц Mk,k+1 заданных списком mkn.
;nmat – число матриц, используемых при описании модели кинемат. цепи
(setq n 0)
      (while (< n nmat)
        (setq  ui$ (nth (+ n 1) ui) li$ (nth (+ n 1) li)
               smi$ (nth n smi) kodi$ (nth n kodi))
```

```
(if (= kodi$ 1)(setq mat (mvx ui$ li$ smi$)))
(if (= kodi$ 2)(setq mat (mvy ui$ li$ smi$)))
(if (= kodi$ 3)(setq mat (mvz ui$ li$ smi$)))
(if (= kodi$ 4)(setq mat (mpxyz ui$ li$ smi$)))
(if (= kodi$ 5)(setq mat (mpxyz 0 ui$ 0) ))
(if (= kodi$ 6)(setq mat (mpxyz 0 0 ui$) ))
(setq mkn (subs mkn (+ n 2) nmat mat) n (+ n 1) )    );while
);================= конец - calc_mkn ======================
```

В результате перемножения матриц $\boldsymbol{M}_{k-1,k}$ в выражении (6.2) получаем компоненты матрицы произведения:

$$\boldsymbol{M}_{0,k} = \begin{bmatrix} m^{0,n}_{11} & m^{0,n}_{12} & m^{0,n}_{13} & m^{0,n}_{14} \\ m^{0,n}_{21} & m^{0,n}_{22} & m^{0,n}_{23} & m^{0,n}_{24} \\ m^{0,n}_{31} & m^{0,n}_{32} & m^{0,n}_{33} & m^{0,n}_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad (6.6)$$

где $m^{0,n}_{14}$, $m^{0,n}_{24}$, $m^{0,n}_{34}$ – определяют координаты x_n , y_n , z_n , являющиеся компонентами вектора $\boldsymbol{X_n}$ **.** Остальные три компонента данного вектора (углы Эйлера) определяем, используя выражения:

$$\begin{aligned} \theta_n &= \arccos m^{0,n}_{33} ; \\ \psi_n &= \arcsin (m^{0,n}_{13} / \sin \theta_n) ; \qquad (6.7) \\ \varphi_n &= \arcsin (m^{0,n}_{31} / \sin \theta_n) . \end{aligned}$$

Таким образом, используя компоненты матрицы (6.6) и данные уравнения, однозначно находим функции (6.2, 6.7), определяющие положение системы O_n или положение выходного звена по вектору $\overline{\boldsymbol{L}_n}$. Аналогичным образом для определения положения k-го звена или значений вектора $\overline{\boldsymbol{X}_k}$

необходимо получить элементы $m^{0,k}$ матрицы (6.6), полученной умножением (6.2) *k*-матриц.

Приведем текст программы, позволяющей выполнять определение элементов матриц произведения $\boldsymbol{M}_{0,k}$ и координат узловых точек механизма манипулятора на фронтальной и горизонтальной проекциях.

```
;==================== начало - call_coord =================
(defun calc_coord ()
;Подпрограмма осуществляющая расчет матриц Mo,k определяющих
положение звеньев механизма и проекций координат узловых точек
;конфигураций
  (sp-calc_coord) (calc_mkn ui li smi kodi) (setq n 0)
;Вычисление матриц M o,k задающих положение звеньев в неподвижной
;системе координат.
 (while (< n nmat)
  (setq moki (mprd moki (nth ( + n 2) mkn) 4) mok (subs mok (+ n 2) nmat moki)
    ;вычисление координат  узловых точек манипулятора на фронтальной
    ;проекции определяемых списком – pfr –
    pfri (list (+ vv (nth 4 moki)) (+ ez (nth 12 moki)))
    pfr (subs pfr (+ n 2) nmat pfri)
    ;вычисление координат  узловых точек манипулятора
    ;на горизонтальной проекции определяемых списком – pgr –
    pgri (list (+ vv (nth 4 moki))  (nth 8 moki))
    pgr (subs pgr (+ n 2) nmat pgri)
    ;вычисление координат  узловых точек манипулятора на профильной
    ;проекции определяемых списком – ppr –
    ppri (list (- vvp (nth 8 moki)) (+ ez (nth 12 moki)))
    ppr (subs ppr (+ n 2) nmat ppri) n (+ n 1) ));setq-while
);================== конец - call_coord ====================
```

Программа calc_coord использует подпрограмму (sp-calc_coord) для определения размерности списков mkn, pfr, pgr, ppr и подпрограмму mprd, предназначенную, для перемножения матриц размером 4×4.

```
;===== ========== начало - sp-calc_coord ==================
(defun sp-calc_coord();программа определяет размерность списков
;в зависимости от значения - nmat
      (setq n 1 mkn (list (list 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1) 0)
            mok (list (list 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1) 0)
            moki (list 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1) pfr (list (list vv ez) 0)
            pgr (list (list vv 0) 0) ppr (list (list vvp ez) 0) )
      (while (<= n (- nmat 1))
        (setq mkn (cons 0 mkn) mok (cons 0 mok)
              pfr (cons 0 pfr) pgr (cons 0 pgr)
              ppr (cons 0 ppr) n (+ n 1))
      )
     (setq mkn (reverse mkn) mok (reverse mok) pfr (reverse pfr)
            pgr (reverse pgr)      ppr (reverse ppr))
);==================== конец - sp-calc_coord ================
```

6.3. Моделирование движения робота РБ-211 на горизонтальной, фронтальной и профильной плоскостях проекций

На рис. 6.6 представлено наглядное изображение робота РБ-211 и приведены геометрические параметры, характеризующие модель кинематической цепи. На рис. 6.7 изображены системы координат, неподвижно связанные со звеньями механизма робота. В табл. 6.1 приведены геометрические параметры, характеризующие модель кинематической цепи исполнительного механизма.

Таблица 6.1

Значения списков ui, li, smi и kodi программы *post*, определяющих геометрическую модель механизма робота РБ-211

Семизвенный пространственный механизм робота РБ-211								
$n = 6,\ m_n = 7$	ui	φ_1	φ_2	φ_3	φ_4	φ_5	-φ-	φ_6
		-85	130	-110	90	135	0*	0
	li	- l -	l_1	l_2	l_3	l_4	- l -	l_6
		0	940	1850	200	350	0*	200
	smi	- s -	- s -	- s -	- s -	- s -	l_5	- s -
		0	0	0	0	0	650	0
	kodi	$k1$	$k2$	$k3$	$k4$	$k5$	$k6$	$k7$
		3	1	1	3	2	9	3

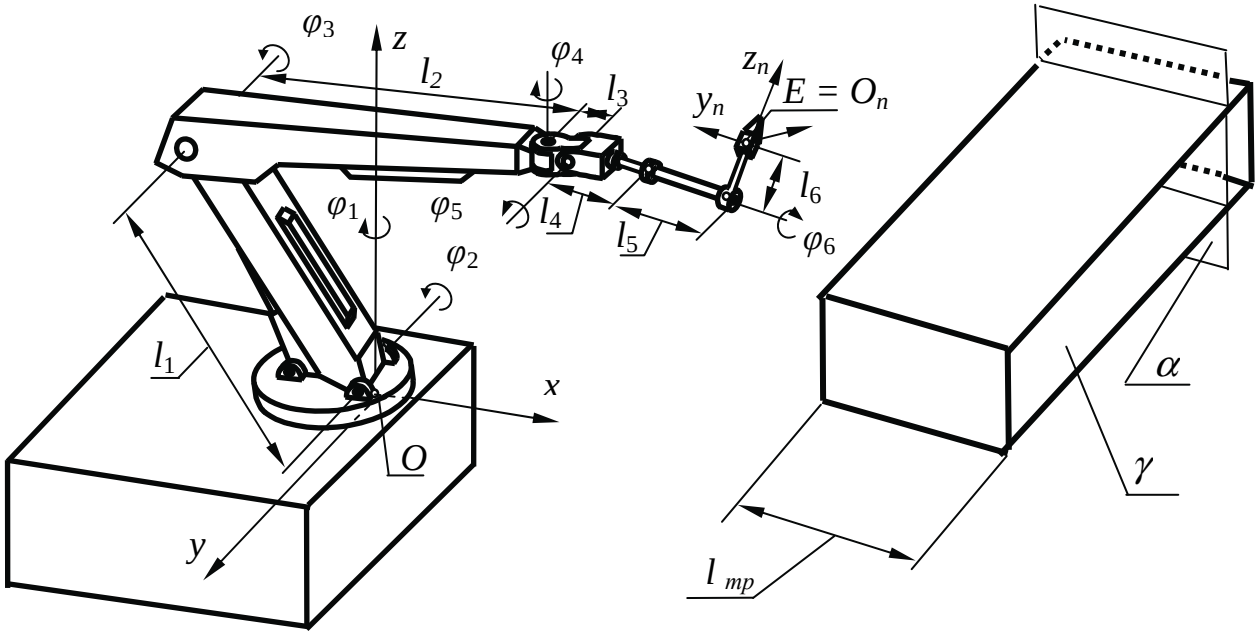


Рис. 6.6. Общий вид и геометрические параметры, характеризующие исполнительные механизмы робота РБ-211

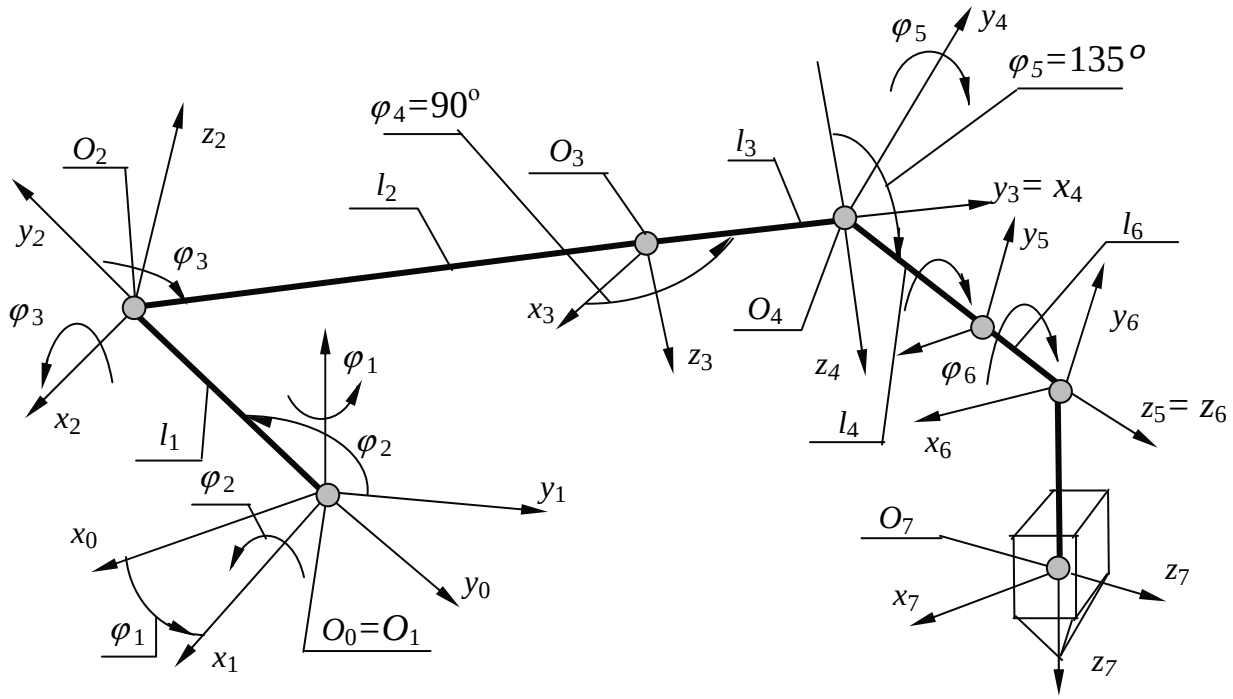


Рис. 6.7. Положение систем координат, используемых для задания геометрической модели механизма манипулятора РБ-211

Далее приведен текст основной программы, позволяющей строить промежуточные конфигурации при синтезе малых движений манипулятора.

```
;==================== начало – post  ====================
(defun post ()
;программа синтеза малых движений робота РБ-211
   (setq nz 6 ;число обобщенных координат механизма манипулятора
   nmat 7 ;кол-во узловых точек или кол-во матриц M(k-1,k)
   ui    ;список определяющий значения обобщенных координат - U1,U2,
         (list 0 -125    130      -80     90      205    0         0)
   li     ;длины звеньев механизма манипулятора - l1,l2, ...
          (list 0   0     94     185    20     35      0       20)
   smi ;смещения вдоль осей вращения кинематических пар - u1s,u2s, ...
         (list    0      0       0      0       0       65        0)
   kodi ;значения кодов кинематических преобразователей,
         (list     3       1        1       3        2        9          3)
      )
     )
   ;проверка достижимости целевой точки, заданной координатами ;xc2 zc2
     (while (> (distance (list xc2 zc2) (nth (+ nmat 1) pfr)) (/ modv 100))
     (move)  (draw_manip 2)
     )
);==================== конец – post  ======================
```

Программа move позволяет установить величину приращения обобщенных координат [10]. При этом вектор приращений обобщенных координат определяется на основе критерия минимизации объема движения и учета пересечения механизма робота с препятствиями [12]. Для построения изображений конфигураций манипулятора используется программа draw_manip.

```
);================= начало - draw_manip ==================
(defun draw_manip (rm)
;Подпрограмма формирующая изображения конфигураций
;на плоскостях проекций.
```

```
   (calc_coord);rm - радиус окружности, изображающей узловые точки
   (while (< n (+ nmat 1))
   (linee'(((nth n pfr)(nth (+ n 1) pfr))))(command "circle" (nth (+ n 1) pfr) rm)
   (linee'(((nth n pgr)(nth (+ n 1) pgr))))(command "circle" (nth (+ n 1) pgr) rm)
   (linee'(((nth n ppr)(nth (+ n 1) ppr))))(command "circle" (nth (+ n 1) ppr) rm)
   (setq n (+ n 1))
   )
);====================== конец - draw_manip ===============
```

Для соединения точек отрезками в программе draw_manip используется программа linee.

```
;================== начало - linee ========================
(defun linee ($$f$$)
;программа для соединения отрезками точек, заданных списком – $$f$$
  (foreach $nak $$f$$ (command “line” (foreach $dfr $nak
                                        (command  (eval $dfr))) ))
);================== конец – linee ========================
```

На рис. 6.8 представлены результаты моделирования робота РБ-211.

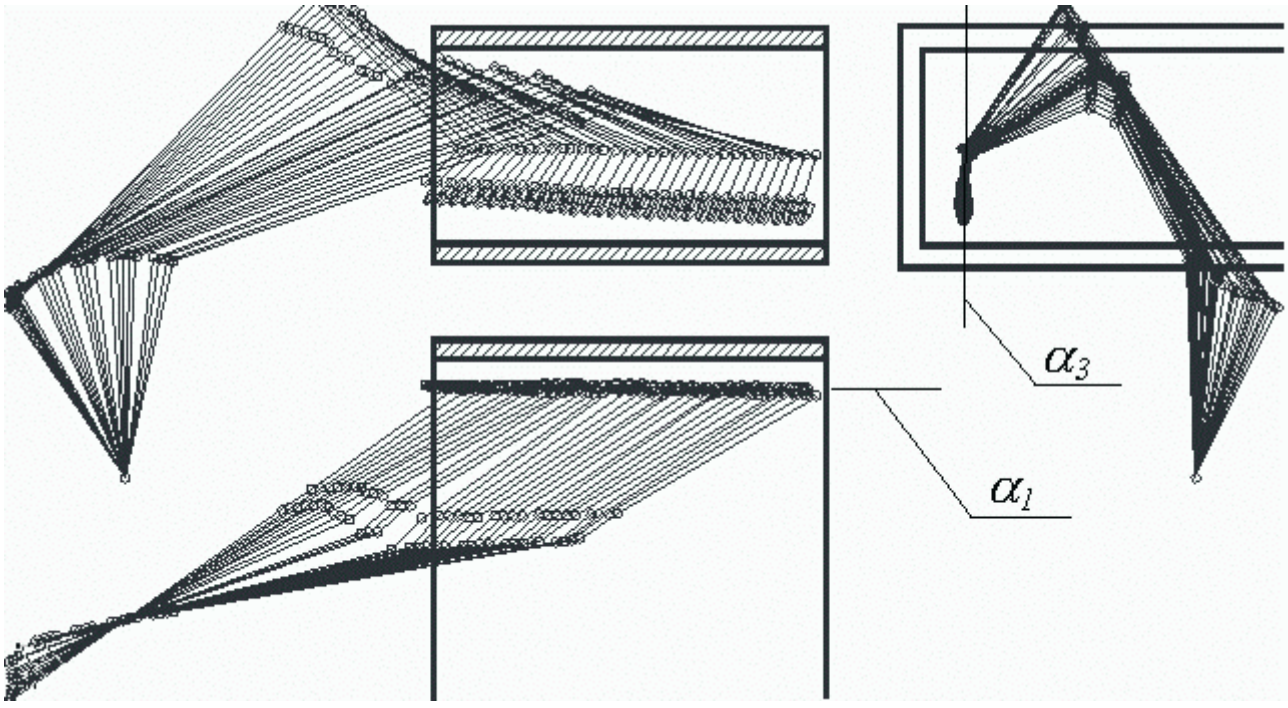


Рис. 6.8. Результаты синтеза малых движений робота РБ-211

6.4. Вопросы и задачи для самопроверки

1. Какие функции языка Автолисп используются для формирования элементов матриц $\boldsymbol{M}_{k-1,k}$?

2. Какие списки параметров необходимо составить для задания моделей кинематических цепей роботов?

3. Каким образом можно осуществить построение конфигураций манипулятора на различных плоскостях проекций?

4. В чем заключается методика составления текстовой программы, предназначенной для вычисления координат узловых точек механизма манипулятора?

5. В чем состоит назначение программы post?

Задача 6.1

Составьте текст программы, позволяющей выполнять расчет элементов матрицы $\boldsymbol{M}_{k-1,k}$ при повороте системы O_k вокруг оси y_{k-1}:

$$\boldsymbol{M}_{k-1,k} = \begin{bmatrix} \cos\varphi_{k-1,k} & 0 & \sin\varphi_{k-1,k} & l_y \sin\varphi_{k-1,k} \\ 0 & 1 & 0 & 0 \\ -\sin\varphi_{k-1,k} & 0 & \cos\varphi_{k-1,k} & l_y \cos\varphi_{k-1,k} \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

где l_y – расстояния от начала координат системы O_k до оси вращения $O_{k-1}y_{k-1}$.

Задача 6.2

Составьте текст программы, позволяющей выполнять расчет элементов матрицы $\boldsymbol{M}_{k-1,k}$ при повороте системы O_k вокруг оси z_{k-1}:

$$\boldsymbol{M}_{k-1,k} = \begin{bmatrix} \cos\varphi_{k-1,k} & -\sin\varphi_{k-1,k} & 0 & l_z \cos\varphi_{k-1,k} \\ \sin\varphi_{k-1,k} & \cos\varphi_{k-1,k} & 0 & l_z \sin\varphi_{k-1,k} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

где l_z – расстояния от начала координат системы O_k до осеи вращения $O_{k-1}z_{k-1}$.

Задача 6.3

Запишите значения элементов списков ui, li, smi и kodi, задающих модель кинематической цепи робота, изображенного на рис. 6.1, если значения обобщенных координат соответственно равны $\varphi_1=50$, $\varphi_2=120$, $\varphi_3=30$, а длины звеньев механизма соответственно равны $l_1 = 100$, $l_2 = 90$, $l_3 = 80$.

7. СОЗДАНИЕ СИСТЕМЫ ПРОВЕРКИ ГРАФИЧЕСКИХ ПОСТРОЕНИЙ, ВЫПОЛНЯЕМЫХ СТУДЕНТАМИ ПРИ РЕШЕНИИ ЗАДАЧ В КУРСЕ «НАЧЕРТАТЕЛЬНАЯ ГЕОМЕТРИЯ И ИНЖЕНЕРНАЯ ГРАФИКА»

7.1. Назначение основных блоков программного обеспечения системы для оценки правильности графических построений

В настоящее время интенсивно внедряются в учебный процесс интегрированные информационные формы обучения. Одной из важнейших задач при этом остается разработка систем тестирования знаний студентов. В данном направлении имеется множество разработок, однако все они в основном построены на основе простого выбора студентом предлагаемых вариантов ответов. Это не всегда является объективной оценкой знаний студентов. Особенно это касается изучения таких дисциплин, как «Начертательная геометрия» и «Инженерная графика», где основной задачей студентов является выполнение графических построений на плоскости чертежа. В связи с этим существует проблема автоматизированного анализа и оценки графических построений при решении студентами той или иной задачи дисциплин «Начертательная геометрия» и «Инженерная графика» при использовании графических систем.

В данной главе предложены решения указанной проблемы на основе использования пакета САПР АВТОКАД и алгоритмического языка программирования АВТОЛИСП. В основу разработанного алгоритма оценки знаний положено использование функций АВТОЛИСП, обеспечивающих доступ к построенным студентом примитивам и их обработке в соответст-

вии с формулированной задачей. Заметим, что студент по каждой отдельной задаче выполняет строго заданный и ограниченный набор базовых графических построений. Поэтому существует возможность оценки данных построений по заданному эталонному решению. Программное обеспечение тестирующей системы представляет собой блочную структуру, схема которой приведена на рис. 7.1 .

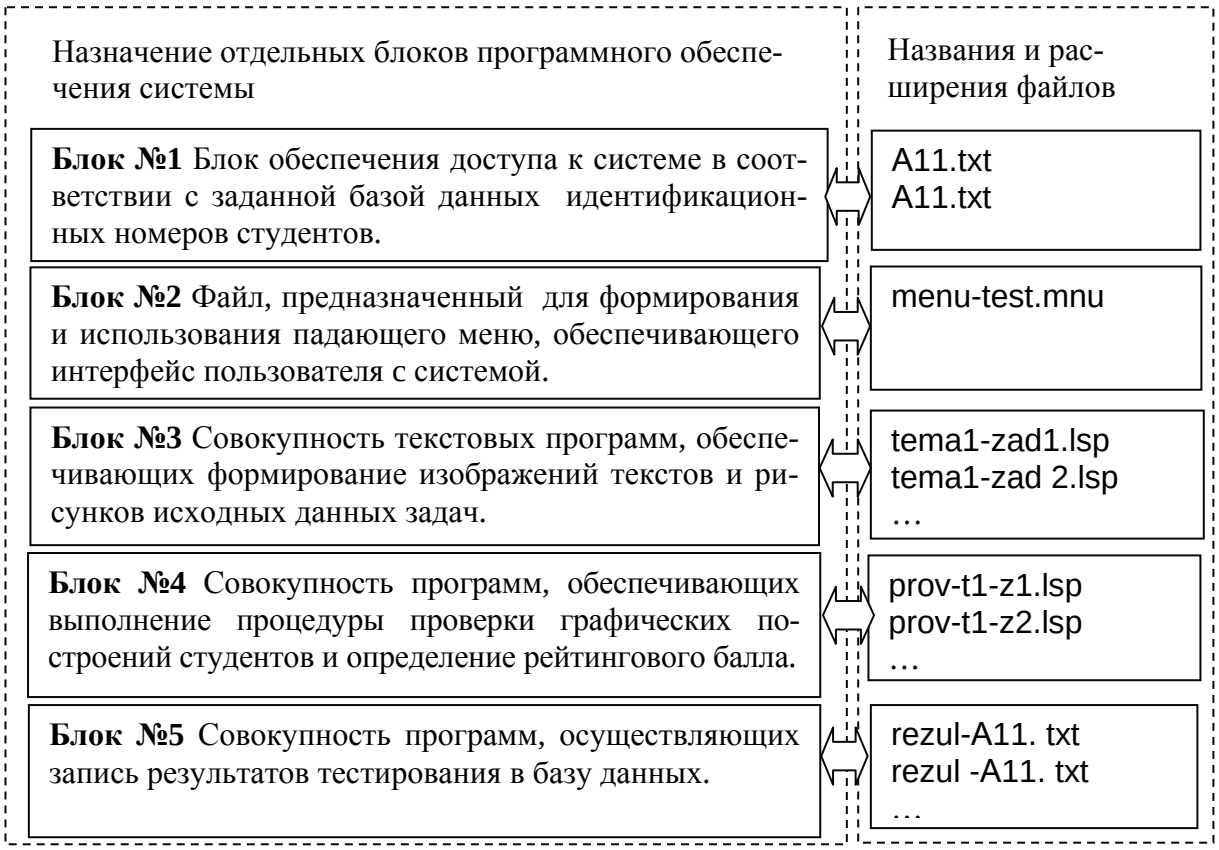


Рис. 7.1. Блоки программного обеспечения системы проверки графических построений

Первый блок предназначен для контроля доступа студентов к тестирующей системе. При этом системой проверяется идентификационный номер студентов в соответствии с заданной заранее базой данных, которая хранится в файлах имеющих названия групп и расширение *.txt* (*A11.txt, A12.txt, …*). После регистрации студентом выбираются различные процедуры работы с системой или самообучение, или тестирование. Доступ к контролю знаний по различным темам возможен только один раз.

Второй блок содержит файл, позволяющий формировать изображение падающего меню системы проверки графических построений студента. Падающее меню позволяет пользователю осуществлять выбор соответст-

вующей темы и задачи (рис. 7.1). При указании необходимой темы и задачи происходит автоматическое формирование изображения исходных данных проверочной задачи. Если студентом указывается элемент падающего меню Проверить, происходит автоматическая проверка графических построений студента. На рис. 7.2 представлено изображение меню системы автоматизированной проверки графических построений при выборе пользователем первой темы и задачи 5.

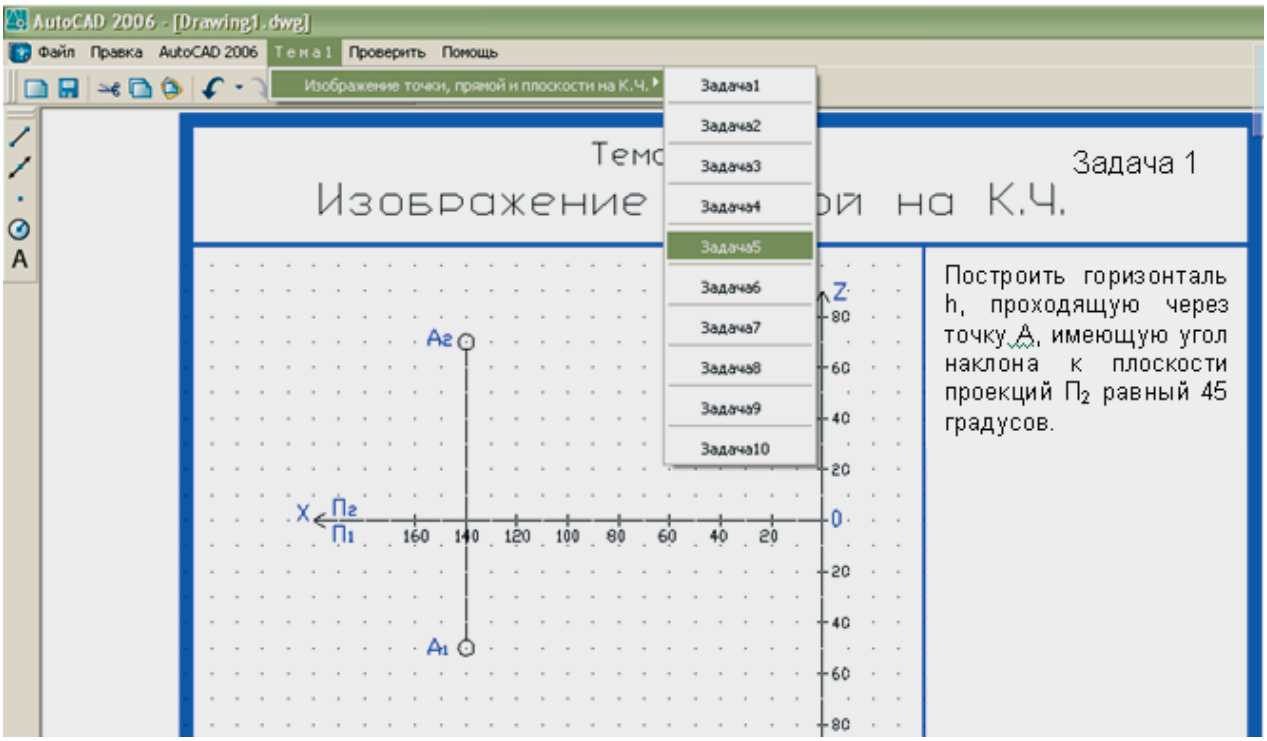


Рис. 7.2. Изображение исходных данных задачи при проверке правильности графических построений

Третий блок содержит совокупность программ (tema1-zad1.lsp, tema1-zad2.lsp, …), описанных на языке AutoLISP, позволяющих получать изображение исходных данных проверочных задач. В данном файле задаются соответствующие координаты точек, длины, углы и др., которые используются при получении изображений исходных данных. Указанные программы автоматически создают среду черчения и условие тестовой задачи. На рис. 7.3 приведен пример изображения исходных данных одной из проверочных задач, связанной с построением горизонтали, проходящей через заданную точку под заданным углом к фронтальной плоскости проекций. Как видно из рисунка, изображение исходных данных состоит из трех основных зон . В первой зоне отражается номер темы и задачи, а

также приводится название темы. Во второй зоне приводится текст задачи. В этой же зоне появляются в дальнейшем комментарии после осуществления процедуры проверки тестовой задачи. В частности, в этой зоне выводятся анализ и распределение рейтинговых баллов на различных этапах решения заданной проверочной задачи. В третьей зоне приводится изображения рисунка исходных данных (оси комплексного чертежа и их обозначения, линии проекционной связи, обозначений проекций геометрических объектов, заданных в условии задачи и др). Для упрощения ввода точек при решении некоторых задач используется шаг смещения графического курсора и сетка с определенно заданным параметром. Исходные данные размещаются в слое, недоступном пользователю системы оценки графических построений.

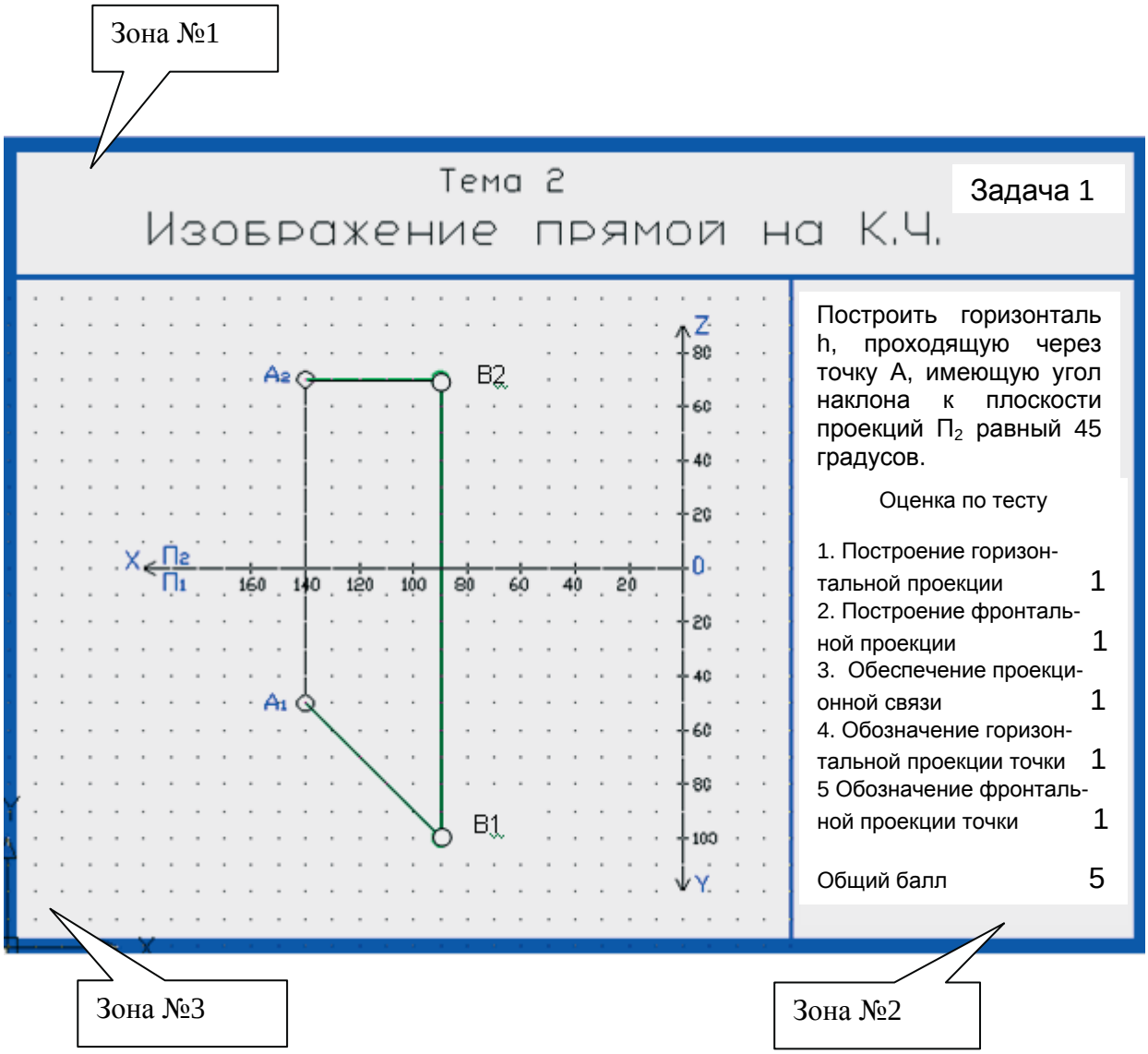


Рис. 7.3. Содержание графической зоны системы после осуществления операции проверки графических построений

После формирования изображения тестовой задачи студент выполняет графические построения на заданном рисунке зоны №3. Графические построения выполняются с использованием панелей инструментов системы САПР ACAD. При этом на рабочем столе размещаются только необходимые панели и необходимые изображения пиктограмм панелей (см. рис. 7.2). После окончания построений пользователь с помощью падающего меню выбирает процедуру проверки текущей проверочной задачи.

Четвертый блок программ тестирующей системы содержит набор файлов, позволяющих осуществлять проверку правильности построений студентов. Каждый отдельный файл (prov-t1-z1.lsp, prov-t1-z2.lsp, …) соответствует той или иной тестовой задаче (tema1-zad1.lsp, tema1-zad2.lsp, …). При выполнении процедуры проверки тестовой задачи программа вначале создает наборы из отдельных примитивов, построенных студентом. Далее происходит сортировка списка наборов примитивов по различно заданным признакам. При этом используются функции для обработки данных о примитивах (см. гл. 4).

В пятом блоке результаты тестирования заносятся в базу данных (в файлы rezul-A11. txt, …).

7.2. Методика формирования изображений исходных данных задач с помощью текстовых программ на языке АВТОЛИСП

Программа, осуществляющая формирование изображения текста и рис. 7.2 исходных данных проверочной задачи, имеет следующий вид:

```
;====================== начало – tema1-zad1 ========================
; Программа предназначена для формирования изображения текста и рисунка
;исходных данных задачи tema1-zad1
(DEFUN tema1-zad1 ()
; Загрузка вспомогательных программ
  (load "D:/Testing/Help-programs.lsp")
  (sapka) ; команда загрузки среды черчения
  (command "Limits" "0,0" "295,245" "GRID" "ON" "GRID" "10" "SNAP" "ON" "SNAP"
"TYPE" "GRID" "SNAP" "10")
; Блок задания координат точек исходных данных (см. рис. 7.3)
  (Sl0) (Ramka) ; команда загрузки чертежа прототипа рабочего поля задачи
  (setq P_L (LIST 110 93)          P_L1 (LIST 110 207)       P_C (POLAR P_L (* PI 1.5) 3)
        P_C1 (POLAR P_L1 (/ PI 2) 3)
        P_T (LIST 95.5 87.5)       P_T1 (LIST 95.5 209)      P_T2 (POLAR P_T 0 6)
        P_T3 (POLAR P_T1 0 6)    P_T4 (LIST 362 280)       P_T5 (LIST 160 280)
```

```
        P_T6 (LIST 52.5 260)        P_T7 (LIST 293.7 235)     P_T8 (LIST 293.7 225)
        P_T9 (LIST 293.7 215.3)     P_T10 (LIST 293.7 205)    P_T11 (LIST 293.7 195.3)
        P_T12 (LIST 293.7 185)      P_T13 (LIST 293.7 175)
  ) ; Блок задания координат точек исходных данных на рабочем поле (см. рис. 7.3)
(setq P_T20 (list 318 155)  P_T21 (list 293 140)  P_T22 (list 293 130)  P_T23 (list 293 115)
       P_T24 (list 293 105)  P_T25 (list 293 90)   P_T26 (list 293 80)    P_T27 (list 293 65)
       P_T28 (list 293 55)    P_T29 (list 293 40)   P_T30 (list 293 30)    P_T31 (list 293 10)
       P_T32 (list 404 132)  P_T33 (list 404 107) P_T34 (list 404 82)    P_T35 (list 404 57)
       P_T36 (list 404 32)    P_T37 (list 404 8)
  )  ; Блок формирования изображения рисунка исходных данных
 (COMMAND
   (sl2)  "text" P_T "6" "0" "A"        "text" P_T1 "6" "0" "A"       "text" P_T2 "3.5" "0" "1"
          "text" P_T3 "3.5" "0" "2"
   (sl0)  "text" P_T4 "6" "0" "Задача 1"                  "text" P_T5 "8" "0" "Тема 2"
         "text" P_T6 "12" "0" "Изображение прямой на К.Ч."     "circle" P_C 3    "circle" P_C1 3
   (sl5) "text" P_T7 "6.5" "0" "Построить горизон-"  "text" P_T8 "6.5" "0" "таль h проходя
          щую,"
        "text" P_T9 "6.5" "0" "через точку A"        "text" P_T10 "6.5" "0" "имеющую наклон к"
        "text" P_T11 "6.5" "0" "плоскости проекций"    "text" P_T12 "6.5" "0" "П2 равный 45"
        "text" P_T13 "6.5" "0" "(градусов)."
  (sl0)   "LINE" P_L P_L1 ""   (sl4)
  ) ; Задание кода файла проверки
  (setq TEST_N "T2-z1")
)   (T2-z1) (COMMAND "ZOOM" "All")
);==================== конец – tema1-zad1 ============================
```

7.3. Оценка правильности графических построений с помощью использования функций доступа к примитивам

На рис. 7.4 представлена блок-схема программы, выполняющей проверку правильности построений студентом графических построений при решении задачи, представленной на рис. 7.2.

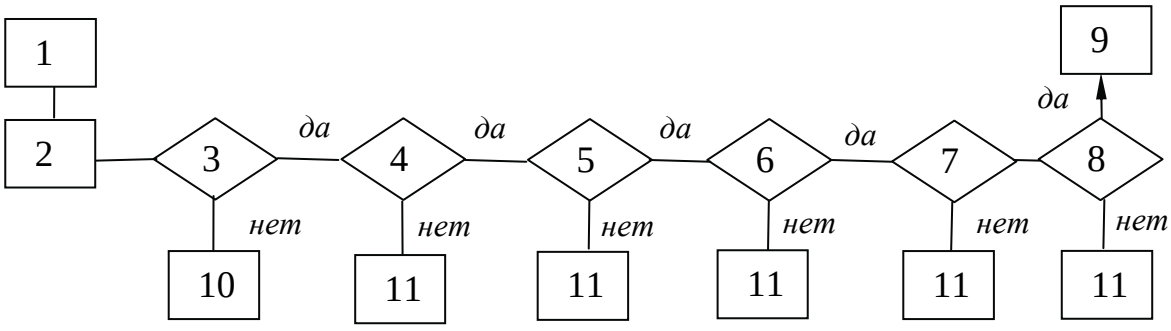


Рис. 7.4. Блок-схема оценки правильности графических построений задачи, представленной на рис. 7.2

На рис. 7.4 приняты следующие обозначения: 1 – создание наборов примитивов, построенных студентом; 2 – формирование списков, задающих координаты точек построенных примитивов; 3 – существует ли в наборе примитивов прямых горизонталь, проходящая через заданную точку A_2; 4 – существуют ли в наборе примитивов прямых прямая, проходящая через конечную точку горизонтальной проекции горизонтали и являющаяся линией проекционной связи; 5 – существует ли в наборе примитивов прямых прямая, проходящая через точку A_1 и имеющая угол наклона 45^{o} по отношению к оси Ox; 6 – существует ли текст обозначения фронтальной проекции точки, принадлежащей горизонтали; 7 – существует ли текст обозначения горизонтальной проекции точки, принадлежащей горизонтали; 8 – выполнено ли правильно текстовое обозначение индексов обозначений точек; 9 – задача решена верно с максимальной рейтинговой оценкой; 10 – задача не имеет правильного решения; 11 – задача имеет неполное решение, а графические построения имеют частичную рейтинговую оценку.

При выполнении текстовых обозначений проверяется положение базовой точки текста, которое должно быть в заданной зоне, а также индекс обозначений. Один из фрагментов текстовой программы, осуществляющей проверку графических построений задачи рис. 7.3, имеет следующий вид:

```
;====================== начало – prov-t1-z1 ======================
;программа оценки правильности графических построений
(DEFUN prov-t1-z1 ()
  (Clean_sl7)
  (SETQ LL (ssget "X" '((8 . "4")(0 . "LINE"))); создание набора примитивов из линий
     cc (ssget "X" '((8 . "4")(0 . "CIRCLE"))) ; создание набора примитивов из окружностей
     tt (ssget "W" '(100 10) '(130 40)) ; создание набора примитивов, входящих в рамку
     i 0   PN0 (list)   PK0 (list) ; создание нулевых списков
  ); Цикл, предназначенный для формирования списков, задающий координаты
;начальных и конечных точек прямых линий
  (IF (/= LL nil)
    (while (< i (sslength LL))
     (setq LL0 (entget(ssname LL i))
          LLN (cdr (reverse (cdr (reverse (assoc 10 LL0)))))
```

```
          LLK (cdr (reverse (cdr (reverse (assoc 11 LL0)))))
         PN0 (cons LLN PN0)  PK0 (cons LLK PK0)   i (+ i 1)
     )
   )
  )
  (setq i 0  PC0 (list)  PC0_R (list))
; Цикл, предназначенный для проверки правильности построений фронтальной и
; горизонтальной проекции прямых A2B2 и A1B1
  (IF (/= LL nil)
    (while (< i (sslength LL))
      (setq PN (nth i PN0)  PK (nth i PK0))
; Определение прямой, проходящей через точку A2 и являющейся горизонталью
  (IF  (cond
         ((and (equal PN A2) (= (nth 1 PK) (nth 1 A2)))
                  (setq FRON (cons 1 FRON)   PR_FRON_PN PN  PR_FRON_PK PK))
         ((and (equal PK A2) (= (nth 1 PN) (nth 1 A2)))
                  (setq FRON (cons 1 FRON)   PR_FRON_PN PN  PR_FRON_PK PK))
       )
       (setq FRON (cons 0 FRON))
      )
; Определение прямой, проходящей через точку A1 и имеющей наклон 45° к
; фронтальной плоскости проекций
   (IF (cond
         ((and (equal PN A1) (equal (ANGLE PN PK) 5.49779 0.01))
                   (setq GOR (cons 1 GOR)   PR_GOR_PN PN  PR_GOR_PK PK))
         ((and (equal PK A1) (equal (ANGLE PN PK) 2.35619 0.01))
                  (setq GOR (cons 1 GOR)   PR_GOR_PN PN  PR_GOR_PK PK))
       )
       (setq GOR (cons 0 GOR))
      )
    (setq i (+ i 1))
    )
  ) ;======================== конец – prov-t1-z1 ========================
```

7.4. Вопросы и задачи для самопроверки

1. Какая функция языка АВТОЛИСП используется для создания наборов примитивов из прямых при оценке правильности графических построений?

2. Какие функции необходимо использовать при создании списков, задающих координаты начальных и конечных точек прямых?

3. Назовите основные блоки программного обеспечения системы проверки графических построений.

4. Каков порядок составления программы, позволяющей создавать тексты и рисунки исходных данных проверочных задач?

5. Каким образом составляется программа оценки правильности графических построений?

Задача 7.1

Составьте текст программы, позволяющей создать списки, имеющие обозначение **PN1** и **PK1**, задающие координаты начальных и конечных точек отрезков прямых, если набор примитивов прямых имеет обозначение **LNL**.

Задача 7.2

Составьте запись в АВТОЛИСПЕ, позволяющую создать набор примитивов, имеющих обозначение *N2*, находящихся в рамке с координатами $x = 200$, $y = 150$ и $x = 120$, $y = 130$.

Ответы к задачам

Ответы к задачам главы 2

• **Задача 2.1** – (setq k1 (* l (cos u)) k2 (* l (sin u)) per (+ l k1 k2)) –

• **Задача 2.2** – (setq gp (sqrt (+ (expt h 2) (expt l 2)))) –

• **Задача 2.3** – (setq p1 (list 15 5) p2 (list 15 25) p3 (list 25 25)
p4 (list 25 15) p5 (list 30 15) p6 (list 30 5)
) –

• **Задача 2.4** – (setq p1 (list 50 50) p2 (polar p1 0 l) p3 (polar p2 1.57 h)) –

• **Задача 2.5** – (setq p1 (list 50 50) p2 (polar p1 1.57 h)
p3 (polar p2 0 l)) p4 (polar p1 0 l))
(if (and (≤ 0 (- h 14)) (≤ 0 (- l 14))) (setq p5 nil)
(setq p5 (inters p1 p3 p2 p4 nil))
) –

Ответы к задачам главы 3

- **Задача 3.1** – (command "LAYER" "M" "6" "C" "2" "6" "LTYPE"
  "center2" "6" "") –
- **Задача 3.2** – (SETVAR "DIMASZ" 10.)(SETVAR "DIMTXT" 7.)
  (SETVAR "DIMEXE" 4.)(SETVAR "DIMDLI" 1.) –
- **Задача 3.3** – (command "line" p1 p5 p4 p3 "" "arc" p1 p2 p3 "circle" p6 r) –
- **Задача 3.4** – (sl0)(command "pline" p1 p5 p7 p4 p1 ""
  "pline" p6 p2 p3 p8 p6 "")
  (sl1)(command "hatch" "jis_wood" "10" "0" p4 p8 "")
  (sl0)(command "line" p5 p6 "" "pline" p7 p8 "") –
- **Задача 3.5** – (sl0)(command "line" p1 p2 p3 p4 p1 "" "line" p6 p7 p8 p5 "")
  (sl1)(command "horiz" P4 P8 P9 "" "horiz" P1 P8 P10 ""
  "vertical" p8 p7 p11 (strcat "%%c" (rtos d1 2 0))
  "vertical" p3 p4 p12 (strcat "%%c" (rtos d2 2 0)) "exit") –

Ответы к задачам главы 4

- **Задача 4.1** – (foreach l1 (10 20 30 40) (sqrt l1)) –
- **Задача 4.2** – (setq LL1 (ssget “W” p1 p2)) –
- **Задача 4.3** – (setq ew1 entlast))
  (setq prim1 (entget ew1))
  (setq spisok (nth 10 prim1)) –
- **Задача 4.4** – (setq tt (strcat " - " t11 "-" t12 "-") tl (* 5 (strlen tt)
  p1 (list 80 100) p2 (polar p1 1.57 h) p3 (polar p2 0 tl)
  p4 (polar p1 0 tl) p5 (polar p1 0.78 3)
) (command "line" p1 p2 p3 p4 p1 "")
  (command "text" p5 "5" "0" tt) –
- **Задача 4.5** – (command "dim" "horiz" p1 p2 p3 (strcat "M" (rtos dr 2 0) "×"
  (rtos ss 2 0)) "exit")

- **Задача 5.1** – (command "vslide" " vtulka")
  (setq l1 (getreal "введите – l1:") l2 (getreal "введите – l2: ")
  h1 (getreal "введите – h1: ") h2 (getreal "введите – h2: ")
  p1 (getpoint "введите положение базовой точки: ")
) (command "redraw") –
- **Задача 5.2** – ***POP1
  [Обрабатываемый материал]
  [Сталь Х18Н9Т]Сталь Х18Н9Т
  [БРАЖ 9-4] БРАЖ 9-4
  [Сплав алюминия АМс6] Сплав алюминия АМс6
  ***POP2
  [Материал режущей части резца]
  [Сталь Р6М5] Сталь Р6М5
  [Твердый сплав, марки ВК6М] Твердый сплав, марки ВК6М
  [Сталь Р18] Сталь Р18
- **Задача 5.3** – ***POP10
  **LOAD
  [среда черчения]
  [линии чертежа] (load "D:/Данные/Студенты/
  sloi1.lsp ")(sloi)
  [системные размерные переменные№1]
  (load"D:/Данные/ Студенты / rasmer.lsp ")(ras)
- **Задача 5.4** – (setq ff (open "dan.txt" "r"))
  (setq b (atoi (read-line ff)) h (atoi (read-line ff)) l (atoi
  (read-line ff))
)
  (close ff)

• **Задача 6.1**

```
;====================== начало – mvy ==================
(defun mvy (ugol dlina smey)
;программа расчета элементов матриц Мк,к+1 при вращении вокруг
;оси oy
;ugol – обобщенная координата,
;dlina – длина звена,
; smey – смещение вдоль оси вращения.
(setq mm$ (list 0  (cos (dtr ugol))  0  (sin (dtr ugol))  (* dlina (sin (dtr ugol)))
                0                    1         0                    smey
          (- 0 (sin (dtr ugol))) 0 (cos (dtr ugol))  (* dlina (cos (dtr ugol)))
                0                    0         0                    1))
======================== конец – mvy ==================
```

• **Задача 6.2**

```
==================== начало – mvz ====================
 (defun mvz (ugol dlina smez)
;программа расчета элементов матриц Мк,к+1 при вращении вокруг
;оси oz
;ugol – обобщенная координата,
;dlina – длина звена,
;smez – смещение вдоль оси вращения.
(setq mm$ (list 0 (cos (dtr ugol))  (- 0 (sin (dtr ugol)))  0   (* dlina (cos (dtr ugol)))
                  (sin (dtr ugol))   (cos (dtr ugol))       0   (* dlina (sin (dtr ugol)))
                  0                  0                      1              smez
                  0                  0                      0                1))
);==================== конец – mvz ====================
```

• **Задача 6.3**

```
                – (setq
                      ui     (list    0   135   -25    -25 )
                      li     (list    0   100    90     80 )
                     smi     (list    0     0     0 )
                     kodi    (list    3     3     3  )
                  ) –
```

• **Задача 7.1**

```
– (setq i 0   PN1 (list)   PK1 (list)) ; создание нулевых списков
  (IF (/= LNL nil)
    (while (< i (sslength LNL))
      (setq LL0 (entget(ssname LNL i))
            LLN (cdr (reverse (cdr (reverse (assoc 10 LL0)))))
            LLK (cdr (reverse (cdr (reverse (assoc 11 LL0)))))
            PN1 (cons LLN PN0)  PK1 (cons LLK PK0)   i (+ i 1)
      )
    )
  ) –
```

• **Задача 7.2**

```
    – (setq  N2 (ssget "W" '(200 150) '(120 130))) –
```

Алфавитный указатель функций Автолисп

САПР *(CAD/CAE) – это система, позволяющая на базе ЭВМ автоматизировать определенные функции, выполняемые человеком с целью повышения темпов и качества проектирования.*

CAD *– это комплекс прикладных программ, обеспечивающих проектирование, черчение, трехмерное геометрическое моделирование деталей и сложных конструкций. Кроме этого применение CAD позволяет выполнять проектирование с элементами анимации (движения), визуализацию и управление базами данных и инженерными документами.*

CAE *– компьютерное технологическое проектирование.*

Компьютерная графика *– одна из подсистем САПР, необходимая для получения графической информации, её обработки и получения готовой документации в виде чертежей.*

Информационная модель детали – *это совокупность сведений, однозначно задающих форму детали и другие данные, необходимые для её изготовления.*

Графические системы *– это комплексы прикладных программ, имеющие специальные аппаратные средства, для выполнения плоской двумерной графики и трехмерного геометрического моделирования.*

Межпрограммный интерфейс *– это реализованная на компьютере возможность организации обмена данными между различными автономно функционирующими пакетами прикладных программ.*

Параметрические изображения *– это изображения, состоящие из совокупностей примитивов, заданных узловыми точками, координаты которых могут быть рассчитаны в соответствии с заданием определенных геометрических параметров. Программа, которая строит параметрические изображения, позволяет при каждом обращении к ней формировать новый чертеж, отличающийся от предыдущих чертежей, построенных этой программой, размерами и новыми элементами. Время получения параметрических изображений с помощью такой программы*

может быть в десятки раз меньше времени, необходимого для создания указанных изображений с помощью редактора Автокада.

Автолисп *– это небольшое подмножество языка Common Lisp, дополненного некоторыми функциями, отражающими специфику Автокада. Он позволяет выполнять не только расчеты, находясь в графическом редакторе, но и создавать подпрограммы, составляющие необходимую систему проектирования.*

Атомы *– это символы и числа, образующие простейшие объекты Автолиспа. Атомы это – 1, 7, 36, a1 и т. п.*

Список *– это набор разделенных пробелами атомов или списков, заключенных в круглые скобки.*

Строковые константы *– значения переменных, задающие тексты: они состоят из символов, слов, обозначений, предложений и т. п.*

Дескриптор файлов *– переменная, используемая для открытия файла с целью ввода или вывода информации.*

Факультативные аргументы *– это аргументы, которые могут как присутствовать, так и нет в списке аргументов функции.*

Системные переменные *– это переменные, используемые для настройки и управления параметрами графической системы, каждая из которых представляется определенным типом данных.*

Visual LISP *– это интегрированная среда разработки программ на языке программирования АВТОЛИСП в системе АВТОКАД, которая значительно облегчает процесс создания программы, её изменения, тестирования и отладки.*

Примитив *– это любой графический элемент, построенный с помощью реализации команд АВТОКАДА.*

Слайд *– это мгновенный снимок чертежа, который сохраняется в файле с расширением .sld . Слайд в отличие от чертежа невозможно редактировать.*

Механизм *– подвижно сочлененная совокупность твердых тел (называемых звеньями), предназначенная для преобразования движений.*

Манипулятор *– управляемое устройство робота и представляющее собой пространственный механизм с разомкнутой кинематической цепью, состоящей из нескольких звеньев и кинематических пар, обеспечивающих подвижность этих звеньев.*

Обобщенная координата механизма *– параметр, однозначно определяющий взаимное расположение звеньев, образующих кинематическую пару.*

Выходное звено (захват) *– конечное звено кинематической цепи манипулятора, движение которого обеспечивает выполнение технологических операций.*

Кинематическая пара *– подвижное соединение двух соседних звеньев.*

Конфигурация манипулятора *– положение кинематической цепи, определяемое обобщенными координатами* φ_1 , φ_2 , ... , φ_n *в неподвижном пространстве.*

Библиографический список

1. Чуприн, А.И. AutoCAD 2005. Platinum Edition / А.И. Чуприн, В.А. Чуприн. – М.: ООО «ДиаСофтЮП», 2005. – 1200 с.

2. Ляшков, А.А. Автолисп и его применение в начертательной геометрии и инженерной графике: учеб. пособие / А.А. Ляшков. – Омск: Изд-во ОмГТУ, 1994. 60 с.

3. Кудрявцев, Е.М. AutoLISP. Программирование AutoCAD14 / Е.М. Кудрявцев. – М.: «ДМК», 1999. – 368 с.

4. Гладков, С.А. Программирование на языке Автолисп в системе САПР Автокад / С.А. Гладков. – М.: ДИАЛОГ-МИФИ, 1991. – 98 с.

5. Проектирование и расчет подъемно-транспортных машин сельскохозяйственного назначения / М.Н. Ерохин и др. – М.: Колос, 1999. – 228 с.

6. Справочное руководство по черчению / В.Н. Богданов и др. – М.: Машиностроение, 1989. – 864 с.

7. Организация графических баз данных на основе пакета программ АВТОКАД: метод. указания / сост. Ф.Н. Притыкин. – Омск: Изд-во ОмПИ, 1991. – 32 с.

8. Притыкин, Ф. Н. Геометрическое моделирование при решении задач робототехники: учеб. пособие / Ф.Н. Притыкин. – Омск: Изд-во ОмГТУ, 1998. – 71 с.

9. Притыкин, Ф.Н. Ориентирование продольной оси схватоносителя при синтезе движений манипуляторов в организованных средах / Ф.Н. Притыкин // Мехатроника. – 2002. – № 1. – С. 16–20.

10. Притыкин, Ф.Н. Геометрическое моделирование процессов адаптивного управления движением мобильных и стационарных роботов в организованных средах: монография / Ф.Н. Притыкин. – Омск: Изд-во ОмГТУ, 2006. – 120 с.

11. Зенкевич, С.Л. Управление роботами. Основы управления манипуляционными робототехническими системами / С.Л. Зенкевич, А.С. Ющенко. – М.: МВТУ, 2000. – 400 с.

12. Притыкин, Ф.Н. Графическое представление телесного угла и окружающего пространства руки при реализации мгновенных состояний манипуляторов / Ф.Н. Притыкин // Проблемы машиностроения и надежности машин. – 2002. – №3. – С. 93–101.

13. Притыкин, Ф.Н. Система тестирования знаний студентов по дисциплине начертательная геометрия / Ф.Н. Притыкин, А.И.Анищенко, Д.А. Машук // Матер. седьмой Всерос. науч.-техн. конф. «Теоретические и прикладные вопросы современных информационных технологий». Улан-Уде, 24–30 июля, 2006. – Ч. 2. – С. 364–367.

14. Притыкин, Ф.Н. Учебный словарь-справочник. Часть 1. Основные термины и функции алгоритмического языка программирования АВТОЛИСП для целей САПР в среде АВТОКАД: учеб. пособие / Ф.Н. Притыкин. – Омск: Изд-во РосЗИТЛП, 2006. – 73 с.

Учебное издание

Притыкин Федор Николаевич, д-р техн. наук, доцент

ПАРАМЕТРИЧЕСКИЕ ИЗОБРАЖЕНИЯ ОБЪЕКТОВ ПРОЕКТИРОВАНИЯ НА ОСНОВЕ ИСПОЛЬЗОВАНИЯ ЯЗЫКА АВТОЛИСП В СРЕДЕ АВТОКАД

Учебное пособие

Редактор Т.А. Москвитина

Компьютерная верстка, дизайн обложки – Е.С. Соколов

ИД № 06039 от 12.10.2001 г.

Сводный темплан 2008 г.
Подписано в печать 11.02.08. Формат 60х84 $^{1}/_{16}$. Бумага офсетная.
Отпечатано на дупликаторе. Усл. печ. л. 7,0. Уч.-изд. л. 7,0.
Тираж экз. Заказ .

Издательство ОмГТУ. 644050, г. Омск, пр-т Мира, 11
Типография ОмГТУ