版本说明

作者:wyl219

初版:完成于2020年12月

Autolisp介绍

AutoLisp,简称Alisp,是古老的编称语言lisp[1]的方言[2],是一门解释型[3]的面向过程[4]语言,其升级版为VisualLisp(简称Vlisp).

Lisp语言是一种以表为基础的语言,最大的特点就是大量成对的括号[5],另外,换行对Alisp语言是**无用**的,因此你可以把所有代码写在一行,也可以随意分行(但是不能在关键字[6]或字符串[7]、数字等中间换行).

其他的前言

本文希望能借用《笨办法学python》的讲述方式，让初学者能在最短的时间内写出能用的程序，不会讲述全部的编程基础，也不会讲太多深入的东西，尽量保持初学者的学习兴趣和乐趣。

本文预计将以26节的篇幅介绍Alisp常用的函数和规则，并介绍一些Vlisp的入门知识。希望初学者看完本问以后有能力和兴趣自己去探索更多更大的世界。

部分名词解释

在开始之前,我想解释部分常用的名词概念,以免在阅读的过程中出现困扰,其他不常用,或我认为对初学者不重要的名词,将通过脚注的方式解释.

- 函数、参数、返回值
    - 函数:执行某个步骤的程序或代码.
    - 参数:函数执行需要的**数据**
    - 返回值:我们需要函数反馈给我们的**数据**
    - 以数学的概念来解释,y=x+1就是一个函数,我们可以写做f(x).其中f是函数名,x是参数,而y就是返回值,当x固定时,y也是固定的.
    - 在编程中函数名是必须的,而参数和返回值不一定是必须的.例如`exit`(退出当前命令)就不需要参数和返回值.
- 表达式:能表示一个值、或者一个调用函数的完整代码
    - 例如一个数字就是一个表达式,因为他能表示一个值,`(exit)`也是一个表达式,因为他是调用`exit`函数的完整代码
- 代码块:一个或多个表达式组成的整体
    - 在大部分编程语言中,代码块都会有明显的分隔,例如在C语言中,**被一对大括号中包括**的被视为一个代码块,在python中,有**相同缩进**的被视为一个代码块
    - ,而在Alisp中并没有代码块的概念.代码块实际上是一个**函数**的多个**参数**,例如`defun`函数,实际上是一个需要**不定数量**参数的函数.但是引入代码块的概念以后,`defun`函数就是一个需要两个参数及一个代码块的函数.
    - 而`if`函数,则是需要最多三个参数的函数,如果某个分支需要多个表达式时,需要用`progn`函数包裹成**一个表达式**
- 循环体:在循环结构中,**每次循环都会执行的**代码块.

准备工作

1. AutoCAD:建议2010及以上版本
    - 不建议安装天正等辅助设计插件,因为天正采用了自定义对象,很多查询和修改的命令无法使用,新手如果不了解之间的区别,很容易踩进大坑.
    - 所以建议使用纯CAD学习.
2. 文本编辑器:建议用CAD自带的Vlisp IDE或笔记本(Notepad)或其他代替程序,初学不建议使用支持代码补全的编辑器,例如飞诗LISP编辑器.初学阶段踩的坑越多,将来Debug[8]用的时间越少.
3. 参考资料
    - lisp葵花宝典A
    - CAD帮助文件:建议安装高版本的CAD离线帮助,或CAD2008的离线帮助文件
    - DXF参考手册
    - AutoCAD命令与系统变量
    - 组码查看器2019.9.19（修复bug）.lsp
    - Vlisp的参考资料,用到的时候再列

ex1:hello world

编程语言的学习基本都是从hello World[9]开始的,其目的是为了了解这个语言的基础语法.

以下代码将在cad中定义一个命令名为ttt的命令,运行后将弹出警告框,警告框的内容为hello world.

```
(defun c:ttt (/)
    (alert "hello world")
)
```

将以上内容**输入**[10]到文本编辑器中,保存为ex1.lsp,将保存后的文件拖入到cad的绘图区[11],运行命令ttt,可以看到弹出了对话框.

代码分析:

第一行中的第一个开括号与第三行的闭括号成组出现,defun是Alisp自带的关键字[6],defun表示定义一个函数,lisp中函数的调用都是这样的形式:

> (函数名 [参数]) 中括号括起来的参数代表可选参数

而defun函数的完整形式可以看葵花宝典A中的介绍,这里不展开说

c:ttt 表示函数名为c:ttt,在Alisp中函数名以c:开头,表示这个函数是一个可以被cad直接使用的命令,命令名**不含c:**.

后面跟着的(/)是函数所需要的参数及局部变量,由于这个函数很简单,不存在任何参数及变量,因此可以用()代替或直接用nil代替.参数、变量及nil以后再解释.

第二行是这个命令的主体,用成对的括号扩起来的部分是一个函数,alert的作用是弹出一个对话框,对话框的内容是由alert函数的参数定义的,本例中的"hello world"就是alert函数的参数.

"hello world"是一个字符串[7],字符串就是用引号括住的表示文字的数据,字符串中不能有换行.

小试手

- 尝试修改警告框的文本

ex2:更多的输出

对于一个程序来说,输入和输出是程序与用户的交互渠道,由于输入部分比较多,这里先说输出.Alisp中的输出主要以输出(打印)到命令行为主

以下代码将在cad中定义一个命令名为ttt的命令,运行后将输出各种信息以展示不同的输出方式.

```
(defun c:ttt ( /  )
    (princ "这是princ的输出")
    (print "这是print的输出")
    (princ "这是带换行的princ的输出\n")
    (prin1 "这是prin1的输出\n")
    (princ)
)
```

和ex1一样,将以上内容**输入**[10]到文本编辑器中,保存为ex2.lsp,将保存后的文件拖入到cad的绘图区[11],运行命令ttt,可以看到命令行输出了信息,也可以按F2键打开单独的命令行窗口.

代码解释

第一行和最后一行和ex1相同,不多解释.

第二至五行是Alisp常用的三种输出函数,具体区别可以看软件输出和葵花宝典里的介绍,这里着重的解释一下\n的意义.

\n表示换行符(\r表示回车符,两者并不相同),这个在任何编程语言中都是相同意义的.这里的\(斜杠)表示转义符[12],用于表现一些无法用文字表现的内容如换行符等,或让某些字符原样输出.例如想要在字符串中原样输出 " 则可以用 **\"** 表示.

第六行的(princ)表示空输出.

小试手

- 该怎么原样输出\n而不是输出一个换行符呢?
- 如果没有第六行的(princ)会有什么效果呢?

ex3:变量、数学运算函数

变量是指**用于存储可变量的符号。**而常量是指不可变的量。

可以这样理解，一个变量就是一个盒子，盒子的名字就是变量的名字，你可以将除了盒子本身以外的任何东西放进盒子里，然后通过盒子的名字代指这个东西。

对于Alisp来说，变量可以存储任何东西，一个数字，一段文本，一段函数或者是一段可执行的代码。

对于数学运算，大多数编程语言都是采用模拟语言的运算符，而lisp则是采用函数的形式进行数学运算，例如自然语言及大多数编程语言表达1加1时写做 1 + 1 ，而lisp中写做`（+ 1 1）`。

以下代码将在cad中定义一个命令名为ttt的命令,对变量进行定义赋值和修改后打印出来.

```
(defun c:ttt ( /  )
    (setq a 1)
    (setq b a)
    (princ (strcat "a的值为:" (itoa a) "\n"))
    (princ (strcat "b的值为:" (itoa b) "\n"))
    (setq a (+ a 1))
    (princ (strcat "现在a的值为:" (itoa a) "\n"))
    (princ (strcat "现在b的值为:" (itoa b) "\n"))
    (princ)
)
```

加载方式以后将不再重复,可以看之前两节

代码解释

与之前相同的部分不再重复.

第二行表示给变量`a`赋值1,对于Alisp,变量是不需要定义的,一个未赋值的变量,其初始值为`nil`,即**空值**.setq函数需要两个或更多**成对**的参数,成对的参数中首个函数是要赋值的变量,末一个参数是要被赋于的值.

第三行表示给变量b赋值为**变量a的值**,注意,是变量a的**值**而非**变量a本身**.对于这一点,本节后面会有更多说明.

第四行及第五行打印出a和b的值,在这里涉及到了函数的嵌套和两个新的函数.

- 函数的嵌套
    - 对于Alisp,允许将一个函数表达式作为另一个函数表达式的一部分,外层函数将接收到内层函数的返回值[^14]作为参数.
    - 对于嵌套函数,从**内**开始执行,对于同层函数,从**左**开始执行.
    - 对于嵌套函数,如果函数中有赋值(setq)函数,执行后**变量的值已经被改变**.对此,本节文末有一个参考程序.
- `itoa`函数的作用是把整数转换成字符串,这个涉及到后面要介绍的数据类型.由于数据类型中的表(list)是很重要的一部分,而且Alisp是一个弱类型[^15]的语言,平时对数据类型接触的并不多,这里不多介绍.
- `strcat`函数的作用是连接多个字符串,`strcat`可以接收**不定数量**的参数 ,但是每个参数均**必须**为字符串,因此在示例中用itoa函数将整数转换成字符串.

从第四行及第五行可以看到,此时变量a与b中存储的数据均为1.

第六行,将变量a的值加1后重新赋值给变量a.

第七行及第八行打印出现在变量a和b的值,可以看到变量a的值变成了2,而变量b的值**依旧**是1.以此说明,第三行中给变量b赋值的是**变量a的值**,而非**变量a本身**.

变量的简单介绍

Alisp中的变量要求很简单,可以由数字、字母、符号组成,甚至可以用**中文**.唯一的要求就是不区分大小写.即变量A与变量a是同一个变量.

虽然Alisp允许,但是依旧要避免与关键字[6]重名.

可以在cad的命令行中输入`!变量名` 例如`!a`来查看变量a中保存的值.

下一节将会介绍更多的变量命名规则以及局部变量和全局变量.

常用的数学运算函数

常用的数字运算函数有:

- + 加
- - 减
- * 乘
- / 除
- expt 乘方
- sqrt 开方
- abs 取绝对值
- rem 取余
- exp 自然反对数
- cos sin tan 等三角函数
- max min 取最大或最小项

除此以外,还有个跟数学运算有关的函数,fix,其作用是对实数取整(舍弃小数部分),但是其本质上是一个**数据类型转换函数**,是将实数转换为了整数.

注意,Alisp未内置四舍五入函数.

另外,对于常用的加1减1操作,还有两个函数即 **1+** 和 **1-** 函数.

再有,cad内置了变量 `pi` ,里面存储了圆周率π的值可以直接调用.

还有很重要的一点,对于Alisp,数学运算函数的返回值的精度与给定参数的精度中**最高**的那个相同.而实数精度高于整数.对此可以自行测试一下以下代码:

`(/ 3 2)`

`(/ 3 2.0)`

测试方式就是直接把括号及括号内的部分直接输入在cad的命令行中回车.

小试手

- itoa这个函数名实际是 int to ascii 的缩写,int代表整量,ascii代表字符,real是实数.那么你能猜到实数转换成字符串的函数会是什么么?
- 同样的,你能猜到字符串转整数的函数名么?
- 我相信你猜不到字符串转实数,整数转实数的函数,因为在这里实数被称为float(浮点数),猜不到的话查查葵花宝典吧.
- 可以把函数本身赋值和任意**符号**给一个变量,用于分支语句中执行不同的流程,对此在后面的**数据类型**和**分支语句**中有更多的介绍
- 关于嵌套函数中使用 `setq` 函数和函数的执行顺序的示例代码

```
(defun c:ttt ( / )
    (setq a 1)
    (princ (strcat "刚开始a的值" (itoa a)
                   "\n修改a的值" (itoa (setq a 2))
                   "\n现在a的值" (itoa a)))
    (princ)
)
(princ)
```

ex4:注释、自定义函数与更好的编码习惯

该节文字较多,希望能有耐心看下去

有一句话是这么说的:"**注释和空行也是程序的一部分**".完善,至少是必要的注释是一个优秀的代码必不可少的.不仅是方便别人学习,也方便自己理解代码.不然你会对自己两个月前的作品有种陌生感.

在本文以后的内容中,将把大部分的代码解释工作放在代码的注释中.

以下代码将在cad中定义一个命令名ttt的**命令**和一个函数名为add1的**函数**,add1函数将接受一个值,并返回这个值加1以后的值,而命令ttt则调用了这个函数.

本例主要用于体现注释的用法.以及函数的定义、如何返回值以及如何调用自定义函数.

```
;|
这里是多行注释.
;多行注释中可以嵌套单行注释
|;
(defun c:ttt ( / a )
    (setq a 2); 这里是单行注释,如一行中出现了字符串外的分号,那么该行之后的都被视为注释,
(princ "1")注释中的这个函数并不会被执行
    (princ (itoa (add1 a)));| 多行注释也可以写到同一行内,不过这实际是因为上面说的遇到分号
以后剩余部分均做为注释 |;
    (princ)
)

;这里是多个单行注释组成的多行注释
;通常会在函数或命令定义处注释说明函数的命令和参数要求以及返回值,例如:

;说明:接收一个数字,加一后返回.
;参数:i 整数或实数
;返回值: i+1后的值,精度与传入值有关
(defun add1 (i /)
    (+ i 1);这里就是返回值
)
```

与之前不同,这次代码中注释的比例比较多,练习的时候可以忽略注释部分

注释

注释分为单行注释和多行注释.单行注释以`;`开头,多行注释以`;|`开头,以`|;`结尾,其中的|一般在键盘回车键上面.

单行注释用处比较多,个人习惯用于注释变量、函数、参数、闭括号对应的函数以及分段.

多行注释通常用于函数和命令的说明.

当需要多行的经常调整行数的注释时,采用多行注释的形式比单行注释更方便,但是,在IDE[^16]中,单行注释或许会更方便

好的注释应该是在有必要的位置,本文剩下的部分,会介绍部分比较好用的注释方式.

函数的自定义

在ex1中介绍过命令的定义,当时就说过,**命令的定义其实就是函数的定义**,只是命令名的开头增加了`c:`的标示.

在ex1中提到defun的第二个参数`( / )`里面是存放了函数所需参数及局部变量,在不包括注释的第一行中,defun的第二个参数展示了**局部变量**的用法.

自定义函数的调用与内置函数相同,格式依旧为`(变量名 [参数])`.

函数的返回值就是**函数最后执行的那个函数的返回值**.各内置函数的返回值可以看葵花宝典中该函数的介绍,这也是为什么要在自定义函数中**用注释说明返回值**的原因.

注意:当自定义函数需要提供返回值,但是在最后执行的那一步函数**没有返回值或不是你需要的返回值时**,可以在自定义函数的最后加上需要返回值所存在的**变量名**.例如:

```

```

变量的作用域

在Alisp中,除了函数的参数以及部分函数中的临时变量(例如`foreach`和`lambda`函数),其余变量均默认为全局变量.

- 全局变量
  - 全局变量的作用是当前CAD打开的当前图纸下
  - 当两个函数或命令使用同一个全局变量时,两个函数对变量的修改会互相影响.
  - 当图纸关闭再重新打开后,全局变量会消失.
- 局部变量
  - 局部变量的作用域是当前函数或命令的执行过程
  - 当当前函数执行完成后,局部变量会被删除(或者说值被置为`nil`)
  - 如果局部变量的变量名与全局变量相同时,局部变量的修改不会影响全局变量.
  - 当**函数**嵌套在**另一个函数**里时,外层函数的局部变量的作用域是**覆盖内层函数**的
    - 例如在外层函数声明变量a=1为局部变量,还有一个全局变量a=2,那么内层函数以全局变量的形式调用变量a时,此时的**a的值为1**,是**外层函数的局部变量**.而非全局变量.
    - 但是当内层函数也把该变量声明成**局部变量**时,调用的变量a是作用域为内层函数的局部变量

可以尝试一下下面的示例代码.

```
(defun c:ttt (/)
    (setq a 1  ;新建两个变量并赋值,由于命令定义时未声明局部变量,因此是全局变量
               b 2  ;这里用到的setq函数的多变量同时赋值,注意参数应该是成对出现的
    )
    (princ (strcat "全局变量a此时的值是:" (itoa a) "\n"))
    (princ (strcat "全局变量b此时的值是:" (itoa b) "\n"))
    (princ "进入t2函数\n")
    (t2);该函数无参数
    (princ "返回ttt命令\n")
    (princ (strcat "全局变量a此时的值是:" (itoa a) "\n"))
    (princ (strcat "全局变量b此时的值是:" (itoa b) "\n"))
    (princ)
)

(defun t2 ( / b)
    ;这里定义了一个不需要参数的函数,并把函数中使用的变量b声明为局部变量
    (setq a 5
               b 6
```

```
    );这里对变量a和b赋值,注意,这里变量b是局部变量
    (princ (strcat "全局变量a此时的值是:" (itoa a) "\n"))
    (princ (strcat "局部变量b此时的值是:" (itoa b) "\n"))
)
```

命令ttt执行后输入如下:

> 命令: ttt
>
> 全局变量a此时的值是:1
>
> 全局变量b此时的值是:2
>
> 进入t2函数
>
> 全局变量a此时的值是:5
>
> 局部变量b此时的值是:6
>
> 返回ttt命令
>
> 全局变量a此时的值是:5
>
> 全局变量b此时的值是:2

可以看到,t2对局部变量b的修改未影响全局变量b,但是对全局变量的修改影响到了全局变量a.

注意,defun第二个参数`(/)`中,`/`前后如果有变量应该与变量之间有至少一个的空格.

注意:函数也有自己的作用域,局部函数的声名与局部变量类似,把函数名写到`/`后面就可以了

由于全局变量易被干扰,且非持久的特性,通常建议使用局部变量,当需要在多个函数或命令间共用数据时,将数据写入到**配置文件**.如果真想用全局变量,也应该给全局变量起一个不容易重名的变量名.

更好的编码习惯

好的编码习惯是很重要的,杂乱无章的代码,随意的变量名,会给你将来的修改带来极大的麻烦.

如果对此有兴趣,可以看<代码的整洁之道>这本书,里面讲解的很详细.本文简单的介绍一些编码习惯.

- 变量的命名
    - 好的变量名应该是一眼就能明白变量的作用,采用abc等简单的变量名,虽然写起来很方便,但是会在你阅读代码时带来麻烦
    - 不用担心变量名太长会影响编程的效率,代码补全是现在常用的IDE如飞诗,VSC等的标配.(cad自带的VLISP并不支持)
    - 虽然Alisp的变量名不区分大小写,但是并不影响你通过大小写分隔你的变量名
    - 变量的命名有几种常见的形式
        - 大小驼峰
            - 大驼峰:每个单词以首字母大写做为标示,例如变量名FirstName
            - 小驼峰:除首个单词首字母小写,其余单词首字母大写,例如变量名firstName
            - 无论大驼峰还是小驼峰,其目的均是通过大写字母分割词汇,使变量名更易读,其得名来自Perl.
        - 蛇形命名
            - 所有字母均小写,各单词之间用下划线`_`分隔,例如变量名first_name
            - 与驼峰命名法类似,通过下划线分割各单词.其得名来自python
        - 匈牙利命名

        - 开头字母用变量类型的缩写，其余部分用变量的英文或英文的缩写，要求单词第一个字母大写。例如变量SDwgName,S表示该变量存储的是字符串(String),DwgName表示变量存储的是dwg文件的文件名
        - 匈牙利命名法得名是因为其创造者是匈牙利人.该命名法比较古老,现在争议较多,主要是认为没必要将变量类型写在变量名中.
        - 但是Alisp的母语言Lisp也是一个同样古老的语言(1958,第二古老的高级编程语言,而第一台计算机是1946年),所以匈牙利命名法在lisp中还是有不少可取之处的
    - 上述几种命名形式只是约定俗成的,并非命名规则,完全可以将上述几种组合起来形成自己的风格.
    - 我个人习惯采用前后缀式的命名法
        - 前缀:变量类型
        - 主干:变量功能或储存的内容
        - 后缀:变量的特性
        - `前缀_主干_后缀`以下划线连接,主干中有多个单词时用大驼峰.
        - 例如一个用于临时储存块名列表的变量名为:lst_BlkName_tmp
    - 对于有些时候也可以简单的用大家都明白的缩写用来当作临时变量名,例如:
        - ss代表选择集
        - pt代表点
        - en代表图元名,ed代表图元信息表等
        - 循环中使用的临时记数变量用i做变量名,多层嵌套时往下顺延使用jk等
        - foreach和lambda中的临时变量用x做变量名,需要多个时顺延使用yz等
    - 通过不同的形式命名不同类型的变量,比如常量全大写,或用`*`包裹.
    - 总的来说,变量的命名应该容易识别和理解,不易有歧义,一个人的代码风格是慢慢形成的,找到适合自己的**固定的**代码风格
    - 如果想学习更好的变量命名方式,可以看看各位大佬的代码,或者参考别的语言的规范文本,例如python的PEP8
- 函数的命名
    - 函数的命名与变量类似.函数也有作用域,不同的是变量大多是局部变量,自己注意不要用重名就可以,而函数的作用域通常是全局的,因此需要考虑避免重名.
    - 常见的做法是增加个人标识的前缀,例如我的函数命名均是以`wy1:`开头,Lee-mac的函数大多以`LM:`开头,但是注意不要以`c:`开头,因为这是cad命令(command)的标示.
    - 函数名的主体应该表现出函数的作用,例如获取用户输入整数的函数名为getint.
- 命令的命名
    - 与函数、变量相同,不同的是命令的作用域都是全局的,因此易全不易简,然后通过命令别名起一个简化的名字,避免冲突.
    - 另外也可以像我一样,不需要常驻的命令都起名成ttt(懒的想名字),然后利用本文后面`命令别名`章节介绍的临时加载.
- 缩进与空行
    - 缩进指的是行前的空格
        - 对于同一层级的函数缩近保持相同
        - 对于嵌套的函数,内层的缩进比外层的多一级
        - 通常缩进为四个空格或者一个制表符(tab键)
        - 对于一个函数中多个参数,如果长度太长或需要增加注释等原因,各参数之间可以增加换行,但是每个参数的缩进应该大于该函数的缩进
        - 对于闭括号,其缩进应该与其相对应的开括号相同
    - 空行通常用来分隔两个代码块,主要用来方便快速定位.个人习惯如下:
        - 多个defun函数或命令之间不小于一个空行
        - 多条函数之间无空行

        - 当多个函数完成一个阶段步骤时,空一行
- 拆分函数
    - 不要把一个函数写的太大太复杂,这并不是好的习惯.
    - 通常如果一个函数超过40行,就可以考虑把这个函数拆分成多个小函数
    - 小函数易维护,也易复用.越大的函数越难以重复利用
    - 把复杂的循环体写成函数,关于循环的概念,会在后面介绍

小思考

- 为什么本节示例代码中ttt命令最后有一个空的princ函数而t2函数中没有呢?

ex5:简单的输入

前面ex2介绍了常用的输出,与输出相对应的就是输入,这里只是列举一些常用的输入函数,会在后面使用到的时候简单的介绍.如果有兴趣可以看葵花宝典.

- 获取用户的输入
    - getint 获取用户输入的**整数**
    - getreal 获取用户输入的**实数**
    - getstring 获取用户输入的**字符串**
    - getkWord 获取用户输入的**关键字**,需要与 `initget` 同时使用
    - getpoint 获取用户**选择**或**输入**的**点**
    - getdist 获取用户**量取**或**输入**的**距离**
    - getangke 获取用户**量取**或**输入**的**角度**,但是返回值是**弧度**
- 获取用户的选择
    - ssget 获取用户选择的**选择集**,该函数参数比较复杂,建议使用前看一下葵花宝典
    - entsel 获取用户选择的**图元名**,该函数的返回值是由**图元名**和**拾取点**组成的**点对**
    - nentsel 获取用户选择的**图元名**,返回值与entsel相同,不同的是,当用户点选的是一个**块里的对象**时,`entsel` 返回的是 **块** 的图元名,而 `nentsel` 返回的是 **块里对象** 的图元名

另外Alisp还有一个特殊的**常量** `pause`,其作用是暂停当前 `command` 函数,等待用户自行输入,这个将再后面的调用cad自带命令那一节介绍.

ex6:获取图元的属性

cad二开的目的就是操作图纸中的图元,因此获取图元的属性,是很重要的一部分.

以下代码将在cad中定义一个命令名ttt的命令,当你选取一个**直线**后在命令行输出**直线**的长度.

该示例的加载方式与其他相同,但是在运行前先需要在cad中绘制一条直线,对于多段线,比较复杂,这里先不涉及.

这次的示例会融合之前介绍过的函数.主要的代码解释会放在代码的注释中.

```
(defun c:ttt ( / en ed start end l )
    (setq en (car (entsel "选择一个直线:")));引导用户选择一个对象,car函数用于获取entsel函数返回值中的图元名
    (setq ed (entget en));获取图元的信息表
    (setq start (cdr (assoc 10 ed)));获取直线的起点
    (setq end (cdr (assoc 11 ed)));获取直线的终点
    (setq l (distance start end));计算起点到终点的距离,即直线的长度
    (princ (strcat "\n直线的长度为" (rtos l) "\n"));输出直线的长度
    (princ)
)
```

本例中出现了几个新的函数:

- `car` `cdr` `assoc`
  - 这几个函数都是用来从**表**中获取内容的函数,为了方便组织,这里不多介绍,放在后面的**表**的部分一起介绍
- `entget`
  - 该函数用于获取图元的信息表,接受的参数是**图元名**,返回值为**图元信息表**,类似下面的形式(节选)

    > ((-1 . <图元名: 7ffffb0d0b0>) (0 . "LINE") (8 . "0") (10 -627.573 4043.27 0.0) (11 4326.52 4043.27 0.0))

  - 图元信息表是一个**嵌套列表**(list),最外层是一个列表,第二层是由**组码**和**数据**组成的**列表**或**点对**
    - **点对**是一种**特殊的列表**,其区别是**列表**中可以有多个**元素**组成,**点对**只有两个**元素**组成,且点对两个元素之间有一个**点**
    - 这里的新概念比较多,了解一下就好,更详细的介绍将在**表与原子**章节介绍.
  - **组码**也叫dxf,是对象信息的索引值,不同的对象有不同的组码,具体可以查看<dxf参考手册>
  - 相同的组码储存的属性通常是相同的,以上面列出来的部分为例,各组码的作用如下:
    - -1 图元名
    - 0 图元类型
    - 8 图层
    - 10 直线起点坐标,对于其他对象,例如文字也有组码10和组码11,但是作用不太一样
    - 11 直线终点坐标
- `distance`
  - 计算两个点之间的距离,接收两个**点坐标**为参数,返回值是**实数**形式的两个点之间的距离.
- `rtos`
  - 将实数转换为字符串,至少接收一个**实数**作为参数(还有其他可选参数,详见葵花宝典),返回实数的字符串形式,当传入的参数是**整数**时会报错

小试手

- 你能写一个获取圆的**直径**的命令么?很简单的,参照在上面的示例代码修改一下就好了.提示一下,圆形的信息表中并没有存储圆的直径,需要从半径计算出来.

ex7:图元的操作

图元的修改包括新建、删除和修改图元,新建和修改图元需要操作图元信息表,现在还没介绍到表,所以不展开介绍.

- 新建图元
  - 使用 `entmake` 或 `entmakex` 函数,将图元信息表代表的图元添加到cad中,entmake函数只需要图元的**必要信息表**,对于图元的必要信息表可以参考这篇贴子
- 删除图元
  - 使用 `entdel` 函数即可,该函数接收一个**图元名**(即entsel获取到的或信息表中组码-1对应的图元名),**当该图元未被删除时,该函数将删除图元,当该图元已被删除时,将恢复该图元.**\
- 编辑图元
  - 使用 `entmod` 函数,该函数接收一个**图元信息表**,并更新其对应的图元,
  - entmod函数用到的图元信息表一般不需要自己重新构建,而是通过修改原有信息表实现的.
  - **注意!**不要试图用entmod函数修改**天正自定义对象**,这会使你的cad**致命错误**

ex8:第一个实用命令

虽然现在介绍的东西还不多,但是已经足够拼出来一个有实际作用的小程序了.

以下代码将在cad中定义一个命令名ttt的命令,当你选取一个**单行文本**后,将在该**单行文本**后增加**米**做为后缀.

该示例的加载方式与其他相同,但是在运行前先需要在cad中添加一个**单行文本**(CAD命令为text),对于**多行文本**,比较复杂,这里先不涉及.

由于选择集的处理需要使用后面介绍到的循环,这里为了能快速出成果,只实现单选功能.

```
(defun c:ttt ( / en ed start end str1 l )
    (setq en (car (entsel "选择一个文本对象:")));引导用户选择一个对象
    (setq ed (entget en));获取图元的信息表
    (setq str1 (cdr (assoc 1 ed)));获取单行文字的内容
    (setq str1 (strcat str1 "米"));文字内容后加上"米"

    (setq ed1 (subst (cons 1 str1) (assoc 1 ed) ed));替换信息表中组码1的内容
    (entmod ed1) ;修改图元

    (princ)
)
```

这次的示例代码中出现了两个新的函数,与ex6相同,这两个函数都是**表**相关的函数,这里只需要了解一下

- `subst`
    - 从一个表(第三个参数)中找出一项(第二个参数),并用新的项(第一个参数)替换它
- `cons`
    - 常用的构造表函数

小试手

- 怎么修改代码,可以同时增加前缀 **"长度"** 和后缀 **"米"** 呢?

ex9:表与原子

Alisp的母语言Lisp是一个**表处理语言**,从名字也看的出来,Lisp就是将列表(list)的最后一个t改为了p.在lisp中表是其极为重要的一项,内容过多,篇幅过长,为了不影响初学者的学习积极性,将表的介绍延后了,但是从ex6和ex8可以看出来,Alisp的实用编程离不开表的相关知识.接下来会有三到四节内容与表直接相关,而且会有较少的示例代码和大量的概念和函数,考验毅力的时候到了.

什么是原子(atom)

Alisp代码中除了表,其他的都是原子.

原子即不可再用表的结构分割的东西,例如整数(int),实数(real),字符串(string)等

什么是表

Alisp代码中除了原子,其他的都是表.

像函数、命令都是**广义**上的表,而**狭义**上的表则是由若干个**可以重复**的**元素**组成的**有序**的列表.

在实际的Alisp编程中,更多的使用的是表的**狭义**的定义.本文其他地方出现的**表**纯指的其**狭义**的定义.

什么是元素

元素就是组成表的各部分,可以是表或者原子.

什么是点对

在ex6中提到过**点对**的概念,这里重复一下:

> **点对**是一种**特殊的列表**,其区别是**列表**中可以有多个**元素**组成,**点对**只有两个**元素**组成,且点对两个元素之间有一个**点**

点对也是表,所以也是**有序**的,但是由于其只有两个元素,所以系统开销更小,也能使用部分表操作函数,但是需要注意的是,某些函数对表和点对的**处理结果**不同.

注意(1 2)和(1 . 2)是不同的两个表.

空表

空表也是一种特殊的表,空表中没有任何元素,但是可以通过增加新的元素扩充.
注意:虽然`nil`代表的是**空**,但是`(nil)` **不是空表**

如何确定是不是原子

可以用`atom`函数,该函数接收一个参数,如果其**不是表**,则返回`t`,否则返回`nil`.

小试手

- 在cad命令行中执行下面的代码,了解一下什么是表,什么不是表

```
(atom 1)
(atom 1.2)
(atom "a")
(atom '())
(atom '(1 2))
(atom '(1 . 2))
```

- 上述代码中的`'`其实是函数quote的一种简写,该函数的作用是返回表达式本身,而非其返回值.这在Alisp中是很常用的函数(否则也不会有简写形式了)

ex10:构造一个表

在**Alisp**中常用的构造(创建)表有以下方式:

- `list`函数
  - list函数接收**不定数量**的参数,将其按顺序构造成**表**返回,注意,当不传入参数时,将构造一个**空表**.
  - list可以接收任何类型的参数,包括表、原子、函数、变量或**变量本身**等,当传入函数或变量本身时,应用quote函数处理.
  - **注意**当传入的参数是变量时,构建出来的表中保存的是**变量的值**而非**变量本身**,例如执行以下代码后

```
(setq a 1)
(setq b (list a))
(setq a 2)
!b
```

    - 变量b中保存的值依旧是(1),而非(2)
    - 不明白!b的作用的朋友,可以翻翻前面的内容
- `cons`函数
    - 该函数有两个作用,一个是**扩充表**,一个是构造一个**点对**,该函数接收两个参数,两个参数均可以是表或原子.
    - 第一个参数是表或着原子并不影响该函数的功能,但是第二个参数影响该函数的功能.
    - 当第二个参数是**表**时,该函数将把第一个参数**扩充**第二个参数中,并做为首项
    - 当第二个参数是**原子**时,该函数将把第一个参数和第二个参数构造成一个点对.
    - 因此cons函数不能可以构造一个形式为`(表 . 原子)`的**点对**,但不能构造一个形式为`(原子 . 表)`的**点对**.
    - 由于某些函数对**表**和**点对**的处理结果不同,因此**除非你明确你现在要做什么,否则不建议使用cons函数**
- `quote`函数
    - 直接用`(quote (1 1))`来构造一个**表**或`(quote (1 . 1))`来构造一个**点对**,别忘记`'`这个用法.
    - 虽然通常可以把`(quote (1 1))`简写成`'(1 1)`,但是有的时候两者并不完全相同,通常使用`'`这个写法是没问题的

在**Vlisp**中也有几个构造表的函数,这里不多介绍,有兴趣的可以看看葵花宝典

小试手

- 用上面介绍的三个函数构造不同列表,了解其区别
- 上面说到`'(1 1)`能构造一个表,那如果表达式是一个嵌套表达式,会是什么效果呢?如果把`'`换成quote函数的形式,又会出现什么情况呢?
- `'(1 (1 . 1))`返回的是什么呢?
- 很多例外情况或者说奇淫巧技,了解它并不是为了能用在编程工作中或者拿来炫耀,而是为了**避免错误**

ex11:扩充和编辑表

扩充表使用下面两个函数:

- append 合并多个**表**
    - 该函数接收至少**不定数量**参数,每个参数都必须为**不为点对**的表
    - 如果传入一个表,将返回表本身,当不传入表,则返回`nil`
    - 如果传入多个表,将返回**按顺序合并**后的**表**
- cons
    - 这个函数上一节介绍过
    - 如果你希望用cons扩充一个表,那么必须确保第二个参数必须是一个表,哪怕是空表
    - **注意:**再次建议除非你知道你在做什么,否则不要随意使用`cons`函数.

Alisp中没有删除表中的元素的函数(**Vlisp**中有,可以在葵花宝典中搜索`remove`),如果希望用纯Alisp删除表中的某个元素,需要用后面介绍到的循环.

可以使用`reverse`函数倒置一个列表

- 该函数接收一个不为**点对**的表,并返回倒置后的表
- 该函数并**不会修改**传入的表.

可以用`subst`函数替换表中的一个元素

- ex8中使用了这个函数

- 该函数接收三个参数,分别是新项,旧项和要处理的**不为点对**的列表
- 与 `member` 和 `assoc` 函数相同,当有多个匹配项时,只会修改第一个匹配项
- 该函数并**不会修改**传入的表.

ex12:编辑表和提取表中的元素

Alisp中有大量的检索获取表中元素的函数,这里介绍几个常用的:

- car 返回表中第一个**元素**
  - 该函数接收一个参数,该参数必须是表.当该参数为空表时返回`nil`,否则返回该表的第一个元素
  - 无论传入的参数是表或者点对,该函数返回的均是一个**元素**
- cdr 返回表中去掉了第一个**元素**的剩余部分
  - 该函数接收一个参数,该参数必须是表,当该参数为空表时返回`nil`.
  - 当传入一个**点对**时.该函数返回的是一个**原子**(因为Alisp中不存在第二个元素是表的点对)
  - 当传入一个不为点对的**表**时,该函数返回的是一个**表**
  - 即`(cdr '(1 2))`返回的是`(2)`,`(cdr '(1 . 2))`返回的是`2`.
- last 返回表中最后一个**元素**
  - 该函数不能接受一个**点对**做为参数
- nth 返回表中指定**序列号**的**元素**
  - 该函数接收两个参数,第一个参数为**序列号**,第二个参数是一个**不为点对**的表
  - 在包括Alisp在内的大部分编程语言中,序列号是从0开始的,而不是从1开始
- member 检索表中是否有某个**元素**,如果有则返回该元素及之后所有的元素组成的**表**,否则返回`nil`
  - 该函数接收两个参数,第一个参数为要搜索的元素,第二个参数是一个表
  - 特别的是,当第二个参数是**点对**时,第一个参数是**点对**前一个元素时会返回**这个点对**,**否则会报错**.
  - 当作为第二个参数的表中有多个与第一个参数相同的元素,会返回第一个匹配到的元素以后的部分.
  - 该函数主要用与判断某个元素是否在表中.
- assoc 在**关联表**(由表或点对组成的表)中搜索一个元素(点对的第一个元素),找到则返回**关联表条目**,否则返回`nil`
  - 该函数会在关联表中搜索第一个元素与第一个参数相同的**表**或**点对**
  - 关联表中至少有一个点对或表,不可以有其他原子,关联表可以为**空表**,此时会返回nil
  - 与member相同,当关联表中有多个匹配项时,会返回第一个匹配项
  - 该函数经常用于从图元信息表中获取对应组码的数据.
  - **注意:**该函数返回的是一个**表**或**点对**,想要获取图元对应的属性,需要用`cdr`函数处理一下

另外Alisp中还有几个car与cdr组成的语法糖[^17],例如`cadr`,`caar`等,这里不多介绍,可以看葵花宝典,或在飞诗lispedit中尝试.

注意:再次建议除非你知道你在做什么,否则不要随意使用`cons`函数.

小试手

- 参考ex8中的示例代码,尝试修改对象的图层、颜色或其他属性,提示,在图元信息表中不一定存在颜色所对应的组码.多看看<dxf参考手册>

ex13:第二个实用命令

看了这么多的概念,是不是想码点代码换换脑子呢?本节将在ex6的示例代码上修改,让其更实用一点.

以下代码将在cad中定义一个命令名ttt的命令,当你选取一个**直线**后,计算出直线的长度,并在cad中获取一个点,在该点插入一个单行文字,以 **"长度:xxx米"** 的形式标注.

该示例的加载方式与准备工作与ex6相同.

```
(defun c:ttt ( / en ed start end l str1 lay pt)
    (setq en (car (entsel "选择一个直线:")));引导用户选择一个对象,car函数用于获取entsel函数返回值中的图元名
    (setq ed (entget en));获取图元的信息表
    (setq start (cdr (assoc 10 ed)));获取直线的起点
    (setq end (cdr (assoc 11 ed)));获取直线的终点
    (setq l (distance start end));计算起点到终点的距离,即直线的长度
    ; 以上内容与ex6相同

    (setq l (/ l 1000));一般绘图时单位均是毫米,这里标注的是米,换算单位
    (setq str1 (strcat "长度:" (rtos l) "米"));拼出来要标注的内容
    (setq pt (getpoint "选择插入点"));引导用户选择一个插入点
    (setq lay (assoc 8 ed)) ;获取直线所在的图层

    (entmake (list '(0 . "TEXT") ;对象类型
                        (cons 1 str1) ;文字内容
                        (cons 10 pt)  ;插入点
                        (cons 40 350) ;字高
                        lay ;图层与选择的直线相同
                    ))
    (princ)
)
```

这次没有引入新的函数

小试手

- 本例是按照常见的工民建施工图的标准设计的,如果是市政之类的总图,通常绘图单位是米,该怎么修改呢?如果你有做总图的需要,别忘记把文字高度修改成你需要的高度.
- 对于存放图层的变量lay,为什么获取时没有像其他数据那样用cdr处理过呢?如处理过,该怎么修改呢?
- 本例是把标注的文字的放在直线所在的图层上,如果想放在固定的别的图层该怎么修改呢?**提示**需要先在cad中新建该图层,新建的图层也可能用entmake,或者用command函数,当然,也可以手动提前在cad的图层管理器中新建出来.

ex14:逻辑判断

前面写了两个实用命令,虽然实用,但是很脆弱,例如ex13中,如果用户选择的不是直线而是一个文字,那么在程序运行过程中就会出错.

或者,如果我们希望让自己的命令更强大,例如选择的是直线则标注长度,如果是圆形则标注直径.

为了避免出错,或者让程序的通用性更强,就要用到逻辑判断和分支功能.

本节和之后的分支语句、循环都是比较重要而且比较考验思维能力的部分,要多查资料多练习,才能更好的掌握.

所谓的逻辑判断,就是判断某个条件是否是真,在Alisp中,除了`nil`,其他的都视为真,包括`0`.对于逻辑运算函数,当为真是返回的是`t`

与数学运算相同,Alisp中也有多个逻辑运算函数,简单列举如下,具体可以查阅葵花宝典.

- = 等于

- /= 不等于
  - 只有任意两个相邻的参数不同时才返回t,否则返回nil,详见葵花宝典
- > 大于
- < 小于
- >= 大于等于
- <= 小于等于

对于以上函数,都是从左向右对比的,当对比的两个数分别为整数和实数时,也可能相等.即`(= 1 1.0)`为真.

大于小于不用多说,明显是用于数字间比较大小的.而等于和不等于可用于其他对象,但是对于**表**可能会出现意料外的错误,这一点,将在稍后介绍.

注:虽然葵花宝典中说等于和不等于只能用于数字和字符串,但是实际也是可以用于表的,只是结果与常识不符

在计算相等时,还有两个函数

- eq 判断两个参数是否是**同一对象**
  - 即`(eq (list 1 2) (list 1 2))`为假,因为这两个列表虽然看起来一样,实际是两个不同的对象
  - 而以下代码为真,因为变量a和b指向同一对象:

    ```
    (setq a (list 1 2))
    (setq b a)
    (eq a b)
    ```

  - 虽然没有证据,但是在我使用中,`eq`函数和`=`函数未发现区别
- equal
  - 与`eq`但有以下两点区别
    - `(equal (list 1 2) (list 1 2))`为真,因为他们虽然不是同一个对象,但是值是相等的
    - `equal`能设定**允许误差值**,使得其应用面更广,例如`(equal 1 2 3)`为真,其第三个参数`3`就是允许误差值

另外还有**与或非**函数:

- and 与
  - 所有参数均为真时为真,否则为假
  - `(and t nil)`为假
- or 或
  - 所有参数均为假时为假,否则为真
  - `(or t nil)`为真
- not 非
  - 只允许一个参数,该参数为真时为假,否则为真
  - `(not nil)`为真

对于`and`和`or`还有一个特殊的应用,即在某些场合代替if函数,将在后面节 **分支语句** 中介绍.

对于字符串,有一个用于**模糊匹配**的函数`wcmatch`,具体介绍可以看一下葵花宝典中的介绍,该函数支持的pat可以视为**阉割版的正则表达式**,对于在Alisp中使用正则表达式,可以看看这篇帖子

另外一些能返回nil的函数也可用于逻辑判断,例如

- member

- assoc
- atom
- listp

ex15:数据类型和图元类型

前面ex3中提到过数据类型,这里稍微展开说一下.

数据类型

使用`type`函数即可看到数据类型

Alisp中有以下常见的数据类型:

- int 整数,无小数部分
  - `(type 1)`
- real 实数,有小数部分,小数部分可以等于0
  - `(type 1.0)`
- string 字符串,即文字,包括字符串形式的数字
  - `(type "a")`
  - `(type "1")`
- list 表,点对也是表,但是空表不是表,而且表达式是表
  - `(type '(1 1))`
  - `(type '(1 . 1))`
  - `(type '())` 返回`nil`
  - `(type '(list))`
- sym 符号,函数、变量本身等是符号
  - `(type 'list)`
  - `(type 'a)`
- nil 空,`nil`及`nil`**本身** 和**空表**的数据类型是`nil`
  - `(type nil)`
  - `(type 'nil)`
  - `(type '())`
- ename 图元名
  - `(type (car (entsel "选择一个对象")))`
- pickset 选择集
  - `(type (ssget))`

对于变量,type函数返回的是**变量的值**的数据类型,而不是**变量本身**的数据类型(sym 符号)

int、real、string三种数据类型是**基本数据类型**,三者之间可以互相转换.

- int to
  - real
    - float函数,int到real的转换不会损失精度
    - `(= 1 (float 1))` 返回`t`
  - string
    - itoa 函数
- real to
  - int

        - fix函数,该函数会舍实数的小数部分,因此real到int的转换**可能会损失精度**
            - `(= 1.0 (fix 1.0))` 返回 `t`
            - `(= 1.1 (fix 1.1))` 返回 `nil`
    - string
        - rtoa函数
- string to
    - int
        - atoi函数
    - real
        - atof函数

上述几个数据类型转换函数,对接收的参数数据类型都有要求,但是如果数据类型错误,结果却不尽相同,可以自行尝试一下

图元类型

图元类型保存在**图元信息表**中,对应组码固定为`0`

图元类型较多,就不再列举了,有兴趣的可以看一下<dxf参考手册>或者自己探索

图元类型和数据类型更多的用在分支语句中.用于针对不同的数据或图元做不同的处理

小试手

- 尝试一下三种基本类型之间的转换,刻意的传入错误的参数,看看有什么反应
- **注意:**数据类型是**符号**,符号**不区分大小写**,而图元类型是**字符串**,字符串**区分大小写**.即
    - `(= 'int (type 1))`与`(= 'INT (type 1))`均为真.
    - `(= "LINE" (cdr (assoc 0 (entget en))))`为真
    - `(= "line" (cdr (assoc 0 (entget en))))`为假.

ex15:分支语句

分支语句是代码逻辑中最重要的一项.

Alisp有两个分支函数:

- if 单条件双分支函数
    - if接受二到三个参数.其中第一个参数是能用于**逻辑判断的表达式**,当第一个参数为真是,执行第二个参数,否则执行第三个参数.
    - 第二和第三个函数必须是**一个表达式**,当需要多个表达式时,需要用`progn`函数括住.
    - 第二个参数**不能省略**
- cond 多条件多分支函数
    - 该函数可以接受多个**表**,每个表的第一项元素必须是能用于**逻辑判断的表达式**
    - 该函数会对每个参数中第一项元素**求值**,如果为真则执行**该参数剩余部分**,否则跳过剩余部分继续执行下一个参数的第一项元素
    - 一个参数如果被执行了,则剩余的参数不会被执行
    - 做为参数的表的第一项可以是`t`,如果这个参数不是最后一个参数,那么这个参数以后的参数都**不可能被执行**.

虽然if函数也能通过嵌套实现多条件多分支,但是cond函数更简单实用一些.

另外,前面提到过`and`函数和`or`函数某些时候也能用来代替`if`函数

原理是`and`函数当找到第一个`nil`时,剩余部分不执行,而`or`函数当找到第一个非`nil`时,剩余部分不执行.因此下面的**单分支**的`if`函数可以用`and`或`or`函数代替:

```
(setq a t)
(if a (print "真"))
```

```
(setq a t)
(and a (print "真"))
```

这种写法更简洁也更高效,熟悉以后也比较易懂,但是不建议新手使用.

两个示例代码

以下代码将扩充自ex13,在cad中定义一个命令名ttt的命令,当你选取一个**图元**后,在cad中获取一个点,如果该图元为直线则标注长度,如果是圆形则标注直径,否则不做任何操作.

第一个代码使用**嵌套的if函数**.

```
(defun c:ttt ( / en ed start end l d str1 lay pt type_en)
    (setq en (car (entsel "选择一个直线或圆:")));引导用户选择一个对象,car函数用于获取entsel函数返回值中的图元名
    (setq ed (entget en));获取图元的信息表
    (setq type_en (cdr (assoc 0 ed)));图元类型

    (if (= "LINE" type_en);如果图元类型是直线
        (progn
            (setq start (cdr (assoc 10 ed)));获取直线的起点
            (setq end (cdr (assoc 11 ed)));获取直线的终点
            (setq l (distance start end));计算起点到终点的距离,即直线的长度
            (setq l (/ l 1000));一般绘图时单位均是毫米,这里标注的是米,换算单位
            (setq str1 (strcat "长度:" (rtos l) "米"));拼出来要标注的内容
        )
        (if (= "CIRCLE" type_en);如果图元类型是圆
            (progn
                (setq d (* 2 (cdr (assoc 40 ed ))))
                (setq str1 (strcat "直径:" (rtos d) "毫米"));拼出来要标注的内容
            )
            (progn ;如果以上都不是
                (princ "\n对象选择错误.")
                (setq str1 nil);把str变量设空,当做一个记号
            )
        );end if 圆
    );end if 直线


    (if str1 ;如果str不为空,执行以下内容
        (progn
            (setq pt (getpoint "选择插入点"));引导用户选择一个插入点
            (setq lay (assoc 8 ed)) ;获取对象所在的图层

            (entmake (list '(0 . "TEXT") ;对象类型
                           (cons 1 str1) ;文字内容
                           (cons 10 pt)  ;插入点
                           (cons 40 350) ;字高
                           lay ;图层与选择的直线相同
                     )
```

```
            )
        )
    )
    (princ)
)
```

第二个代码使用**cond代替嵌套的if函数**.

```
(defun c:ttt ( / en ed start end l d str1 lay pt type_en)
    (setq en (car (entsel "选择一个直线或圆:"))) ;引导用户选择一个对象,car函数用于获取
entsel函数返回值中的图元名
    (setq ed (entget en)) ;获取图元的信息表
    (setq type_en (cdr (assoc 0 ed))) ;图元类型

    (setq str1 (cond ;把赋值步骤放在外面
                            ((= "LINE" type_en) ;如果图元类型是直线
                                (setq start (cdr (assoc 10 ed))) ;获取直线的起点
                                (setq end (cdr (assoc 11 ed))) ;获取直线的终点
                                (setq l (distance start end)) ;计算起点到终点的距离,
即直线的长度
                                (setq l (/ l 1000)) ;一般绘图时单位均是毫米,这里标注的
是米,换算单位
                                (strcat "长度:" (rtos l) "米") ;拼出来要标注的内容)
                            )
                            ((= "CIRCLE" type_en) ;如果图元类型是圆
                                (setq d (* 2 (cdr (assoc 40 ed ))))
                                (strcat "直径:" (rtos d) "毫米")
                            )
                            (t  ;如果都不是
                                (princ "\n对象选择错误.")
                                (exit) ;退出程序
                            )
                        ) ;end cond
    ) ;end steq

    (setq pt (getpoint "选择插入点")) ;引导用户选择一个插入点
    (setq lay (assoc 8 ed)) ;获取对象所在的图层

    (entmake (list '(0 . "TEXT") ;对象类型
                        (cons 1 str1) ;文字内容
                        (cons 10 pt)  ;插入点
                        (cons 40 350) ;字高
                        lay ;图层与选择的直线相同
                    )
    )
    (princ)
)
```

这里引入了一个新函数,`exit`,该函数会结束当前的命令,并执行`*error*`函数,程序员可以重新定义`*error*`函数,以**恢复部分设置**.

本次的示例未修改设置,所以没涉及到`*error*`函数.会在后面更多的介绍一下.

ex16:简单的循环

循环是解放双手的利器,例如ex15中的示例程序,只能一个一个选择对象标注,而利用循环以后,就能批量选择,批量标注.

先给出一个**循环体**的概念,循环体就是每次循环都要执行的部分.

Alisp提供了三个不同的循环和遍历[^19]函数:

- `while` 根据逻辑判断确定何时结束
    - 该函数的**第一个表达式**是**可以用于逻辑判断的表达式**,当该表达式为真,则执行剩余的内容.循环体执行完成后再次执行上述步骤,直至**该表达式的值为假**.
    - **注意:**每次循环该表达式也会执行一次,所以该表达式也是**循环体**的一部分.
    - **注意:**如果该表达式**恒定为真**,那么while循环不会结束,会导致cad卡死
    - **注意:**如果该表达式**恒定为假或第一次执行时就为假**,那么剩下的部分不会被执行
    - while循环常用于**不能提前确定循环次数**的循环,例如遍历**符号表**[^18],如果能提前确定循环次数,应该优先考虑 `repeat` 循环
- `repeat` 指定次数的循环
    - 该函数的**第一个表达式**是**值为*整数*的表达式**,该表达式仅**执行一次**,剩余部分将循环执行.
    - 该函数常用于遍历选择集,通过 `sslength` 获取选择集的长度,对于表的遍历,通常使用 `foreach` 函数.
- `foreach` 遍历列表循环
    - 该函数第一个参数是指定一个变量.该变量是**作用域**为该 `foreach` 函数的局部变量.
    - 第二个参数是要被遍历的**表**
    - 剩余部分是**循环体**
    - foreach函数能做的事情,repeat函数都能实现,不过foreach在遍历表时,省去了计算表长度和从表中挨个取值的步骤,更方便

注意: Alisp中没有**跳出循环**或**跳过循环体剩余部分**的功能.

示例代码

代码一:使用while循环,遍历所有图层,并打印图层名到命令行

```
(defun c:ttt ( / lay)
    (setq lay (tblnext "layer" t)); 获取符号表中的第一项
    (while lay
        (print (cdr (assoc 2 lay)))
        (setq lay (tblnext "layer"));获取符号表的下一项,当到最后一项后,返回nil
    )
    (princ)
)
```

这里引入了一个新函数,`tblnext`,该函数用于**遍历**符号表,函数比较简单,可以去葵花宝典看看

代码二:使用repeat循环,遍历一个存储了多个图层名的表,并以默认设置新建图中不存在的图层.

```
(defun c:ttt ( / lst_layName i str_layName )
    (setq lst_layName (list "1" "2" "3"));用于存储图层名的列表

    (repeat (setq i (length lst_layName));循环次数,并把表长度保存方便后面获取表内元素
        (setq i (1- i)) ;因为表的索引号是从0开始,因此最后一个元素的索引号比表的长度小1.
        (setq str_layName (nth i lst_layName))
```

```
        (if (not (tblsearch "layer" str_layName));如果图层符号表中没有这个图层
            (entmake    (list '(0 . "LAYER")
                                '(100 . "AcDbSymbolTableRecord")
                                '(100 . "AcDbLayerTableRecord") '(70 . 0)
                                '(6 . "Continuous")
                                (cons 2 str_layName)
                            )
            )
            (print (strcat "图层" str_layName "已存在"))
        )
    )
    (princ)
)
```

这里引入了一个新函数,`tblsearch`,该函数用于在符号表中搜索,函数比较简单,可以去葵花宝典看看

代码三:用foreach改写代码二

```
(defun c:ttt ( / lst_layName)
    (setq lst_layName (list "1" "2" "3"));用于存储图层名的列表

    (foreach str_layName lst_layName
        ;以下与代码二相同
        (if (not (tblsearch "layer" str_layName));如果图层符号表中没有这个图层
            (entmake    (list '(0 . "LAYER")
                                '(100 . "AcDbSymbolTableRecord")
                                '(100 . "AcDbLayerTableRecord") '(70 . 0)
                                '(6 . "Continuous")
                                (cons 2 str_layName)
                            )
            )
            (print (strcat "图层" str_layName "已存在"))
        )
    )
    (princ)
)
```

ex17: 选择集的操作

选择集是cad中经常用到的概念,它是由**若干个 不重复**的图元组成的**有序**集合.

通常我们会通过`ssget`函数引导用户选择图元新建一个选择集.该函数未选择任何对象时,会返回`nil`,选择任意个对象时,会返回一个**选择集**,即使是使用了`":S"`参数.

ssget是一个非常复杂的函数,可以指定用户以**什么形式**(框选,栏选等)选择**什么对象**(筛选).更多更详细的介绍可以看看葵花宝典,不过葵花宝典里也不全,更多的还是要靠自己摸索了.

对于选择方法,常用的参数有:

- `"C"` 窗交,即cad中从右到左的选择,会选择与窗口**相交**的所有对象.接收两个**点**.
- `"W"` 窗选,即cad中从左到右的选择,会选择**全部在窗口**内的所有对象.接收两个**点**.
- `"F"` 栏选,与多段线**相交**的所有对象,接收多个**点**组成的**点表**.
- `"X"` 全选,选择图纸内**所有的对象**,使用这种方式时,不需要用户参与
- `"P"` 上一个选择集
- `":L"` 会排除被锁定的对象

对于框选或栏选,是选不到**显示范围以外**的对象的.

ssget获取到的选择集,内部图元的排序通常是与图元建立的顺序有关.但是**栏选**和**点选**会按照**选择到的顺序**排序.

ssget**不能**发送提示信息,网络上能找到带提示的ssget,但是使用效果与原生ssget有差距.这也是Alisp现阶段不能解决的问题.

选择集也有**空**选择集,可以用不带参数的ssadd函数生成或使用`ssdel`清空一个选择集中所有图元获得.与**空表**不同,`(type '())`返回`nil`,`(type (ssadd))`返回`PICKSET`.

可以通过`sslenght`函数获取选择集的长度,用`ssname`获取选择集中指定**序列号**的图元,注意**序列号**依旧是从`0`开始.

可以通过`ssadd`和`ssdel`增减选择集,但是需要注意,这两个函数会直接修改作为参数的选择集.

同时使用的选择集数量有**上限**,Alisp只允许不超过128个选择集,因此要及时将用不到的选择集**变量**设为`nil`或将其设置为**局部变量**.

还有其他跟选择集有关的函数,例如:

- `ssnamex` 获取选择集创建的方式
- `ssgetfirst` 用于实现先选择后执行
- `sssetfirst` 在cad中**夹取**选择集,类似快速选择的功能

小试手

- 尝试用不同的参数调用ssget

ex18：调用cad的自带命令

使用`command`函数调用cad自带命令,下面的代码将调用line命令绘制一条直线

```
(command "line" pause pause "")
```

`pause`用于暂停`command`函数,由用户给定下一步操作.

- `command`函数调用cad命令后,会将其他参数**按顺序依次传递**给cad**命令行**.想要知道该以什么顺序传递什么,可以直接在命令行运行`(command "命令名")`.
  - 该函数有个特别的地方,当cad执行该函数的时候,并**不会暂停**lisp代码的执行,因此你可以在`command`函数中使用`getpoint`等函数.
  - `command`函数只是**把参数依次传递给cad命令行**,因此可以把一个`command`函数拆分成多个`command`函数,达到相同的目的,对于这点,可以参考示例.
- 有些命令会有命令名前加`.`或`_`的孪生命令,这些命令是为命令行模式优化的.
- 当需要选择**点**的时候,可以直接传入**点表**,也就是**三维坐标组成的表**,不是**点对**
- 当需要选择**对象**的时候,可以直接传入**图元名**或**选择集**
- 当需要输入**选项**的时候,可以输入该选项的**字符串**形式
- 当需要输入**数值**的时候,可以直接传入**整数**或**实数**
- 当需要输入**回车**的时候,可以直接传入**空字符串** `""`
  - 但是需要注意的是,当你传入一个参数的时候,**大部分情况下**实际上已经包含了**回车**,因此并不像在cad软件中**每步**都需要按**回车**或**空格键**,同样最后一个参数也不一定就是**空字符串**.
  - 例外的有例如复制`copy`等命令,在**选择对象的步骤**,传入**图元名**或**选择集**后,需要增加一个**空字符串**,告诉cad对象选择完毕.

    - 当`command`函数最后一个参数是该命令不需要的**空字符串**时,根据cad的设置,cad可能会将其识别为**执行上次命令**,不过并不会有实际的影响,只会在命令行提示**命令xxx(当前执行的命令名)不存在**
- 当只需要在cad中新建一个对象的时候,`entmake`函数通常比`command`函数更高效
- 默认情况下,通过`command`调用多次cad命令后,使用cad的**撤销**功能一次只会撤销一次`command`函数.可以通过cad命令`_undo`来解决该问题.

示例代码

下面的代码,将在由用户选择对象,调用cad命令修改这些对象的图层为0层.

```
(defun c:ttt ( / ss  )

    (setvar "cmdecho" 0) ;关闭命令行回显
    (command "_undo" "be") ;打开undo分组
    (setq old_error *error*)  ;备份error函数
    (setq *error* wyl:err)  ;设置自己的error函数
    ;start部分结束

    (princ "选择要修改的对象:") ;提示语
    (setq ss (ssget ":L"));选择未锁定的图层
    (command "CHPROP" ss "" "LA" "0" "" ) ;修改对象的图层

    ;end部分开始
    (command "_undo" "e") ;结束undo分组
    (setvar "cmdecho" 1) ;打开命令行回显
    (setq *error* old_error)  ;恢复默认的error函数
    (princ)
)
(princ)

(defun wyl:err (msg)
    ;当在命令执行中出现错误,或中断命令后,会执行error函数.
    ;在该函数中恢复修改了的系统设置
    (command "_undo" "e")
    (setvar "cmdecho" 1)
    (setq *error* old_error)
    (princ msg) ;打印错误提示
)
```

除了之前提过的`*error*`函数外,本例中出现了一个新的函数`setvar`,该函数用于修改**系统变量**,另外还有一个`getvar`函数用于读取**系统变量**,关于具体有那里**系统变量**和具体作用,可以看cad的帮助文件或<AutoCAD命令与系统变量>

示例代吗:拆分command函数

下面代码,将获取三个点,然后通过`command`函数,执行`line`命令绘制一个三角形.

```
(defun c:ttt ( / pt1 pt2 pt3)
    (setq pt1 (getpoint))
    (setq pt2 (getpoint pt1))
    (setq pt3 (getpoint pt2))
    (command "line" pt1 pt2 pt3 pt1 "")
)
```

而下面的代码将一个`command`函数拆分成多个`command`函数.

```
(defun c:ttt ( / pt1 pt2 pt3)
    (setq pt1 (getpoint))
    (setq pt2 (getpoint pt1))
    (setq pt3 (getpoint pt2))
    (command "line" )
    (foreach pt (list pt1 pt2 pt3 pt1)
        (command pt) ;利用foreach函数将三个点表分别传给command函数
    )
    (command "" )
)
```

ex19：调用自己写的其他命令与命令别名

调用自己写的其他命令,不需要使用`command`函数,以上节的示例`ttt`命令来说,可以直接用以下代码调用:
`(c:ttt)`
即以调用函数的方式调用命令带`C:`的名字.

这个技巧更多的用于修改命令别名,例如要把上例的命令名修改为`cc`,可以用以下代码

`(defun c:cc nil (c:ttt))`

另外还有一个技巧,很多自己写命令使用频率很低,并不需要常驻在cad中,可以通过下面方式包装一下,包装以后可以用cc命令加载lsp,之后继续用ttt命令:

```
(defun c:cc nil
    (load "xxx.lsp")
    (c:ttt)
)
(princ)
```

`load`函数用于加载lsp文件,参数是lsp文件的完整路径,当lsp文件在cad的**支持文件路径**下,可以只有文件名加后缀.

注意:路径本身是一个字符串,在字符串中作为路径分隔符的`\`是也特殊作用的(转义符),所以要把路径中的`\`替换成`\\`或`/`.

当然,如果只是这样做也有点多此一举的感觉,所以我实际上是在**命令面板**中用的.

ex20:更好用的ex8

ex8中只实现了单选,到现在已经可以实现框选批量修改了.另外增加了判断是否是数字的功能,该功能通过**正则表达式**实现,如果展开了介绍正则表达式,恐怕还需要这么多的篇幅,所以这里省略了.如果有兴趣,可以自行了解这个**强大的**功能.

以下代码将在cad中定义一个命令名ttt的命令,框选多个**单行文本**后,将在数字形式的**单行文本**后增加**米**做为后缀.

```
(defun c:ttt ( / ss i wyl:todo XD::String:RegExpS )
    ;
    ;说明:主要执行函数,判断对象是否是正确的数字格式的字符串.
    ;参数:en:图元名
    ;返回值:无
```

```
    (defun wyl:todo(en / str1 ed1 ed)
        (setq ed (entget en))
        (setq str1 (cdr (assoc 1 ed)));获取单行文字的内容

        (if (XD::String:RegExpS "^-?[1-9]\\d*(\\.\\d+)?$" str1 "");中间这串乱码就是
正则表达式,匹配函数用的现成的
            (progn
                (setq str1 (strcat str1 "米"));文字内容后加上"米"
                (setq ed1 (subst (cons 1 str1) (assoc 1 ed) ed));替换信息表中组码1
的内容
                (entmod ed1) ;修改图元
            )
        )
    )
    ;下面这个函数里的内容不需要深究
    ;
    ;功能 对字符串进行正则表达式匹配测试.
    ;参数:
    ;pat = 正则表达式模式 ,对应vbs正则表达式的模式(expression)。说明: \\号要用\\\\替代.
    ;str = 字符串
    ;key = i g m , i不区分大小写（Ignorecase）,g全局匹配（Global）.   m 多行模式
（Multiline），以上几个关键字可以组合使用，或用 .
    ;返回: 返回匹配的字符列表，或无一匹配返回nil

    (defun XD::String:RegExpS (pat str key / end1 keys matches x)
        (if (not *xxvbsexp)
            (setq *xxvbsexp (vlax-get-or-create-object "VBScript.RegExp"))
        )
        (vlax-put *xxvbsexp 'Pattern pat)
        (if (not key)
            (setq key "")
        )
        (setq key (strcase key))
        (setq keys '(("I" "IgnoreCase") ("G" "Global")
                                ("M" "Multiline")
                            )
        )
        (mapcar
            '(lambda (x)
                 (if (wcmatch key (strcat "*" (car x) "*"))
                     (vlax-put *xxvbsexp (read (cadr x)) 0)
                     (vlax-put *xxvbsexp (read (cadr x)) -1)
                 )
             )
            keys
        )
        (setq matches (vlax-invoke *xxvbsexp 'Execute str))
        (vlax-for x matches (setq end1 (cons (vla-get-value x) end1)))
        (reverse end1)
    )

    ;main
    (princ "选择需要增加后缀的单行文本")
    (setq ss (ssget (list (cons 0 "TEXT"))));筛选单行文本

    (repeat (setq i (sslength ss))
        (setq i (1- i))
        (wyl:todo (ssname ss i))
```

```
    )
    (princ)
)
```

在这个例子里,把函数写成了命令里的局部函数,前面说过,lisp是解释型语言,因此**定义函数**需要在**调用函数**前.

其实从这个例子也能看出来,想实现一个功能只需要几行就够了(比如ex8),但是想让一个程序好用,代码量翻倍的增加.

到现在为止,如果认真的看完了上面所有的内容,配合参考资料已经可以自行完成80%的Alisp编程了,剩下的篇幅将用来介绍Alisp的几个进阶函数和Vlisp的简单介绍.

ex21:更多自定义函数相关

这里说的自定义函数是指除命令外的**自定义通用函数**.函数也被称为**轮子**,轮子造好了,放个板子连接上各个轮子,这就是一辆车.所以轮子的积累是编程中最重要的一部分.

在平时的编程中,应该有意识的把一些常用的功能写成通用的函数,要实现这一目的,就要好好规划函数的功能,尽量小而精.

另外网络上也有很多函数库,例如**晓东lisp API**,码云平台上的**AutoLispBaseFunctionLibrary**.

另外晓东还有一个XDRX API因为是用c++写的,能实现更多Alisp实现不了的功能,不过这个API比较重.

多积累自己的函数库(宁缺毋滥),能有效的提高自己的效率.

ex22:匿名函数lambda

`lambda` 函数的作用就是写一个**匿名的函数**.但是我认为这样定义显示不出来lambda的作用.而且会让初学者没东西接触lambda,介绍lambda,就要一起介绍mapcar和apply函数.本节只是单纯的介绍lambda的用法.

个人看法:lambda、mapcar和apply的配合能让你的代码更优雅更简练,但并不是**必须**的.如果没精力,不学也没太大的问题.

`lambda` 用来定义一个匿名函数,使用方法类似 `defun` 函数,用defun定义的函数,把defun和函数名替换成lambda就可以了.

下面用defun定义一个简单的函数:

```
(defun wyl:add1( a )
    (1+ a)
)
```

如果要定义成匿名函数,只需要改成这样

```
(lambda ( a )
    (1+ a)
)
```

你可以把匿名函数赋值给一个变量,因此下面的代码与上面defun定义的函数**效果是一样**的

```
(setq wyl:add1
    (lambda ( a )
        (1+ a)
    )
)
```

可以在命令行运行以下代码测试 `(wyl:add1 1)`

ex23:mapcar,apply

mapcar

该函数接收多个参数,其实第一个参数是一个**函数的本身**(别忘记前面的`'`),其余参数均是**表**.

该函数会依次把第二个及以后的表中的**每一个元素依次传递**给作为第一个参数的函数.作为参数的**表的数量**需要与作为第一个参数的**函数**能**接受的参数**数量一致.之后会把每次**作为参数的函数**运算后的返回值合成一个**表**作为返回值返回.

mapcar参数里各个表并不要求长度一致,但是mapcar只会计算最短的表的长度,**多余部分会被舍弃**.

当传入表的数量不满足作为第一个参数的函数需要的参数数量时,会报错.

示例

`(mapcar '+ (list 1) (list 2))` 返回 `(3)`

`(mapcar '+ (list 1 2) (list 2 3))` 返回 `(3 5)`

`(mapcar '+ (list 1 ) (list 2 3))` 返回 `(3)`

`(mapcar '1+ (list 1 ) (list 2 ))` 报错 **参数太多**

`(mapcar 'eq (list 1 ))` 报错 **参数太少**

`(mapcar '+ 1 2 )` 报错 **参数类型错误**

`(mapcar '+ (cons 1 2 ) (list 2 3 ))` 报错 **列表错误**

apply

该函数**只接收**两个参数,第一个参数是一个**函数的本身**,第二个参数是一个**表**.

该函数会把表中的**每一个元素同时传递给**第一个参数.如果**表的长度**与作为第一个参数的函数能接收的**参数的数量**不符,会报错.

该函数会返回第一个参数的函数的返回值.

示例

`(apply '+ (list 1 2))` 返回 `3`

`(apply '+ (list 1 2 3))` 返回 `6`

`(apply '1+ (list 1 ))` 返回 `2`

`(apply '1+ (list 1 2 ))` 报错 **参数太多**

`(apply 'eq (list 1 ))` 报错 **参数太少**

`(apply '+ (list 1 2) (list 1 2))` 报错 **参数太多**

ex24:mapcar,apply和lambda的配合

这三个函数每个函数单拉出来都会觉得平平无奇,但是配合起来,那就是古天乐了.

macpar与lambda配合的示例代码

以下代码以正常方式定义一个函数,接收两个**二维点**作为参数,返回两个点连线的中点.

```
(defun wyl:ttt(pt1 pt2 / x y)
    (setq x (/ (+ (car pt1) (car pt2)) 2))
    (setq y (/ (+ (cadr pt1) (cadr pt2)) 2))
    (list x y)
)

(defun c:ttt ( / pt1 pt2 pt_mid);测试语句
    (setq pt1 (list 1 2)
         pt2 (list 2 1)
        pt_mid (wyl:ttt pt1 pt2)
    )
)
```

以下代码与上面代码功能相同,用lambda和mapcar函数改写一下

```
(defun c:ttt ( / pt1 pt2 pt_mid);测试语句
    (setq pt1 (list 1 2)
         pt2 (list 2 1)
    )
    (setq pt_mid (mapcar '(lambda (a b ) (/ (+ a b) 2)) pt1 pt2));注意lambda前面
的'
)
```

macpar与apply配合的示例代码

现在有列表(1 2 3),我们需要将列表中每一个元素+1后求合.

下面代码是用foreach函数的实现.

```
(setq a 0)
(foreach x '(1 2 3)
 (setq x (1+ x))
 (setq a (+ a x))
)
```

下面是用macpar与apply改写后的代码

```
(apply '+ (mapcar '1+ '(1 2 3)))
```

macpar、apply与lambda配合的示例代码

有一个表**lst_lay**,里面保存了不确定数量的图层名,假设现在表是这样的("a" "b" "c") ,现在需要用户通过序号选择其中一个图层,需要有类似下面的提示:

```
(princ "\n选择需要恢复的图层状态")
(princ (strcat
"\n1.a"
"\n2.b"
"\n3.c"))
```

因为表的长度不确定,需要在输出的时候生成序号拼接.

下面代码是使用foreach函数实现的.

```
(setq i 0)
(setq a "")
(foreach x '("a" "b" "c")
  (setq i (1+ i))
  (setq a(strcat a "\n" (itoa i) "." x ))
)
```

下面改写后的代码

```
(setq i 0)
(apply 'strcat
    (mapcar '(lambda(x)
                        (setq i (1+ i))
                        (strcat "\n" (itoa i) "." x )
                    )
        '("a" "b" "c")
    )
)
```

ex25:其他有用的函数

这里列举了几个我自己学习过程中不重视,但是后来发现很有用的几个函数

- `grread` 是Alisp的一个进阶函数,他**即时**的获取用户的各种输入如按键、鼠标移动、鼠标点击等.通过该函数,能实现很多需要即时输出的功能
- `redraw` 该函数能亮显一个对象
- `(command "zoom" "o" en/ss "")` 这条代码能让视口按照对象en或选择集ss缩放
- `read` 和 `eval` 用来把字符串当代码执行的函数
- `grvecs` 在屏幕上绘制临时显示的线
- `trans` 将点坐标在世界坐标系和用户坐标系之间转换的函数

ex26：了解Vlisp

前面说过Alisp的面向过程的编程语言,而Vlisp相对Alisp最大的变化就是融入了部分面向对象的概念.但是Vlisp依旧是面向过程的.

面向对象的概念这里就不细说了,只说一下在Vlisp下的优点.

- **大部分**工作Vlisp比Alisp的效率更高
- 因为引入了面向对象的概念,简化了设计工作

对于第二点多说一点.

在Alisp中,图元的属性是保存在**图元信息表**中的,受性能限制,信息表中不会保存图元的所有属性,大部分属性都要通过计算得出.

在ex6中展示了通过Alisp计算直线长度的代码,摘抄如下:

```
(setq en (car (entsel "选择一个直线:")));引导用户选择一个对象,car函数用于获取entsel函数返回值中的图元名
(setq ed (entget en));获取图元的信息表
(setq start (cdr (assoc 10 ed)));获取直线的起点
(setq end (cdr (assoc 11 ed)));获取直线的终点
(setq l (distance start end));计算起点到终点的距离,即直线的长度
```

而在Vlisp下,以上代码可以简化为:

```
(setq en (car (entsel "选择一个直线:")));引导用户选择一个对象,car函数用于获取entsel函数返回值中的图元名
(setq obj (vlax-ename->vla-object en)) ;将图元名转换成VLisp中用的OBJ
(setq l (vla-get-Length obj)) ;获取对象的长度
```

另外一方面,很多不同的对象有相同的属性,对这类对象,在Alisp中需要根据图元的类型,按不同的方式计算,而Vlisp不需要.

在ex6中提到多段线比较复杂,是因为多段线的每个顶点坐标在信息表中都是以组码10保存的,下面用Alisp改些ex6,让其同时支持多段线和直线.

```
(defun c:ttt ( / en ed start end l en_type pt1 l_tmp )
    (setq en (car (entsel "选择一个直线或多段线:")));引导用户选择一个对象,car函数用于获取entsel函数返回值中的图元名
    (setq ed (entget en));获取图元的信息表
    (setq en_type (cdr (assoc 0 ed)))

    (setq l (cond
                ((= "LINE" en_type)
                    (setq start (cdr (assoc 10 ed)));获取直线的起点
                    (setq end (cdr (assoc 11 ed)));获取直线的终点
                    (distance start end)
                )
                ((= "LWPOLYLINE" en_type)
                    (setq l_tmp 0) ;用于存放长度的临时变量
                    (setq pt1 (cdr (assoc 10 ed)));多段线的第一个顶点
                    (setq ed (cdr (member (cons 10 pt1) ed)));信息表中第一个顶点以后的部分
                    (foreach x ed
                        (if (= 10 (car x));如果是顶点坐标
                            (progn
                                (setq l_tmp (+ l_tmp (distance pt1 (cdr x))));累加各段长度
                                (setq pt1 (cdr x));更新顶点坐标
                            )
                        );end if
                    );end foreach
                    l_tmp;返回累加后的值
                )
                (t
                    (print "选择的对象不是直线或多段线")
                    (exit)
                )
            );end cond
    )
```

```
    (princ (strcat "\n长度为" (rtos l) "\n"));输出直线的长度
    (princ)
)
```

下面是用Vlisp改写后的代码

```
(defun c:ttt ( / en obj l  )
    (setq en (car (entsel "选择一个直线或多段线:")));引导用户选择一个对象,car函数用于获取entsel函数返回值中的图元名
    (if (not (member (cdr (assoc 0 (entget en))) (list "LINE" "LWPOLYLINE")));如果不是直线或多段线
        (progn
            (print "选择的对象不是直线或多段线")
            (exit))
    )
    (setq obj (vlax-ename->vla-object en)) ;将图元名转换成VLisp中用的OBJ
    (setq l (vla-get-Length obj)) ;获取对象的长度
    (princ (strcat "\n长度为" (rtos l) "\n"));输出直线的长度
    (princ)
)
```

结语

历时三天写完初版,共26节,四万余字,接下来还要补充些东西,校对一下,添加一些图片.

写本文并不是想让人看完一后就一飞冲天了,只是希望能有更多的人进入Alisp这个圈子.

Alisp是一个目的导向很强的语言,除了少部分做CAD二开的人以外,更多的人是因为工作中需要用CAD,抱着提高自己效率,让自己有更多休息时间才来学的.

如果真按照其他语言,比如<C++ prime plus>那样写,九百多页,两百万字,又有多少人能看的下去呢?

又或者可以像大多数编程书一样,把教材写成手册,先讲数据类型、变量、运算符再讲栈堆算法,一本书看完也找不到一个完整的程序的示例,我就看过这样的书<C#图解教程>,书的质量确实不错,可是看到二百多页还在类的概念里挣扎,完全不知道怎么把自己学到的东西应用起来.

我觉得这样是很打击初学者的积极性的.

如果你希望依靠Alisp吃饭,比如去天正、鸿业之类的二开公司工作,那你应该从<C++ prime plus>开始看,但是如果你只是有兴趣,抛开那么大块头,从实战开始吧.

本人能力有限,时间也比较短,肯定会有大量的错漏,希望高手能指出来.

---

1. lisp是第二古老的高级编程语言.因此缺少大量常见编程语言拥有的特性,例如for循环(lisp中的foreach循环与for循环完全不同),中断或跳过循环,数组(lisp中的表更类似列表而不是数组) ↩

2. 编程语言的方言就是在母体语言的基础上,增加功能或特性的新的语言. ↩

3. 编译型语言是在编译时将代码翻译成机器能识别的语言.解释型语言是在执行时才将代码翻译成机器能识别的语言,因为在执行时多了翻译的步骤,执行效率通常会低于编译型语言. ↩

4. 面向过程是一种以过程为中心的编程思想 ↩

5. 有个笑话是,苏联间谍偷到了美国核弹发射台的程序源代码,但是拿到手发现只有左括号. ↩

6. 关键字就是程序自己使用的**名字**,例如内置的函数等 ↩ ↩ ↩

7. 字符串是代表文字内容的数据,需要被引号包裹. ↩ ↩

8. Debug,调试,用于找出代码中的错误或遗漏.调试除了找到程序中的错误,更多的是利用不同的样本测试程序有什么意外的错误. ↩

9. C语言的第一个演示程序就是Hello World ,通过该演示程序能展示本语言所需的**最小结构体** ↩

10. 建议自己输入到文本编辑器中,而不是通过复制粘贴的方式 ↩ ↩

11. 绘图区指的是CAD中可以用来绘图的黑色区域(某写版本是其他颜色),Alisp源码文件可以通过拖放到绘图区加载,拖放到工具栏上不会有任何效果. ↩ ↩

12. 在大部分编程语言中,都会有一部分符号有特殊的含义,例如Alisp中的双引号 " ,如果需要在字符串(由双引号包裹)中表现一个双引号,就需要用到**转义符**,告诉解释器,这个引号就只是引号,请不要和其他引号配对. ↩