Universidad de León

Escuela Superior y Técnica
de Ingenieros de Minas

GRADO EN INGENIERÍA DE LA ENERGÍA

TRABAJO FIN DE GRADO

PARAMETRIZACIÓN Y MODELIZACIÓN DE ELEMENTOS MECÁNICOS MEDIANTE LENGUAJE DE PROGRAMACIÓN EN AUTOCAD.

León, Septiembre de 2016

Autor: Carlos Rodríguez Belloso
Tutor: D. Jaime Cifuentes Rodríguez

El presente proyecto ha sido realizado por D. Carlos Rodríguez Belloso, alumno de la Escuela Superior y Técnica de Ingenieros de Minas de la Universidad de León para la obtención del título de Grado en Ingeniería de la Energía.

La tutoría de este proyecto ha sido llevada a cabo por D. Jaime Cifuentes Rodríguez, profesor del Grado en Ingeniería de la Energía.

Visto Bueno

Fdo.: D. Carlos Rodríguez Belloso	Fdo.: D. Jaime Cifuentes Rodríguez
El autor del Trabajo Fin de Grado	El Tutor del Trabajo Fin de Grado

RESUMEN

El presente estudio que voy a desarrollar tiene como objetivo mostrar las posibilidades que tienen la combinación de programas CAD, como AutoCAD, y los lenguajes de programación.

Mediante la combinación de esas dos herramientas, aumentan nuestras posibilidades a la hora de trabajar con dibujos técnicos y geométricos.

En el estudio, presentaré las características básicas del programa AutoCAD y del lenguaje de programación AutoLISP y, además, diseñaré dos rutinas para ver las posibilidades de esta combinación.

La primera rutina será para la elaboración de tornillos de cabeza hexagonal y, la segunda, para la resolución de sondeos para encontrar el estrato de un mineral.

ABSTRACT

This study that i will develop is aimed at showing the possibilities for the combination of CAD programs such as AutoCAD, and programming languages.

By combining these two tools, it increases our possibilities when working with technical and geometric designs.

In the study, I will introduce the basic features of the AutoCAD program and AutoLISP programming languages and also will design two routines to see the possibilities of this combination.

The first routine will be to prepare hex head screws and the second will be for for the resolution of polls to find the layer of a mineral.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 AutoCAD

1.1 ¿Qué es AutoCAD?

AutoCAD es un software de diseño asistido por computadora utilizado para dibujo 2D y modelado 3D, es decir, es un programa de dibujo técnico desarrollado por Autodesk para el uso de ingenieros, técnicos y otros profesionales de carreras de diseño.

El nombre AutoCAD surge como creación de la compañía Autodesk, donde Auto hace referencia a la empresa y *CAD* a dibujo asistido por computadora (por sus siglas en inglés *computer assisted drawing*), teniendo su primera aparición en 1982.

Las interfaces de programación que admite AutoCAD son ActiveX Automation, VBA (Visual Basic® for Applications), AutoLISP, Visual LISP, ObjectARX y .NET. El tipo de interfaz que se utilice dependerá de las necesidades de la aplicación y de la experiencia en programación de cada usuario.

1.2 ¿Para qué sirve AutoCAD?

AutoCAD es un programa, como su nombre dice, para diseñar, en el que se puede realizar todo tipo de diseños técnicos muy útiles para ingenieros, arquitectos, etc., pudiendo crear diseños en 2d y 3d, planos, objetos, cortes de objetos, etc.

1.3 Funciones y comandos

1.3.1 Funciones

Al igual que otros programas de Diseño Asistido por Ordenador (DAO), AutoCAD gestiona una base de datos de entidades geométricas (puntos, líneas, arcos, etc.) con la que se puede operar a través de una pantalla gráfica en la que se muestran estas, el llamado editor de dibujo. La interacción del usuario se realiza a través de comandos, de edición o dibujo, desde la línea de órdenes a la que el programa está fundamentalmente orientado. Las versiones modernas del programa permiten la introducción de éstas mediante un usuario en inglés GUI, que automatiza el proceso.

Como todos los programas de DAO, procesa imágenes de tipo vectorial, aunque admite incorporar archivos de tipo fotográfico o mapa de bits, donde se dibujan figuras básicas o primitivas (líneas, arcos, rectángulos, textos, etc.), y, mediante herramientas de edición, se crean gráficos más complejos. El programa permite organizar los objetos por medio de *capas* o estratos, ordenando el dibujo en partes independientes con diferente color y grafismo. El dibujo de objetos seriados se gestiona mediante el uso de *bloques*, posibilitando la definición y modificación única de múltiples objetos repetidos.

La extensión del archivo de AutoCAD es **.**dwg, aunque permite exportar en otros formatos (el más conocido es el .dxf). Maneja también los formatos IGES y STEP para manejar compatibilidad con otros softwares de dibujo. El formato .dxf permite compartir dibujos con otras plataformas de dibujo CAD, reservándose AutoCAD el formato.dwg para sí mismo. El formato.dxf puede editarse con un procesador de texto básico, por lo que se puede decir que es abierto. En cambio, el.dwg solo podía ser editado con AutoCAD, si bien desde hace poco tiempo se ha liberado este formato (DWG), con lo que muchos programas CAD distintos del AutoCAD lo incorporan, y permiten abrir y guardar en esta extensión, con lo cual lo del DXF ha quedado relegado a necesidades específicas.

1.3.2 Comandos básicos

Desde sus primeras versiones, AutoCAD ha usado los mismos comandos básicos para dibujar y se van añadiendo nuevos comandos a medida que salen las nuevas versiones.

- Dibujo 2d

Tabla 1-1. Comandos AutoCAD para dibujo 2d

Comando	Alias	Botón	Detalle
LINE	L		Dibuja rectas consecutivas especificando un punto inicial y los subsiguientes.
CIRCLE	C		Dibuja un círculo especificando su centro y radio (o diámetro)
ARC	A		Dibuja un arco especificando tres puntos
RECTANGLE	REC		Dibuja un rectángulo especificando dos esquinas opuestas
ELLIPSE	EL		Dibuja una elipse especificando un eje (recta) y la distancia al extremo de su otro eje
POLYGON	POL		Dibuja un polígono regular especificando su número de lados, su centro y un radio (a un vértice o al centro de un lado)

PLINE	PL		Dibuja rectas y arcos consecutivos de la misma forma que el comando LINE dando como resultado un solo objeto de varios segmentos.
HATCH	H		Dibuja un sombreado especificando el área a sombrear y el tipo de textura en un cuadro de diálogo.

- Visualización

Tabla 1-2. Comandos AutoCAD para visualización

Comando	**Alias**	**Botón**	**Función**
PAN	P		Arrastra la vista en la pantalla. Mantiene la ampliación.
ZOOM Window	Z, W		Amplía la vista especificando una ventana rectangular.
ZOOM Dynamic	Z, D		Cambia la vista de forma dinámica, especificando tamaño y posición de una ventana
ZOOM Scale	Z, S		Cambia la ampliación de la vista especificando un factor respecto de la vista actual (X) o respecto del espacio papel (XP)
ZOOM Center	Z, C		Mueve la vista centrando un punto especificado en el dibujo.
ZOOM Object	Z, O		Amplía la vista de un objeto seleccionado.
ZOOM All	Z, A		Muestra todos los objetos dibujados y los límites de la grilla previamente definida

- Modificación

Tabla 1-3. Comandos AutoCAD para modificación

Comando	Alias	Botón	Detalle
MOVE	M		Mueve objetos seleccionados, especificando dos puntos de desplazamiento
COPY	C		Copia objetos seleccionados, especificando dos puntos de desplazamiento
ERASE	E		Borra los objetos seleccionados
ARRAY	AR		Copia objetos seleccionados de forma múltiple, especificando parámetros en un cuadro de diálogo.
ROTATE	RO		Gira objetos seleccionados, especificando punto base de rotación y ángulo de referencia
STRETCH	S		Modifica objetos moviendo sus vértices seleccionados con ventana segmentada
TRIM	TR		Corta segmentos de objetos a partir de intersecciones.
EXTEND	EX		Extiende una línea hasta otra previamente especificada
OFFSET	O		Crea objetos que distan la misma longitud del objeto original en todos sus puntos
MIRROR	M		Crea objetos de forma simétrica al original respecto de un eje “espejo” de dos puntos.

FILLET	F		Redondea un vértice. Se puede especificar el radio del arco de redondeo
CHAMFER	CHA		Recorta un vértice. Se puede especificar la distancia desde el vértice a cada lado
EXPLODE	X		Descompone un objeto de múltiples lados en rectas y arcos sencillos

- Referencias a objetos

Tabla 1-4. Comandos AutoCAD para referencias a objetos

Comando	**Alias**	**Botón**	**Detalle**
Endpoint	END		Acerca el cursor al punto final de una línea
Midpoint	MID		Acerca el cursor al punto medio de una línea
Node	NOD		Acerca el cursor a un punto
Intersection	INT		Acerca el cursor a una intersección de dos o más líneas
Extension	EXT		Ubica el cursor sobre la extensión de una línea
Center	CEN		Acerca el cursor al centro de un círculo, de un arco o de una elipse
Quadrant	QUA		Acerca el cursor al cuadrante de un círculo, de un arco o de una elipse

Tangent	TAN		Acerca el cursor a un punto de tangencia
Insertion	INS		Acerca el cursor al punto de inserción de un bloque
Perpendicular	PER		Acerca el cursor a un punto perpendicular
Nearest	NEA		Muestra el punto más cercano al cursor sobre un objeto
Parallel	PAR		Ubica el cursor sobre una paralela a otra recta

2 AutoLISP. Conceptos previos y definiciones

2.1 Introducción

AutoLISP es un lenguaje de programación derivado del lenguaje lisp. Es utilizado para generar rutinas orientadas al uso específico de AutoCAD y sus derivados. Permite desarrollar programas y funciones para el manejo de entidades del tipo gráfico.

2.2 Lenguaje de programación LISP

El Lisp (o LISP) es una familia de lenguajes de programación de computadora de tipo multiparadigma con una larga historia y una sintaxis completamente entre paréntesis.

Especificado, originalmente, en 1958 por John McCarthy y sus colaboradores en el Instituto Tecnológico de Massachusetts, el Lisp es el segundo lenguaje más antiguo de programación de alto nivel de extenso uso hoy en día, solamente el FORTRAN es más viejo.

El Lisp fue creado, originalmente, como una notación matemática práctica para los programas de computadora, basada en el cálculo lambda de Alonzo Church. Se convirtió rápidamente en el lenguaje de programación favorito en la investigación de la inteligencia artificial (AI). Como uno de los primeros lenguajes de programación, el Lisp fue pionero en muchas ideas en ciencias de la computación, incluyendo las estructuras de datos de árbol, el manejo de almacenamiento automático, tipos dinámicos y el compilador auto contenido.

Su nombre proviene de LISt Processing (Procesado de Listas), puesto que la base de su funcionamiento es el manejo de listas en vez de datos numéricos.

Una lista es un conjunto de símbolos que pueden ser nombres de variables, datos concretos numéricos o textuales, funciones definidas por el propio usuario, etc. El símbolo es, pues, la unidad básica con un contenido o un significado para el programa en LISP.

La lista es el mecanismo que junta una serie de símbolos y los evalúa, es decir, los procesa obteniendo un resultado. El lenguaje LISP procesa directamente las listas en cuanto se encuentran formadas y obtiene o "**devuelve**" el resultado de ese proceso.

Esta característica del manejo de listas otorga al LISP una gran versatilidad y le distingue de otros lenguajes de programación orientados a la manipulación de números.

Las ventajas que supone la utilización de un lenguaje basado en LISP para programar desde AutoCAD se podrían resumir en los siguientes puntos:

- Facilidad para manejar objetos heterogéneos: números, caracteres, funciones, entidades de dibujo, etcétera. Para LISP, basta representar cualquiera de esos objetos con un "símbolo" y no hay necesidad de definir previamente qué tipo de datos va a contener ese símbolo.
- Facilidad para la interacción en un proceso de dibujo.
- Sencillez de aprendizaje y comprensión.
- El hecho de que el LISP sea un lenguaje muy utilizado en investigación y desarrollo de Inteligencia Artificial y Sistemas Expertos.
- Sencillez de sintaxis.

El LISP es un lenguaje que es evaluado en vez de compilado o interpretado.

En los lenguajes interpretados por la computadora, cada palabra es convertida a lenguaje máquina, lo que hace que sean muy lentos. En cambio, los lenguajes compilados son mucho más rápidos porque en el proceso de compilación todo el programa se convierte en instrucciones de máquina.

Un lenguaje evaluado como el LISP es un paso intermedio entre los interpretados y los compilados. No es tan rápido como estos últimos pero resulta mucho más flexible e interactivo.

El mecanismo evaluador de LISP es la propia lista. Una lista es un conjunto de símbolos separados entre sí por, al menos, un espacio en blanco y encerrados entre paréntesis. Desde el momento en que existe una expresión encerrada entre paréntesis, el LISP la considera como una lista y la evalúa intentando ofrecer un resultado.

El LISP no es un lenguaje de programación único, sino que existen muchas versiones de LISP: MacLISP, InterLISP, ZetaLISP, Common LISP.

2.3 AutoLISP, una versión específica de lisp

AutoLISP es la herramienta más potente para optimizar la ejecución de AutoCAD. Le habilita para "automatizar" AutoCAD, incluso más allá de lo que puede llevar a cabo usando macros.

Los programas hechos en AutoLISP amplían los comandos y aplicaciones de AutoCAD creando, así, una solución óptima para cada problema en particular, desde el simple trazo de una línea hasta el diseño de un plano o pieza, llegando a cálculos complejos, convirtiéndose en gran ayuda para las aplicaciones de ingeniería.

Entre las aplicaciones más notables de AutoLISP se pueden citar:

- Dibujo de figuras bidimensionales con características específicas.
- Creación de objetos tridimensionales.
- Generación de gráficas de funciones basándose en ecuaciones.
- Cálculos de áreas y tablas de datos, combinación de comandos de dibujo para realizar determinados tipos de tareas.
- La creación de nuevas y únicas órdenes AutoCAD.
- La inserción de funciones especiales para dibujar y para calcular.
- Análisis detallados de gráficos y de dibujos dentro del editor de dibujos de AutoCAD.

Los programas en AutoLISP son simples archivos de texto, con la extensión obligatoria .LSP, donde el usuario escribe uno o varios programas contenidos en ese archivo. Una vez hecho esto, basta cargar el archivo.

Además, se pueden escribir directamente instrucciones en AutoLISP desde la línea de comandos del dibujo en AutoCAD, es decir, escribir conjuntos de símbolos encerrados entre paréntesis. AutoLISP evalúa inmediatamente esa lista y ofrece un resultado. Y, si la expresión contiene definiciones de funciones o variables, quedan cargadas en la memoria para su utilización posterior.

2.4 Elementos del lenguaje

Los objetos en AutoLISP representan todos los tipos de componentes de un programa. En esencia son dos, como ya se ha dicho, listas y símbolos.

Atendiendo a sus características se pueden definir varios tipos de objetos en AutoLISP:

- Lista: es un objeto compuesto de:
    - Paréntesis de apertura.
    - Uno o más elementos separados por, al menos, un espacio en blanco.
    - Paréntesis de cierre.

  - Los elementos de la lista pueden ser símbolos, valores concretos (“constantes" numéricas o textuales), o, también, otras listas incluidas.

- Elemento: cualquiera de los componentes de una lista.
- Átomo: representa una información indivisible, un valor concreto o un símbolo de variable que contiene un valor.
- Símbolo: cualquier elemento de la lista que no sea un valor concreto. El símbolo puede ser un nombre de variable, un nombre de función definida por el usuario o un nombre de comando de AutoLISP.
- Enteros: valores numéricos enteros, ya sean explícitos o contenidos en variables.
- Reales: valores numéricos con precisión de coma flotante.

2.5 Procedimientos de evaluación en AutoLISP

La base de todo intérprete de LISP es su algoritmo evaluador. Este analiza cada línea del programa y devuelve un valor como resultado. La evaluación en AutoLISP se realiza de acuerdo con las siguientes reglas:

- Los valores enteros, reales, cadenas de texto, descriptores de archivos, así como los nombres de subrutinas o comandos de AutoLISP, devuelven su propio valor como resultado.
- Los símbolos de variables participan con el valor que contienen (que les está asociado) en el momento de la evaluación
- Las listas se evalúan de acuerdo con el primer elemento. Si este es un nombre de función inherente o comando, los elementos restantes de la lista son considerados como los argumentos de ese comando. En caso contrario, se considera como un nombre de función definida por el usuario, también, con el resto de elementos como argumentos.

Cuando los elementos de una lista son a su vez otras listas, estas se van evaluando desde el nivel de anidación inferior (las listas más “interiores"). Puesto que cada lista contiene un paréntesis de apertura y otro de cierre, cuando existen listas incluidas en otras, todos los paréntesis que se vayan abriendo deben tener sus correspondientes de cierre.

También es posible evaluar directamente un símbolo (extraer, por ejemplo, el valor actual contenido en una variable) tecleando el signo de cerrar admiración seguido del nombre del símbolo.

- Por ejemplo: ! P1

2.6 Convenciones de AutoLISP

- Las expresiones contenidas en un programa de AutoLISP pueden introducirse directamente desde el teclado durante la edición de un dibujo de AutoCAD, escribirse en un archivo de texto ASCII o ser suministradas por una variable del tipo cadena.
- Los nombres de símbolos pueden utilizar todos los caracteres imprimibles (letras, números, signos de puntuación, etc.), salvo los prohibidos:

  ( ) . ' “ ;
- Los caracteres que terminan un nombre de símbolo o un valor explícito (una constante numérica o de texto) son:

  ( ) ' " ; (espacio en blanco) (fin de línea)
- Una expresión puede ser tan larga como se quiera, es decir, puede ocupar varias líneas del archivo de texto.
- Los espacios en blanco de separación entre símbolos son interpretados como un solo espacio en cada par de símbolos.
- Los nombres de símbolos no pueden empezar por una cifra. Las mayúsculas y minúsculas son diferentes para AutoLISP.
- Los valores explícitos (constantes) de números pueden empezar con el carácter + o , que es interpretado como el signo del número.
- Los valores de constantes de número reales deben empezar con una cifra significativa. El carácter se interpreta como el punto decimal. También se admite + o - para el signo y "e" o “E” para notación exponencial (científica).
- No es válida la coma decimal ni tampoco se puede abreviar, como en "6" (hay que poner Ø.6)

3 Programación en AutoLISP

3.1 Creación de un programa en AutoLISP

Los programas en AutoLISP son archivos de texto con la extensión .LSP. Por tanto, se crean directamente con un Editor.

Nombre de archivo: linea.lsp

La función creada línea, al ser cargada dentro de AutoCAD, y ejecutada, provocará la creación de una línea entre dos puntos indicados.

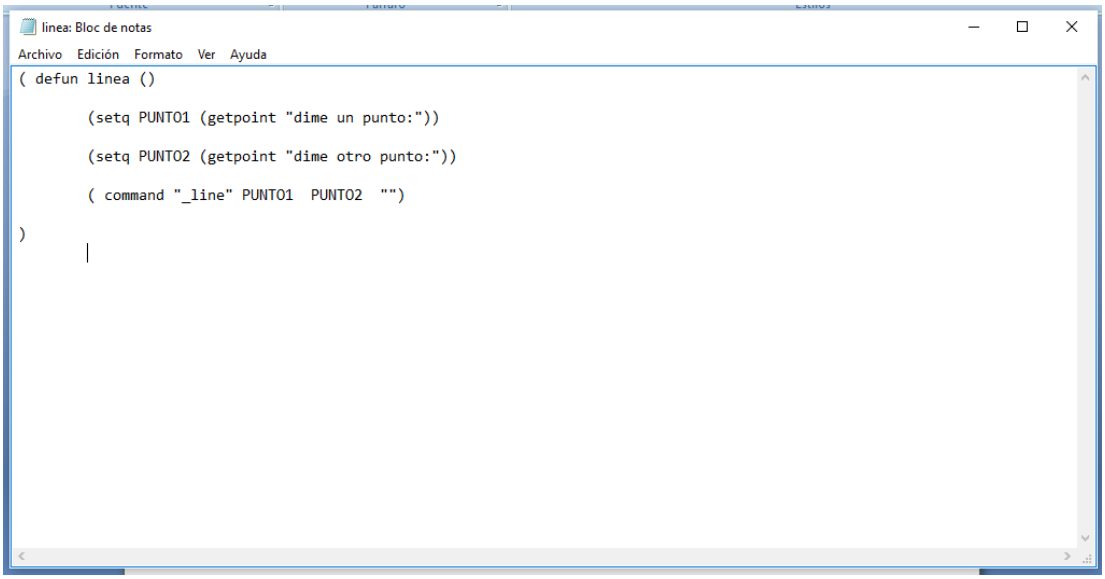


Figura 3-1.Ejemplo de programa en AutoLISP

3.2 Cargar programas

Existen principalmente tres formas para cargar un programa en AutoCAD:

- Directamente sobre la barra de comandos.

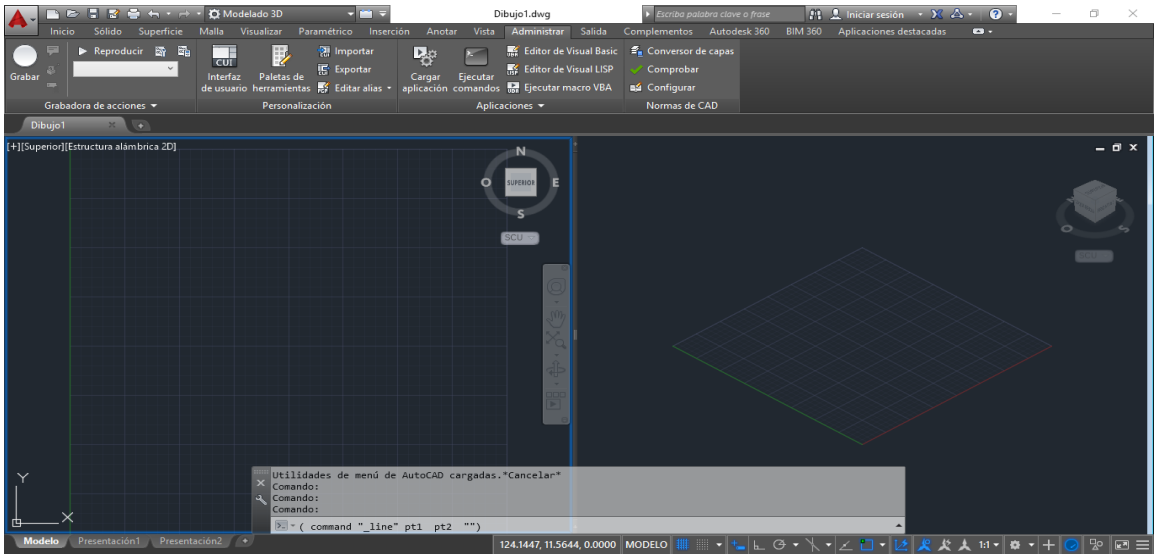


Figura 3-2. Carga de programa informático Modo 1

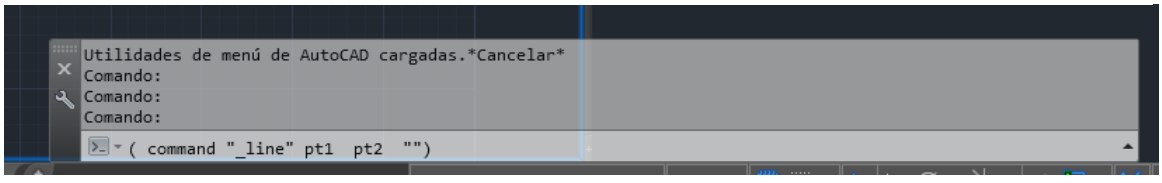


Figura 3-3. Carga de programa informático Modo 1

- Cargando el fichero desde el exterior, mediante la pestaña “cargar aplicación”.

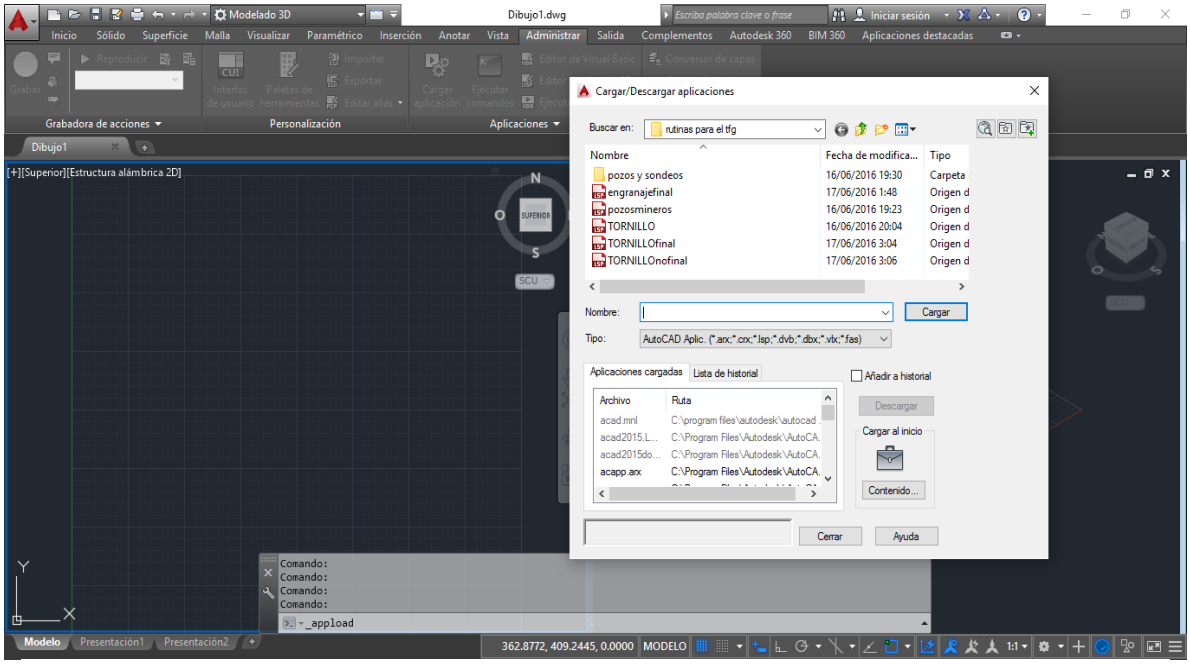


Figura 3-4. Carga de programa informático Modo 2

- A través del entorno visual, mediante la pestaña “editor visual de LISP”.

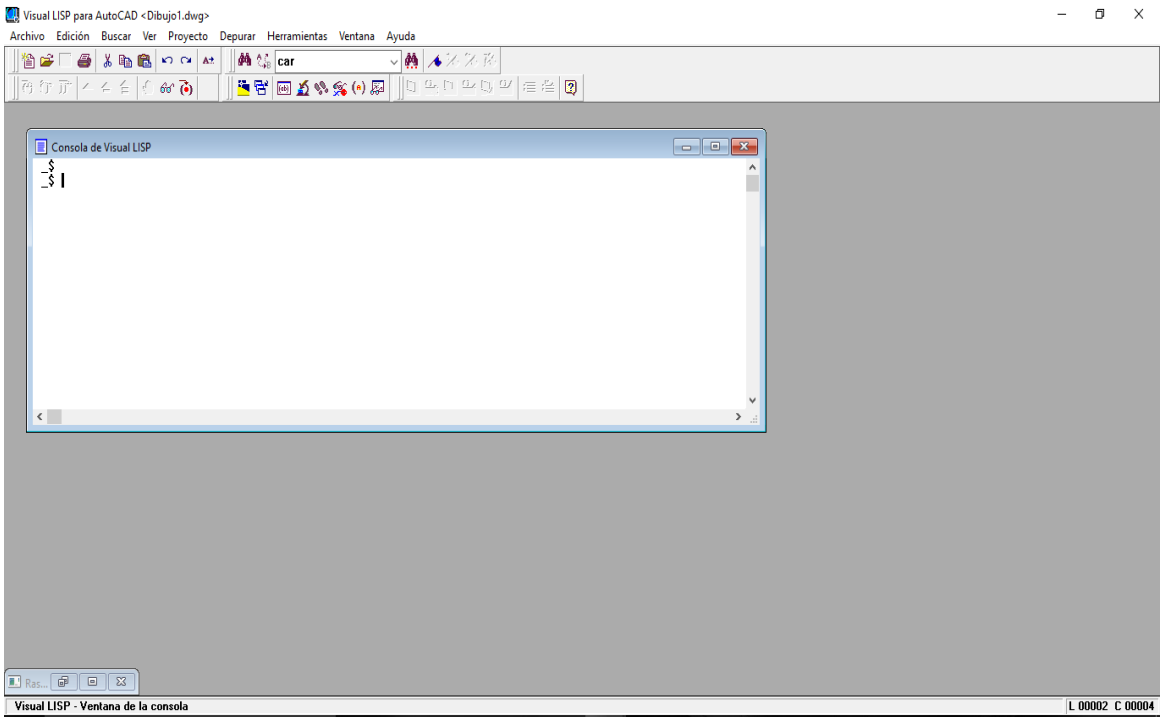


Figura 3-5. Carga de programa informático Modo 3

3.3 Definir una función/rutina/programa

En AutoLISP, una función es directamente un programa, pues su evaluación ofrece un resultado una vez cargado el archivo que contiene su definición. Así, un archivo .LSP puede contener muchos programa, según el número de funciones definidas en él. Para evaluarlas, no es necesario volver al archivo que las contiene, basta con cargarlo una sola vez.

Todas las funciones tienen esta estructura:

(DEFUN <simb> <lista argum> <expr>...)

- (DEFUN) se utiliza precisamente para definir funciones o programas de AutoLISP.
- <simb>, que es el símbolo o nombre de la función a definir. Conviene que los nombres de símbolos contengan como máximo seis caracteres, por razones de espacio ocupado en la memoria.
- Lista de argumentos (al ser una lista debe ir entre paréntesis), que puede estar vacía o contener varios argumentos y, opcionalmente, una barra inclinada y los nombres de uno o más símbolos locales de la función. Los argumentos o símbolos globales son variables que se almacenan en la memoria y permanecen en ella, lo mismo que los nombres de funciones definidas, de forma que pueden ser utilizados por otros programas o funciones de AutoLISP.

Para extraer el valor actual de esas variables, basta con introducir desde la Línea de Comando un signo "!" seguido del símbolo o nombre de la variable.

Los símbolos locales son opcionales. La barra inclinada debe estar separada del primer símbolo local y, si lo hubiera, del último global por, al menos, un espacio en blanco. Los símbolos locales se almacenan en la memoria solo temporalmente, mientras la función definida está en curso. En cuanto termina, desaparecen de la memoria, por lo que no pueden ser utilizados por otros programas o funciones. Se emplean cuando se necesitan, únicamente, dentro de la función definida, evitando que ocupen memoria inútilmente.

Si el símbolo local se encontrara ya creado antes de ser utilizado en la función definida, recuperará el valor que tenía al principio una vez terminada la función. Si no se especifican como locales al definir la función, todos los símbolos creados con SETQ dentro de ella se consideran como globales.

El último elemento de la función definidora DEFUN es la expresión en AutoLISP, que va a servir de definición de <simb>. Esta expresión puede ser tan compleja como se quiera, ejecutando otras funciones definidas, cargando archivos .LSP, etc.

Ejemplo:

Command: (*DEFUN c: seno (x) (SETQ xr (* PI (/ x 18Ø.Ø))) (SETQ s (SIN xr)))*

La función definida "seno" utiliza una variable global que es "x". Lo que hace esa función es crear una variable llamada "xr", que transforma el valor del ángulo "x" dado en grados sexagesimales a radianes. Para ello, divide "x" entre 18Ø.Ø y multiplica ese resultado por PI. El valor obtenido en radianes se almacena en "xr".

A continuación, se crea una nueva variable llamada "s". SIN es el comando de AutoLISP que calcula el seno de un ángulo expresado en radianes (en este caso, el valor contenido en "xr"), y el valor de ese seno, que es el resultado final que se persigue, se almacena en la variable "s".

Las funciones definidas por el usuario mediante (DEFUN) pueden integrarse en AutoCAD como una orden más. Para ello, deben tener un nombre precedido por C: y comprender una lista de variables globales vacía, aunque pueden tener variables locales.

En el **Anexo1** de este trabajo, explico todo el entorno de AutoLISP y lo amplío con la explicación de las funciones más importantes a la hora de programar con AutoLISP.

3.4 Ejemplos de rutinas.

Para entender cómo funciona un lenguaje de programación, la mejor manera es explicar una serie de rutinas sencillas para ver los procedimientos que siguen dicho lenguaje de programación.

Seguidamente, programaré y explicaré tres rutinas para ofrecer una introducción a la programación de AutoCAD mediante AutoLISP.

3.4.1 Rutina para crear polígonos.

Esta rutina realiza polígonos de cuatro lados. La rutina necesita que selecciones en la ventana de trabajo la posición de los cuatro vértices, mediante la función Getpoint, después, traza las líneas entre esos cuatro puntos.

```
( defun c: poligonos ()
 (setq pt1 (getpoint "dime un punto:"))
(setq pt2 (getpoint "dime otro punto:"))
(setq pt3 ( getpoint "dime otro punto:"))
(setq pt4 (getpoint "dime otro punto:"))
( command "_line" pt1  pt2  pt3  pt4 pt1 "")
)
```

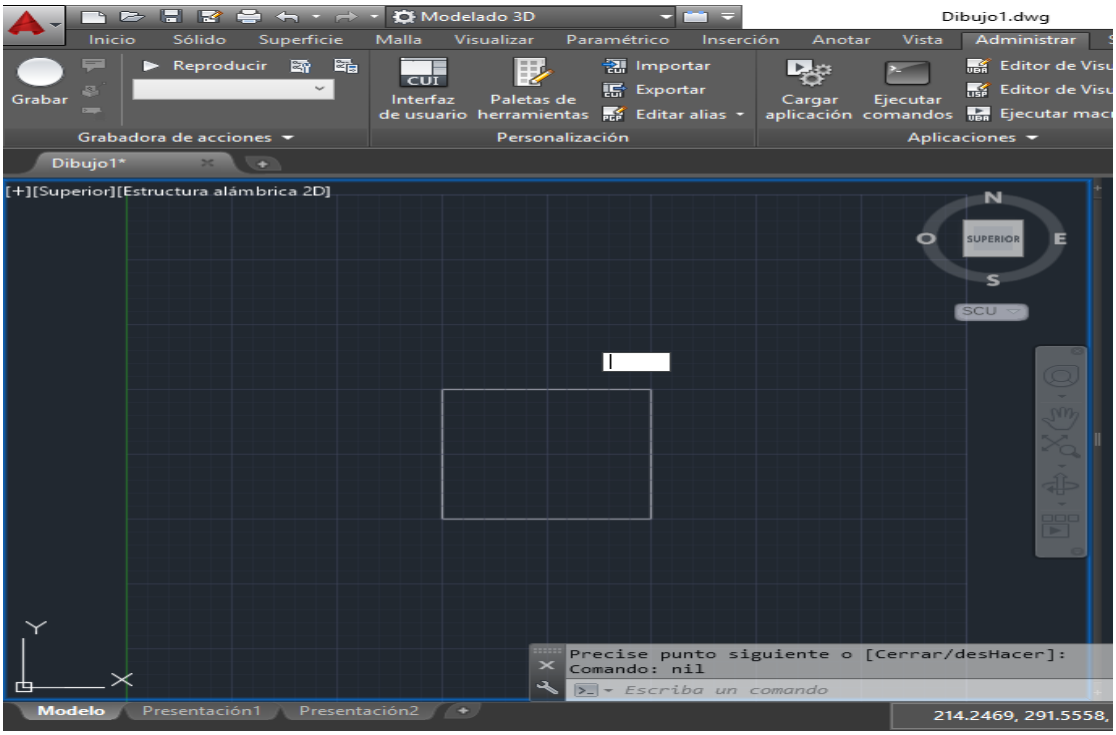


Figura 3-6 Rutina para crear polígonos

3.4.2 Rutina para crear círculos concéntricos.

Esta rutina realiza círculos concéntricos cada vez mayores hasta que el radio del último círculo supera un radio determinado.

Primero, guardamos en variables la posición del centro y en radio inicial y, después, el programa, mientras ese radio sea menor de 50, realiza círculos y añade 5 unidades más al radio hasta que llega a 50 y deja de realizar círculos.

```
(defun c:circulobucle:()
 (setq a (getpoint "dame un centro:"))
(setq b (getint "dame un radio:"))
 (if (< b 50)
(while (< b 50 )
 (setq b ( + b 5 ))
(command "_circle" a b )
)
)
)
```

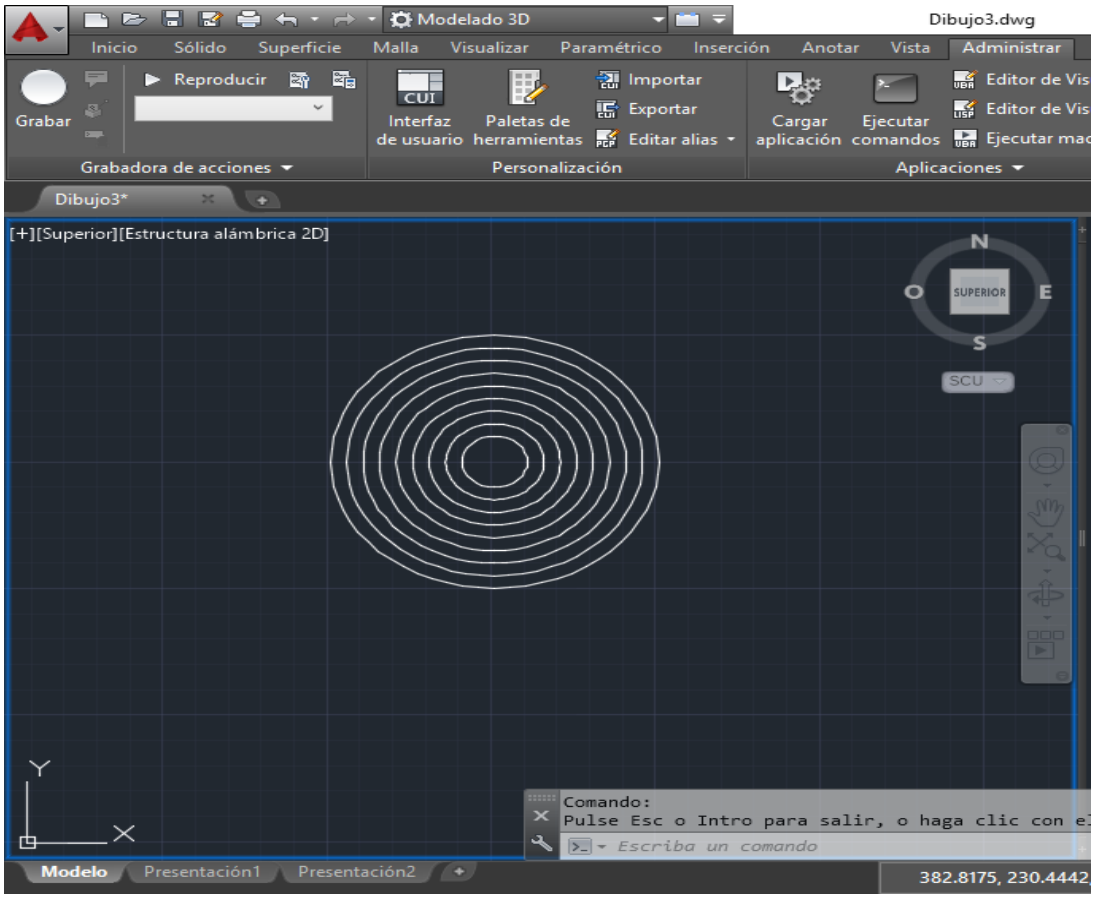


Figura 3-7. Rutina para crear círculos concéntricos

3.4.3 Rutina para crear polígonos en varios puntos a la vez.

En esta rutina, primero, necesitamos indicar, mediante Getpoint, la posición de los centros de los polígonos que, después, dibujará.

Después, mediante List creamos una lista con las coordenadas x e y de los distintos centros.

Por último, mediante un While programamos que, mientras la longitud de la lista sea mayor que cero, realizará el siguiente proceso, que es guardar una variable punto con las primeras coordenadas y, mediante el comando polígono, dibujar el polígono y, después, renombrar la variable listacentros eliminando las coordenadas que hemos utilizado.

```
(defun C:pentagonosmultiples:()
(initget 1)
(setq pt1 (getpoint "dame punto:")) (terpri)
(setq pt2 (getpoint "dame punto:")) (terpri)
(setq pt3 (getpoint "dame punto:")) (terpri)
(setq pt4 (getpoint "dame punto:")) (terpri)
(setq pt5 (getpoint "dame punto:")) (terpri)
(setq listacentros (list (car pt1) (car (cdr pt1)) (car pt2) (car(cdr pt2)) (car pt3) (car (cdr pt3)) (car pt4) (car (cdr pt4)) (car pt5) (car(cdr pt5))) )
(initget 5)
(setq diam (getint "dame radio:"))
(while (> (length listacentros) 0)
(setq punto (list (car listacentros) (car (cdr listacentros))))
(command "_polygon" "5" punto "I" diam "")
(setq listacentros (cdr (cdr listacentros)))
)
)
```

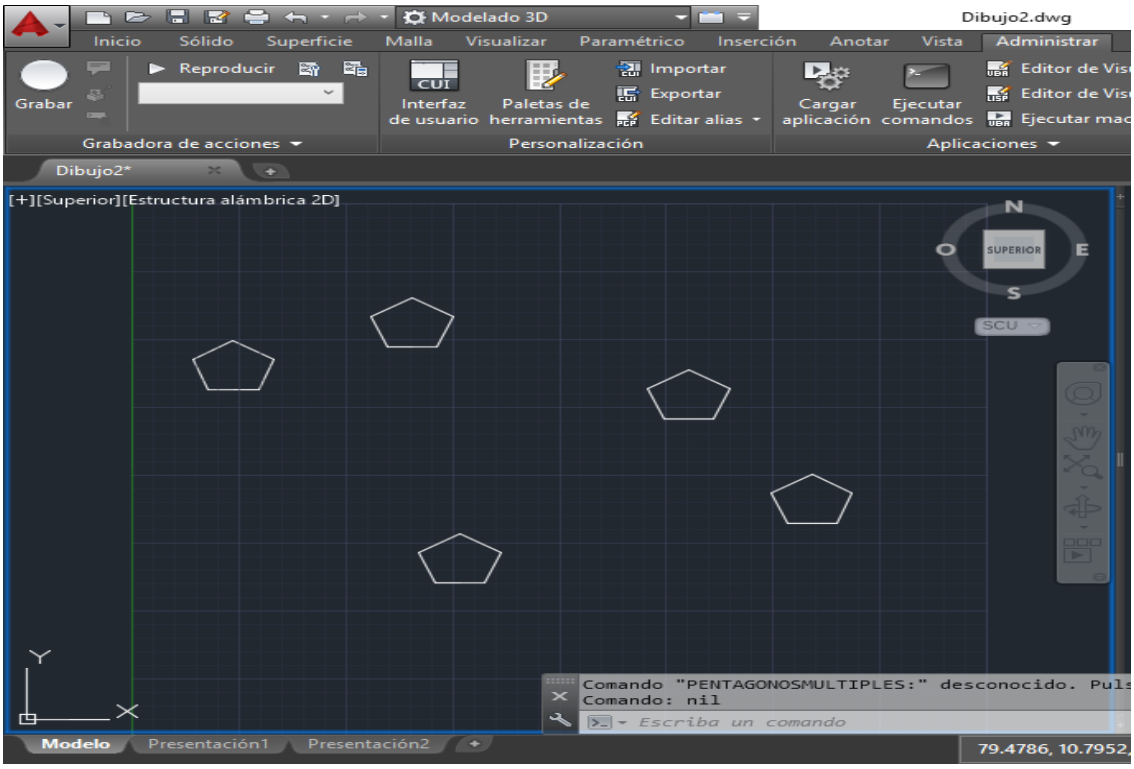


Figura 3-8 Rutina para crear varios polígonos simultáneos

4 Elementos roscados.

4.1 Introducción y definiciones

El estudio de los elementos de unión roscados es de vital importancia, pues permiten el fácil montaje y desmontaje de piezas o elementos de máquinas, facilitando, así, el mantenimiento de los sistemas industriales entre los que se encuentran, principalmente, los sectores automotrices y de la construcción de maquinaria en general.

Los tornillos son elementos de máquinas que se utilizan para la sujeción de dos o más componentes.

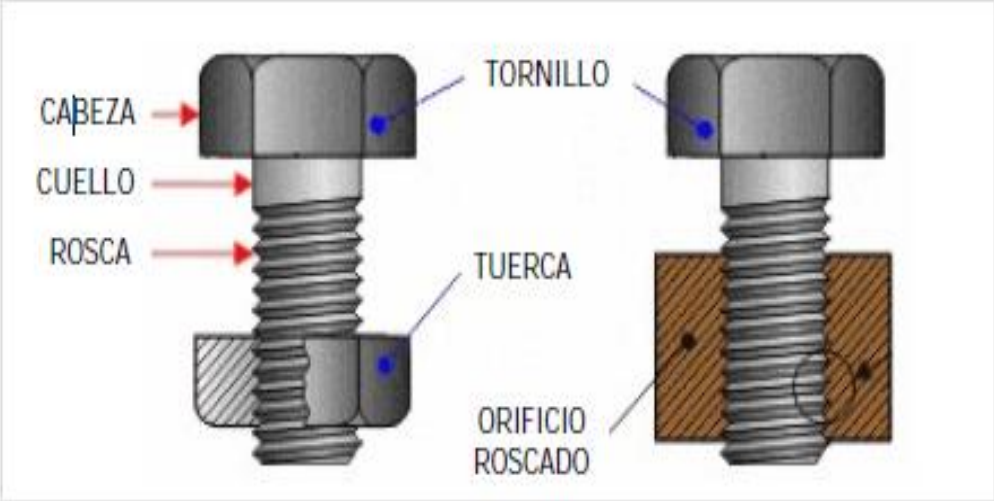


Figura 4-1. Esquema del tornillo

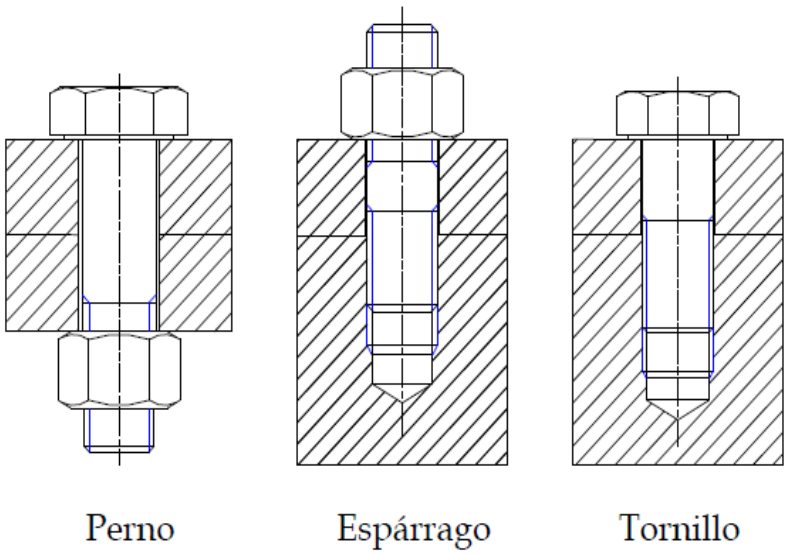


Figura 4-2. Representación esquemática de tornillos

Definiciones:

- Tornillo hexagonal: es un dispositivo de fijación mecánico con la cabeza en forma de hexágono, roscado exteriormente, lo que permite insertarse en agujeros previamente roscados en las piezas.
- Tuerca: es un elemento roscado internamente, que se utiliza para unir piezas con agujeros pasantes, mediante el uso de otros elementos roscados externamente.
- Rosca: es una serie de filetes (picos y valles), helicoidales de sección uniforme, formados en la superficie de un cilindro.
- Filete: es un hilo en forma de espiral de la rosca de los elementos roscados.
- Diámetro nominal: es el diámetro exterior o mayor de la rosca. Se utiliza comercialmente para la identificación de los elementos de tornillería.
- Diámetro de raíz: es el diámetro interior o menor de la rosca.
- Diámetro primitivo: es el diámetro promedio entre los diámetros nominales y de raíz.
- Cuerpo: es la porción no roscada de un tornillo.
- Cabeza: es la forma limitada dimensionalmente llevada a efecto en uno de los extremos del tornillos, cumpliendo la función de proveer una superficie de apoyo y permitiendo, además, el acople con herramientas.
- Altura de la cabeza o de la tuerca: es la distancia comprendida entre la parte superior de la cabeza del tornillo (o tope de la tuerca), hasta la superficie de contacto o apoyo, medida paralelamente al eje del tornillo (o de la tuerca).
- Arandela estampada de cabeza o de tuerca: es una superficie circular en relieve estampada en la superficie de contacto o apoyo, de la cabeza o de la tuerca.
- Entrecara de la cabeza o de la tuerca: es la distancia medida perpendicularmente al eje del tornillo (o de la tuerca) a través de los lados opuestos.
- Entrearistas de la cabeza (o de la tuerca) : es la distancia medida perpendicularmente al eje del tornillo desde la intercepción de los lados consecutivos de la cabeza (o de la tuerca), hasta la intercepción opuesta situada a 180º de la primera.
- Empalme: son los puntos de unión entre la cabeza y el cuerpo del tornillo.
- Radio de empalme: es el radio que origina la curvatura de unión entre el cuerpo y la cabeza del tornillo.
- Vástago: es la porción comprendida ente la superficie de apoyo de la cabeza y el extremo del tornillo.
- Longitud: es la distancia medida sobre los ejes del tornillo, desde la superficie de apoyo de la cabeza hasta el extremo.
- Longitud de la rosca: es la distancia medida paralelamente al eje del tornillo desde su extremo hasta el último filete completo de la rosca

- Paso: es la distancia axial entre puntos correspondientes de dos filetes (o hilos) adyacentes de una rosca.
- Hilos por pulgada: es la cantidad de filetes completos de la rosca contenidos en una pulgada. Su inverso es igual al paso.
- Angulo de rosca: es el ángulo formado por dos flancos contiguos.
- Rosca a derecha y a izquierda: son las roscas que penetran girando a derecha y a izquierda respectivamente.
- Avance: es la distancia axial que recorre un punto de un filete, cuando el elemento roscado da una vuelta completa.
- Rosca sencilla: es la rosca en la que el avance es igual al paso.
- Rosca múltiple: es la rosca en la que el avance es múltiplo del paso (2,3…

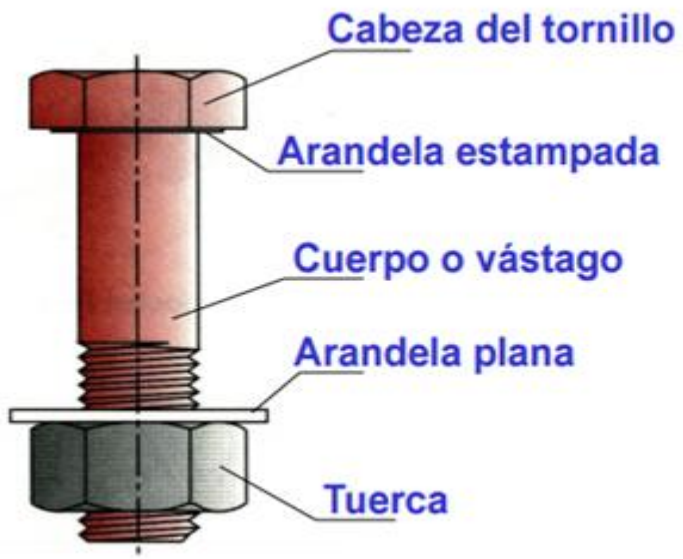


Figura 4-3. Partes de un tornillo

La rosca consiste en un filete helicoidal de varias espiras conformado sobre una superficie cilíndrica, cuyas formas y dimensiones permiten que el filete de otras roscas se ajuste a la ranura que forma el mismo.

Los tipos de rosca, mayormente utilizados, corresponden a la Rosca Unificada y a la Rosca Métrica.

- Rosca Métrica: Esta rosca es la del Sistema Internacional SI y posee una rosca simétrica de 60º, un entalle redondeado en la raíz de una rosca del tipo externo y un diámetro menor más grande en las roscas externas e internas. Este perfil se recomienda cuando se requiere elevada resistencia a la fatiga, existiendo en las series de Paso Basto y Paso Fino.

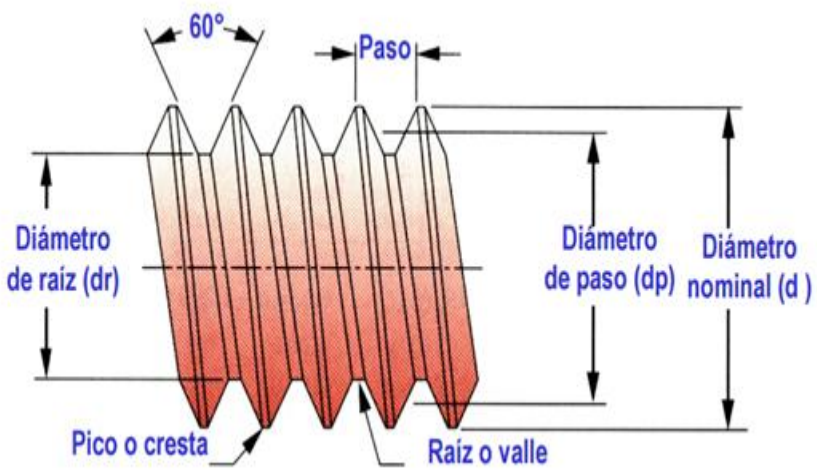


Figura 4-4. Partes de una rosca

4.2 Representación convencional de roscas

La norma UNE-EN-ISO 6410-1:1996: Dibujos técnicos. Roscas y piezas roscadas. Parte 1: Convenios generales. (ISO 6410-1:1993), define los convenios generales de representación de roscas.

En la actualidad, hay un sistema rápido y sencillo de representar roscas que consiste en dibujar las generatrices del vástago como si no estuviese roscado, indicando la rosca por dos líneas paralelas al contorno de espesor fino.

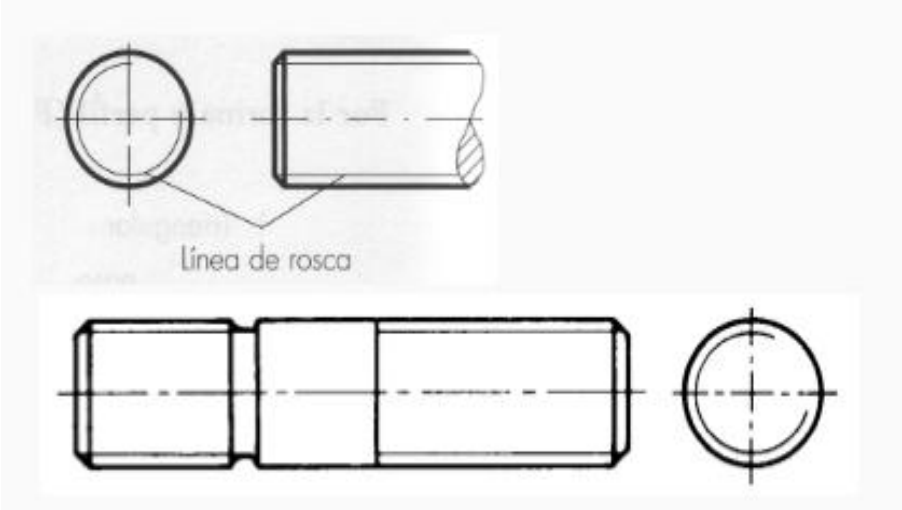


Figura 4-5. Representación convencional de tornillos

En tuercas y cabezas de tornillo, se dibujan las aristas del bisel como arcos de circunferencia.

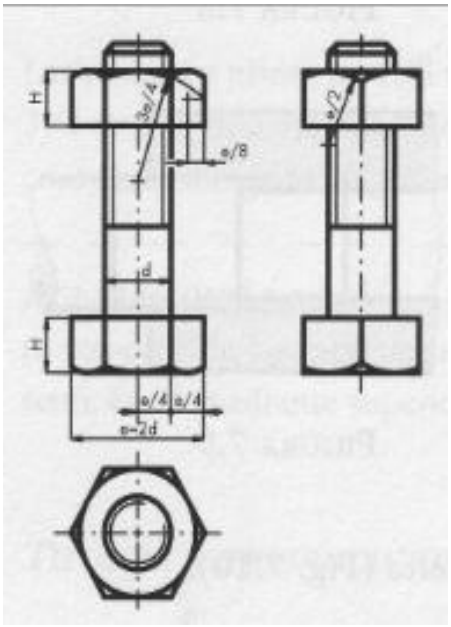


Figura 4-6 .Representación convencional de tornillos

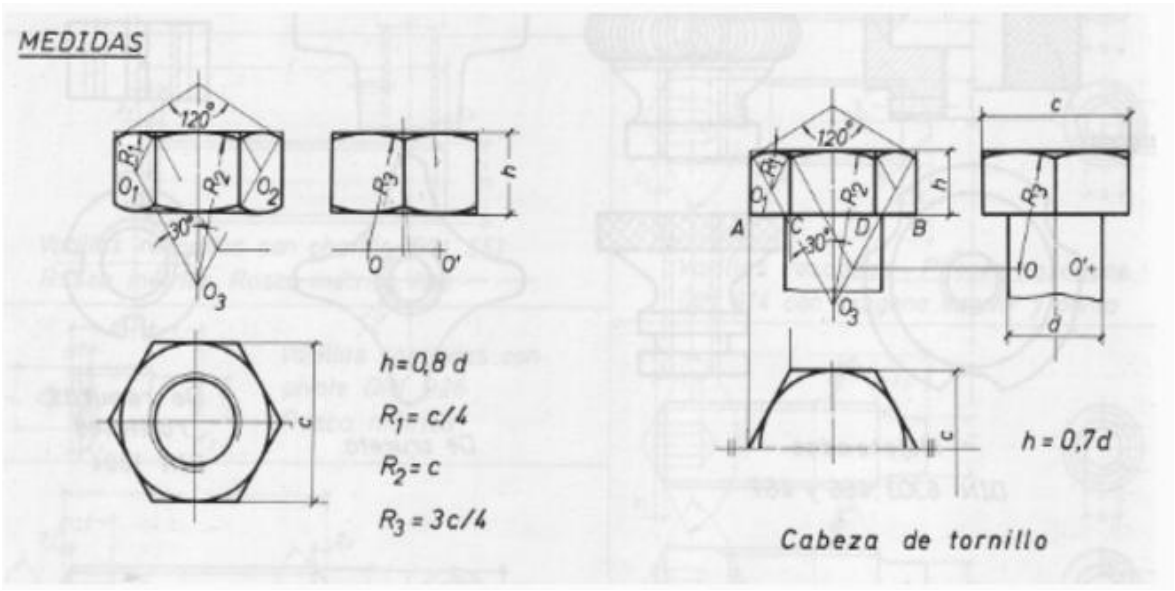
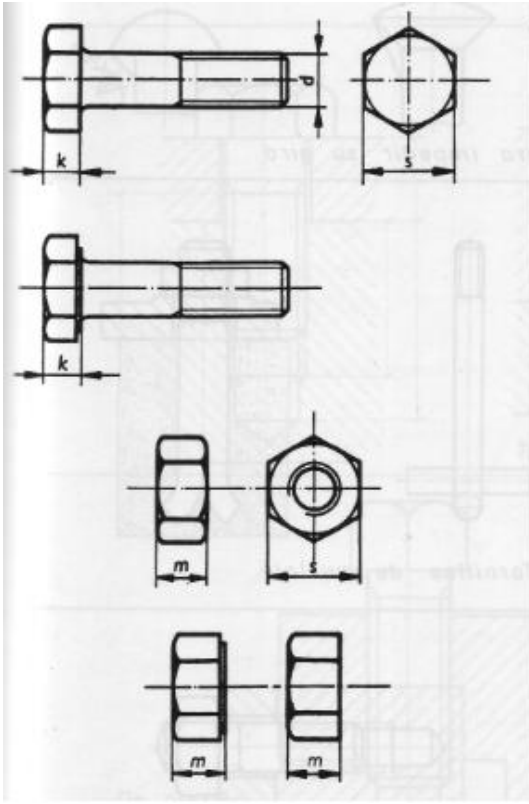


Figura 4-7. Representación convencional de tornillos

Entrecaras, altura de cabeza y altura de tuerca. UNE 17052-75. Esta norma establece las medidas de las distancias entre caras de las cabezas hexagonales, así como las alturas de las tuercas que se emplean en los medios de fijación.



Medidas en mm.

Diámetro de rosca d	Entrecaras del hexágono		Altura de la cabeza k	Altura de la tuerca m
	Normal	Reducida		
1,6	3,2	–	1,1	1,3
2	4	–	1,4	1,6
2,5	5	–	1,7	2
3	5,5	–	2	2,4
4	7	–	2,8	3,2
5	8	–	3,5	4
6	10	–	4	5
7	11	–	5	5,5
8	13	12	5,5	6,5
10	17	14	7	8
12	19	17	8	10
14	22	19	9	11
16	24	22	10	13
18	27	24	12	15

Figura 4-8.Representación convencional de tornillos

En la representación simplificada, se prescinde de las aristas del bisel de las tuercas y cabeza del tornillo, así como del extremo del tornillo. Se recomienda esta representación simplificada para diámetros nominales de roscas menores de 6 mm.

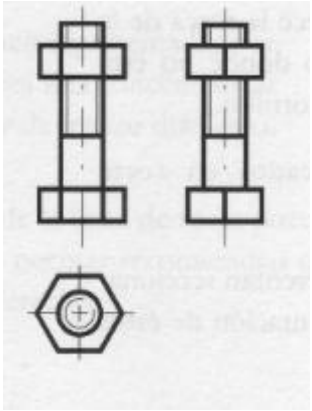


Figura 4-9.Representación convencional de tornillos

4.3 Comparativa entre la realización manual y la realización automática de elementos roscados.

El principal objetivo de este estudio es ver las ventajas que supone realizar el dibujo de elementos roscados, mediante una rutina en AutoLISP, frente a su realización de manera manual.

Para ello, primero tenemos que ver que, para realizar manualmente un elemento roscado, necesitamos seguir una serie de pasos, los cuales están realizados en el **Anexo 2.**

Al realizar, mediante los dos procedimientos, el elemento roscado obtenemos estos resultados:

- Tiempo de realización manual: **aproximadamente 15 minutos**.
- Tiempo de realización automática: **menor de 5 minutos.**

Observamos que la reducción de tiempo y trabajo es muy grande, por lo que supone un gran paso para mejorar la eficiencia en esta clase de trabajos.

4.4 Rutina para elaborar un tornillo de cabeza hexagonal.

En el **Anexo 3,** está desarrollada toda la rutina completa para la realización de elementos roscados. Aquí, solo muestro un fragmento que sirve como ejemplo.

```
(defun C:Tornillo ()

        (setq PI (getpoint "\Punto de insercion del tornillo: "))
        (setq D (getint "\nDistanca entre aristas del exagono de la cabeza: "))
        (setq H (getint "\nAltura de la cabeza del tornillo: "))
        (setq DIAMROS (getint "\nDiamtro de la rosca: "))
        (setq LONGROS (getint "\nLongitud de la rosca: "))
        (setq LONGTOR (getint "\nLongitud del tornillo: "))
        (setq PIx (car PI))
        (setq PIy (car (cdr PI)))
        (setq P1 (list (- PIx (/ D 2)) (+ PIy (- H (* 0.0428932 D)))))
        (setq P2 (list (- PIx (/ D 4)) (+ PIy (- H (* 0.0428932 D)))))
        (setq P3 (list (+ PIx (/ D 4)) (+ PIy (- H (* 0.0428932 D)))))
        (setq P4 (list (+ PIx (/ D 2)) (+ PIy (- H (* 0.0428932 D)))))
        (setq PC1 (list (- PIx (* 0.375 D)) (+ PIy H)))
        (setq PC2 (list PIx (+ PIy H)))
        (setq PC3 (list (+ PIx (* 0.375 D)) (+ PIy H)))
        (setq PB1 (list (- PIx (/ D 2)) PIy))
        (setq PB2 (list (- PIx (/ D 4)) PIy))
        (setq PB3 (list (+ PIx (/ D 4)) PIy))
        (setq PB4 (list (+ PIx (/ D 2)) PIy))
        (setq PT1 (list (- PIx (/ DIAMROS 2)) PIy))
```

Figura 4-10.Fragmento de la rutina para elementos roscados

4.5 Explicación de la rutina

1. Para dibujar el esquema de un tornillo de cabeza hexagonal, lo primero que hacemos es introducir un punto que nos servirá de referencia para la construcción del tornillo. A este punto lo llamo PI y lo introduzco mediante un Getpoint.

2. Después, guardo las coordenadas X e Y del punto PI (PIx y PIy), que me sirven para obtener los otros puntos.

3. El siguiente paso es guardar en variables los valores que configuran la construcción del tornillo, como son:

- La distancia entre aristas del hexágono de la cabeza
- Diámetro de la rosca
- Longitud de la rosca
- Longitud del tornillo

4. Mediante la relación entre las medidas de un tornillo ya normalizadas obtenemos los demás puntos.

Ejemplo:

El punto PI es el punto central de la base de la cabeza. Para obtener el punto simétrico del techo de la cabeza será un punto que tenga la misma coordenada X que PI y la coordenada Y del punto será la de PI más la altura de la cabeza.

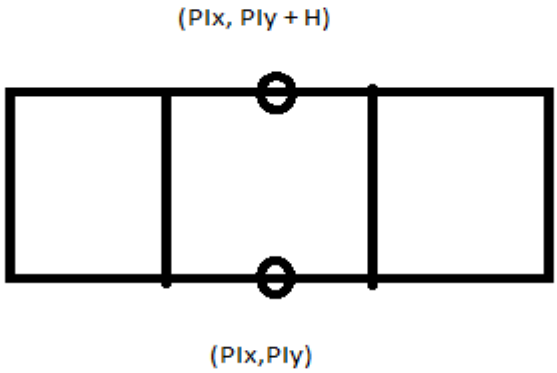


Figura 4-11. Cabeza hexagonal

5. Por último y tras obtener todos los puntos significativos, trazo las líneas mediante el comando “line” entre los distintos puntos.

4.6 Visualización en AutoCAD

Después de desarrollar la rutina para la realización de tornillo, cargarla y ejecutarla en AutoCAD, obtenemos la representación del tornillo hexagonal con total precisión y, como se trata de un archivo de AutoCAD, podemos realizar todas las modificaciones que necesitemos antes de su impresión o guardado en formato pdf y similares.

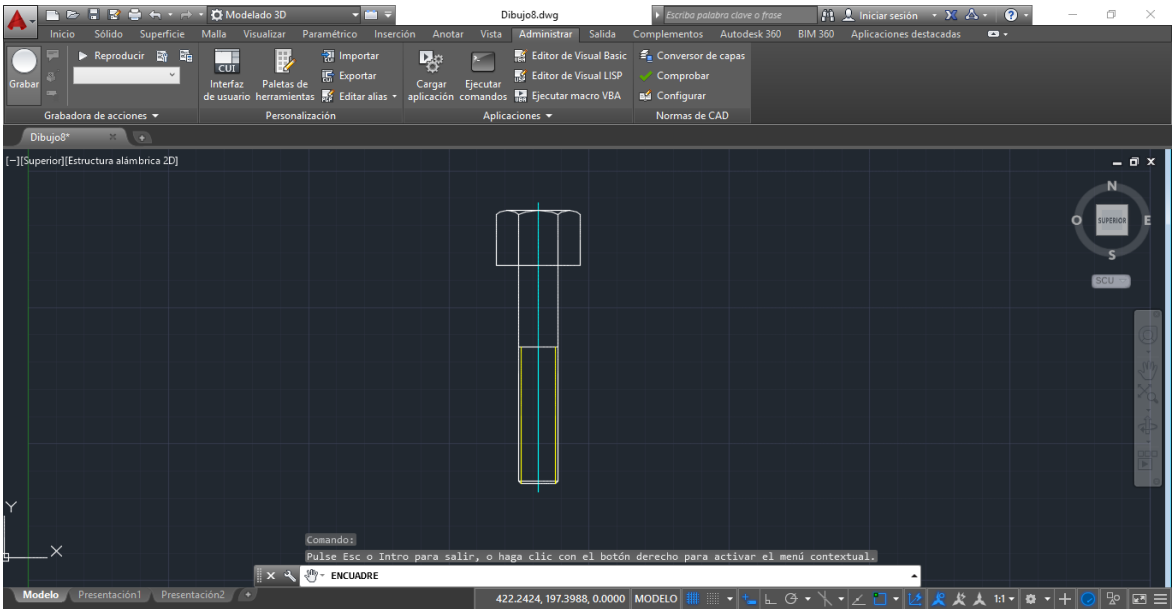


Figura 4-12. Visualización del tornillo en AutoCAD

5 Pozo Minero

5.1 Definiciones y partes de una explotación minera

- Explotación de la mina: todos los trabajos que se realicen en el interior de una mina como: excavaciones, túneles, galerías, etc.
- Rumbo: dirección o rumbo de una línea horizontal del estrato.
- Pendiente o inclinación: ángulo que forma el estrato con el plano horizontal.
- Techo: plano superior del estrato.
- Muro: plano inferior del estrato.
- Pozo de extracción: pozo vertical o, aproximadamente, vertical abierto desde la superficie para la explotación de la mina.
- Brocal o marco del exterior: entrada entibada y horizontal al pozo de extracción.
- Pozo inclinado: sigue la dirección del estrato con su mismo ángulo de pendiente.

- Túnel: galería subterránea, horizontal o casi horizontal, y que sale a la superficie por uno sus extremos.
- Galería transversal, cortavena o socavón crucero: galería que atraviesa el estrato uniendo galerías.
- Galería de explotación: es la que está situada siempre en el mismo estrato marcando el rumbo o dirección del mismo.
- Chimenea, pozo ascendente entre galerías, coladero o contracielo: es una galería ascendente de abajo hacia arriba.
- Pozo ciego, pozo descendente o galería clavada: es una galería ascendente de abajo hacia arriba.
- Afloramiento: intersección de la vena o estrato con el terreno.

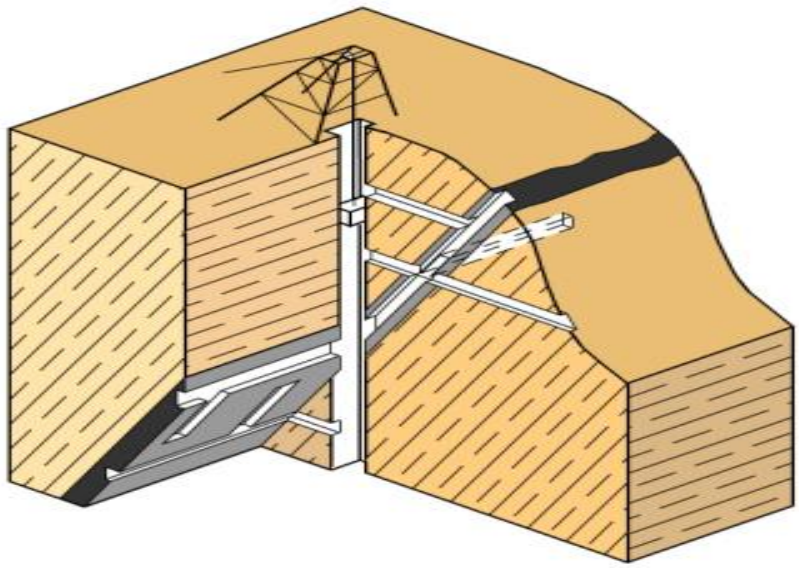


Figura 5-1. Esquema de explotación minera

5.2 Comparativa entre realización manual y automática de los pozos mineros

Como hemos visto en el apartado de los elementos roscados, la automatización de dibujos técnicos presenta ventajas frente a la ejecución manual. En este apartado, veremos las diferencias de resolver un problema de pozos mineros mediante los dos procedimientos.

Para entender mejor el funcionamiento del programa y la comparación de los dos procedimientos, tenemos que conocer los pasos que se siguen manualmente para su resolución, los cuales están explicados en el **Anexo 4.**

Al realizar, mediante los dos procedimientos, el pozo minero obtenemos estos resultados:

- Tiempo de realización manual: **aproximadamente 25 minutos.**
- Tiempo de realización automática: **menor de 5 minutos.**

Observamos que la reducción de tiempo y trabajo es muy grande, por lo que supone un gran paso para mejorar la eficiencia en esta clase de trabajos.

5.3 Rutina para explotación minera

En el **Anexo 5,** está desarrollada toda la rutina completa para la realización de pozos mineros. Aquí, solo muestro un fragmento que sirve como ejemplo.

```
(defun c:pozosmineros ()

    (setq pa (getpoint "dime la ubicacion de la boca del pozo A:"))

                                    (setq pax (car pa))

                                    (setq pay (car(cdr pa)))

                                    (setq paz (caddr pa))

    (setq pendiente1 (getint "dime la pendiente del pozo A:"))

    (setq rumboa (getint "dime el rumbo del pozo a:") )

                                    (command "color" 3)

                (cond

                    (  ( and ( > rumboa 0) ( < rumboa 90 ))

                            (setq rumbo1 (/ (* rumboa PI) 180 ))

                            (setq perforacion1 (getint "dime la perforacion1:"))

                            (setq h (* perforacion1 (cos ( / ( * pendiente1 PI) 180))))

                            (setq z (* perforacion1 (sin ( / ( * pendiente1 PI) 180))))

                            (setq techo1 (list (+ pax (* h (cos rumbo1))) (+ pay (* h (sin rumbo1))) (- paz z)))

                                        (command "_line"  pa techo1 "")
```

Figura 5-2. Fragmento de la rutina para la resolución de pozos mineros

5.4 Explicación de la rutina

1. Indico las coordenadas/ubicación de la boca del pozo A (la primera perforación).
2. Guardo en variables separadas las coordenadas X, Y y Z para poder operar con ellas.
3. Le pido que me registre en una variable el valor de la pendiente del pozo A.

4. Le pido que me guarde en una variable independiente el valor del rumbo del pozo A.

*Este valor de los grados del rumbo se tiene que introducir en valor absoluto. Teniendo en cuenta que AutoCAD toma como ángulo cero el perteneciente al eje X de coordenadas y toma como sentido positivo de los ángulos el anti horario.

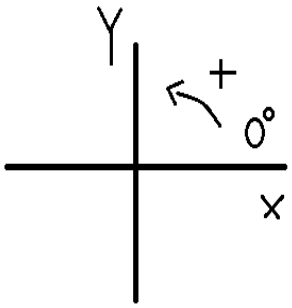


Figura 5-3. Sentido del ángulo

También, se tiene que introducir el valor teniendo en cuenta el sentido de la pendiente del pozo. Ejemplo:

Rumbo S27ºE y 56ºS

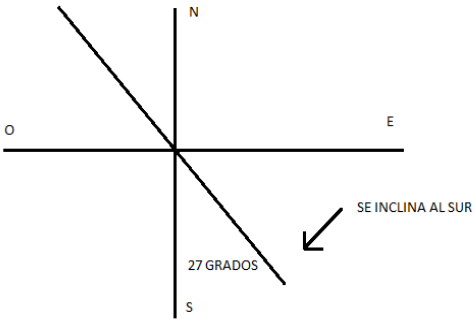


Figura 5-4. Sentido del rumbo

5. Ejecuto la orden color para cambiar el color del pozo A, que ejecuto en las siguientes órdenes.

6. Debido a que el pozo puede tomar diferentes orientaciones, lo que utilizo es un condicional para poder programar todas las opciones.

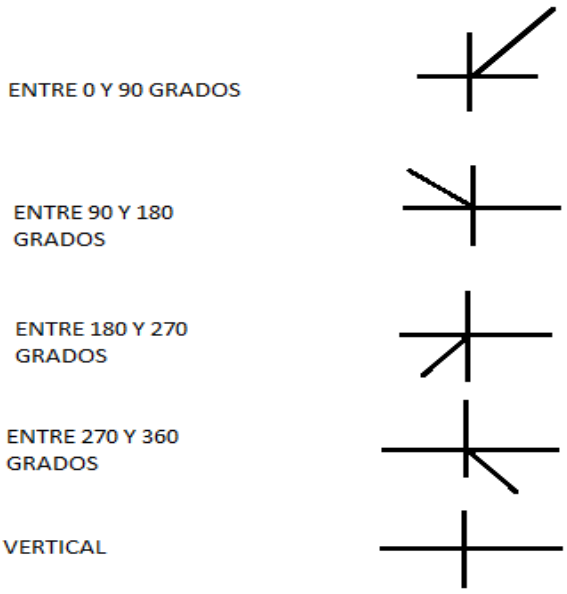


Figura 5-5. Orientaciones del los pozos

* Mediante un condicional, proceso las diferentes expresiones y ejecuto la que es cierta y, mediante un operador lógico IF, programo las diferentes expresiones para que el condicional evalúe la expresión cierta.

*** Para ejecutar los pozos verticales, se tiene que introducir el valor 1000 para especificar que el pozo es vertical.**

7. El siguiente paso es operar con la pendiente, el rumbo de los pozos y las perforaciones hasta llegar a los techos y muros para obtener dos puntos: uno del techo y otro del muro. Después, trazar líneas que atraviesen esos puntos para ver, gráficamente, la trayectoria del pozo. Este proceso se realiza mediante geometría.

8. Después de ejecutar los paréntesis, obtengo un punto del techo (techo1) y uno del muro (muro2).

9. Después de obtener techo1 y muro1, cambio el color para ejecutar el pozo B.

10. Para ejecutar el pozo B, primero, debemos situar gráficamente el pozo. Para situar el pozo B, lo hago tomando como referencia el pozo A. Es decir, guardo en variables separadas las diferencias en los 3 ejes que tiene la ubicación del pozo B respecto al pozo A.

* Los datos se introducen con estas condiciones:
  - 30 metros al Este (+ X) DistanciaX = +30
  - 20 metros al Sur (- Y) DistanciaY=- 20
  - 10 metros más bajo (-Z) DistanciaZ=- 10

11. Después, guardo en una variable la posición del pozo B, que tendrá como coordenadas las del pozo A más las diferencias antes guardadas.

12. Después, repito todo el proceso realizado con el pozo A hasta obtener dos puntos: uno del techo (techo2) y del muro (muro2).

13. Guardo en variables independientes las coordenadas de los cuatro puntos: techo1, muro1, techo2 y muro2.

14. Posteriormente, cambio otra vez el color para realizar los planos que conforman el techo y el muro.

15. Para realizar los planos del techo y del muro del estrato, primero, trazo con el Command "line" una línea del punto muro2 al muro1, la cual estará contenida en el muro, y guardo esa línea como un objeto en una variable.

16. Como consideramos los planos del techo y los muros paralelos, desplazamos la línea antes creada desde muro1 a techo2. Esta línea estará contenida en el plano techo.

17. Por las diferencias entre los puntos muro1 y muro2 que se conservan en la línea desplazada (se realiza un movimiento paralelo), obtengo las coordenadas de un tercer punto del techo (techo3) y guardo sus coordenadas en variables independientes para, después, trabajar con ellas.

18. La siguiente operación que realizo es la creación del plano del techo, mediante el comando "3dface", tomando como puntos del plano los puntos: techo1, techo2 y techo3. Posteriormente, guardo ese plano como un objeto en una variable.

19. Para obtener el plano del muro, ya que son paralelos, copio el plano del techo y lo desplazo del techo2 al muro1, obteniendo, así, el plano del muro.

20. Obtenido gráficamente el pozo minero, necesito calcular 3 datos:

- ***Potencia del pozo***
- ***Pendiente del pozo***
- ***Buzamiento del pozo***

21. Para obtener la potencia del pozo, hago un cambio de "scp", es decir, se cambian los ejes de referencia y se sitúan sobre el plano del techo. Al cambiar los ejes, las coordenadas de los puntos también cambian y como el techo y el muro son paralelos, guardo la coordenada Z de un punto del muro y esa es la potencia del pozo, ya que el techo está referenciado a Z=0. Después, deshago el cambio y restablezco las coordenadas universales con otro cambio de "scp".

22. Después, para obtener el buzamiento y la pendiente del pozo, necesito obtener una horizontal del plano del techo. Para ello, me ayudo de los operadores COND y AND para programar una rutina que me trace un plano que corte al plano del techo por el punto de cota intermedia y, como los puntos pueden variar, programo para todas las opciones y, cuando encuentra el correcto, traza un plano horizontal que atraviesa ese punto e interseca dichos planos obteniendo una recta horizontal de intersección.

23. Para poder ver la pendiente que tiene el plano necesito verlo proyectante, por lo que realizo otro cambio de "scp" y referencio el eje Z de coordenadas a la recta horizontal de intersección, quedando el plano proyectante. Guardo las coordenadas cambiadas en

variables y, mediante geometría, calculo el ángulo de la pendiente. Luego, deshago el cambio y restauro las coordenadas universales.

24. Por último, para averiguar el buzamiento mido el ángulo entre la recta horizontal de intersección y una recta del plano con dirección norte-sur.

5.5 Visualización en AutoCAD

Mediante AutoCAD, podemos observar gráficamente los pozos desarrollados, mediante la rutina de programación. AutoCAD, también, permite partir en varias ventanas de visualización el espacio de trabajo.

Así, cuando se ejecute la rutina para la realización de pozos podemos observar la planta, el alzado y una vista 3d de los pozos.

Por último, en una línea de comandos nos aparecen los resultados de los datos que necesitamos extraer como son el buzamiento, la pendiente y la potencia del pozo.

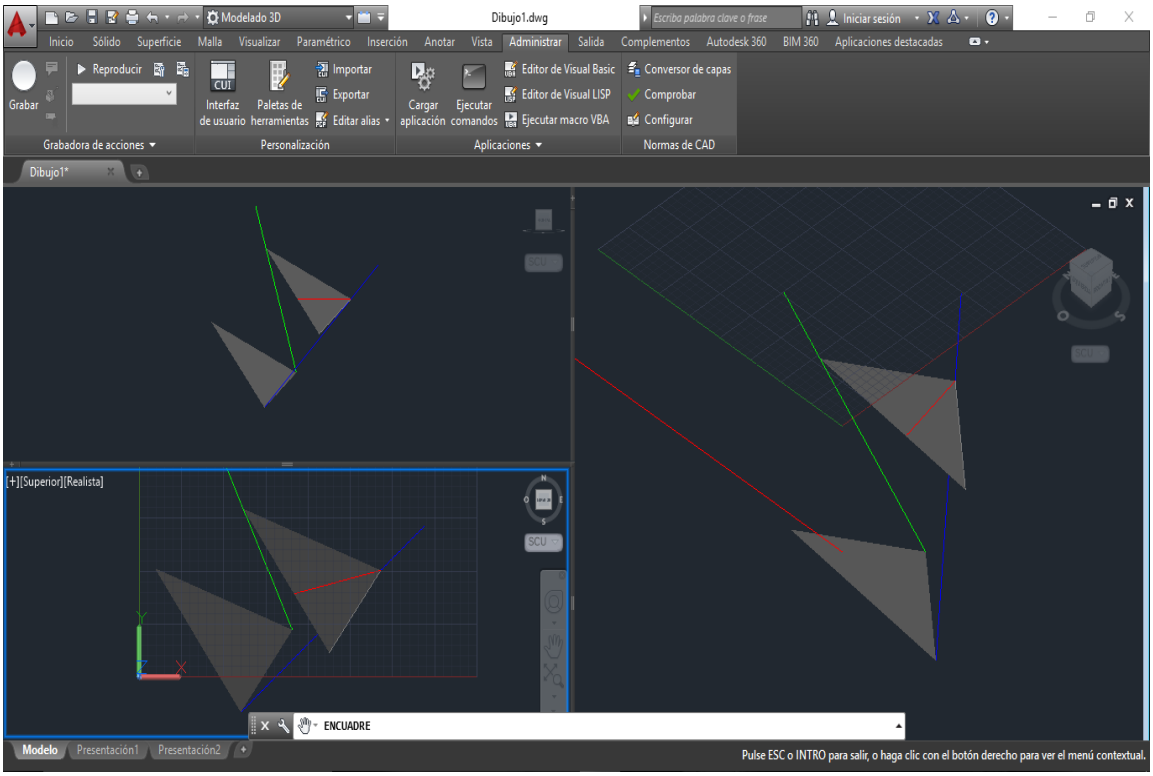


Figura 5-6. Visualización de los pozos AutoCAD nº1

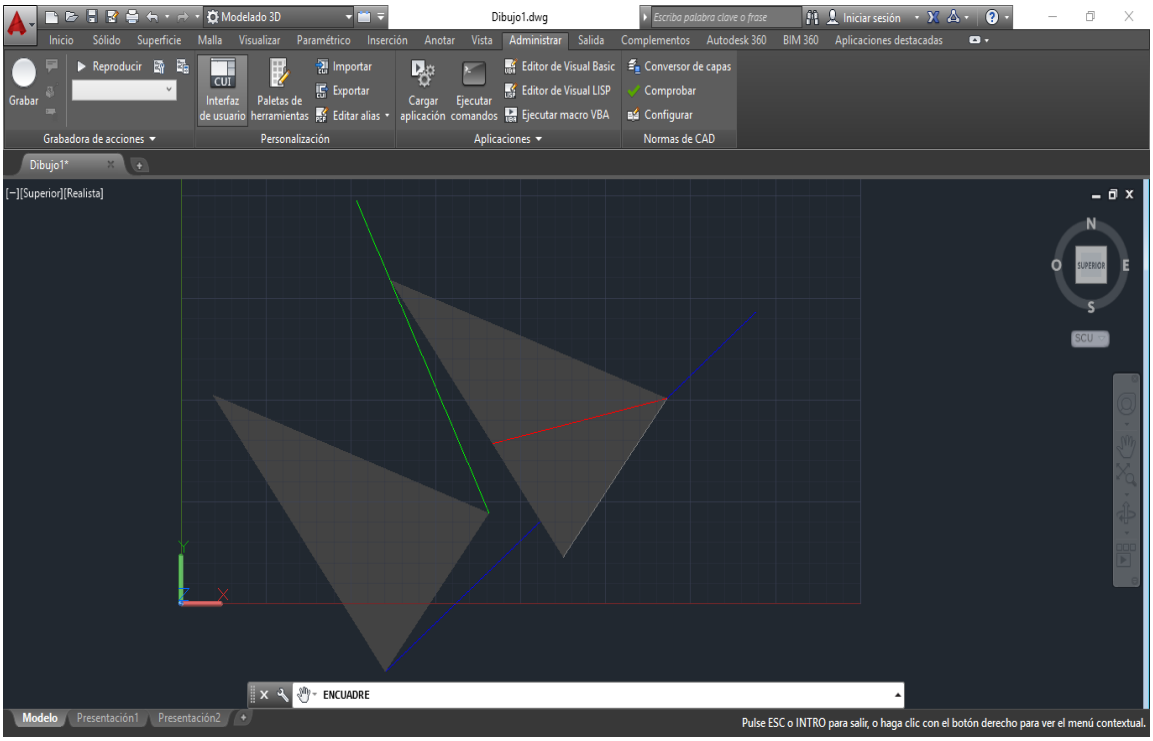


Figura 5-7. Visualización de los pozos AutoCAD Nº2

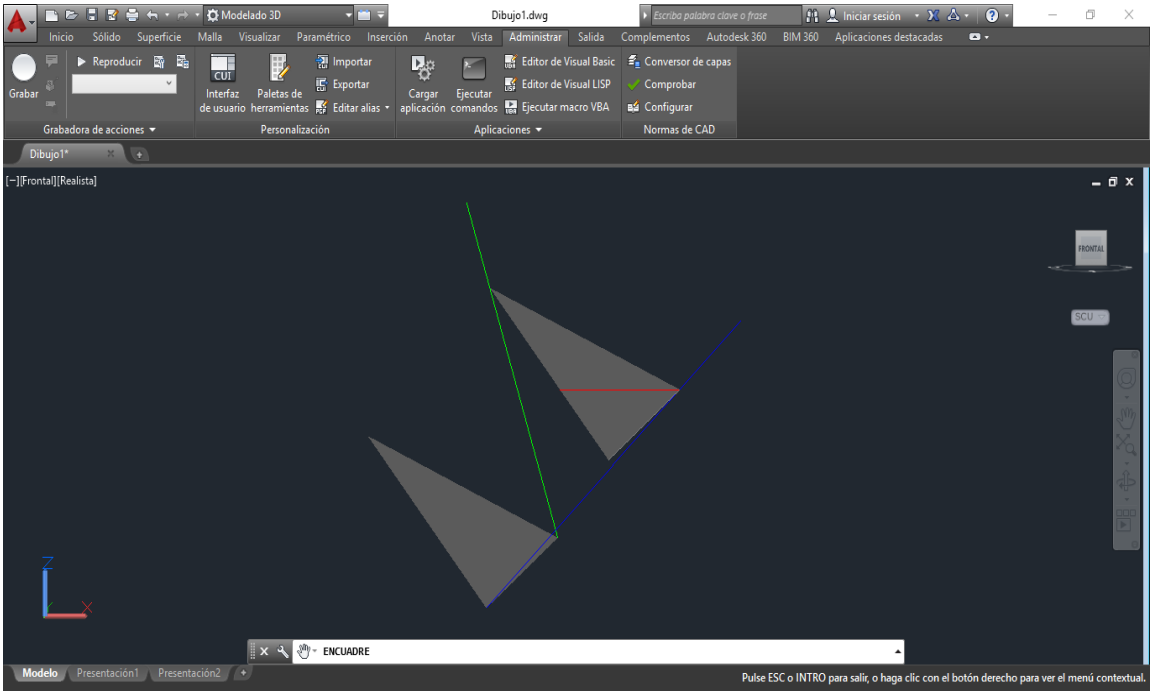


Figura 5-8. Visualización de los pozos AutoCAD nº 3

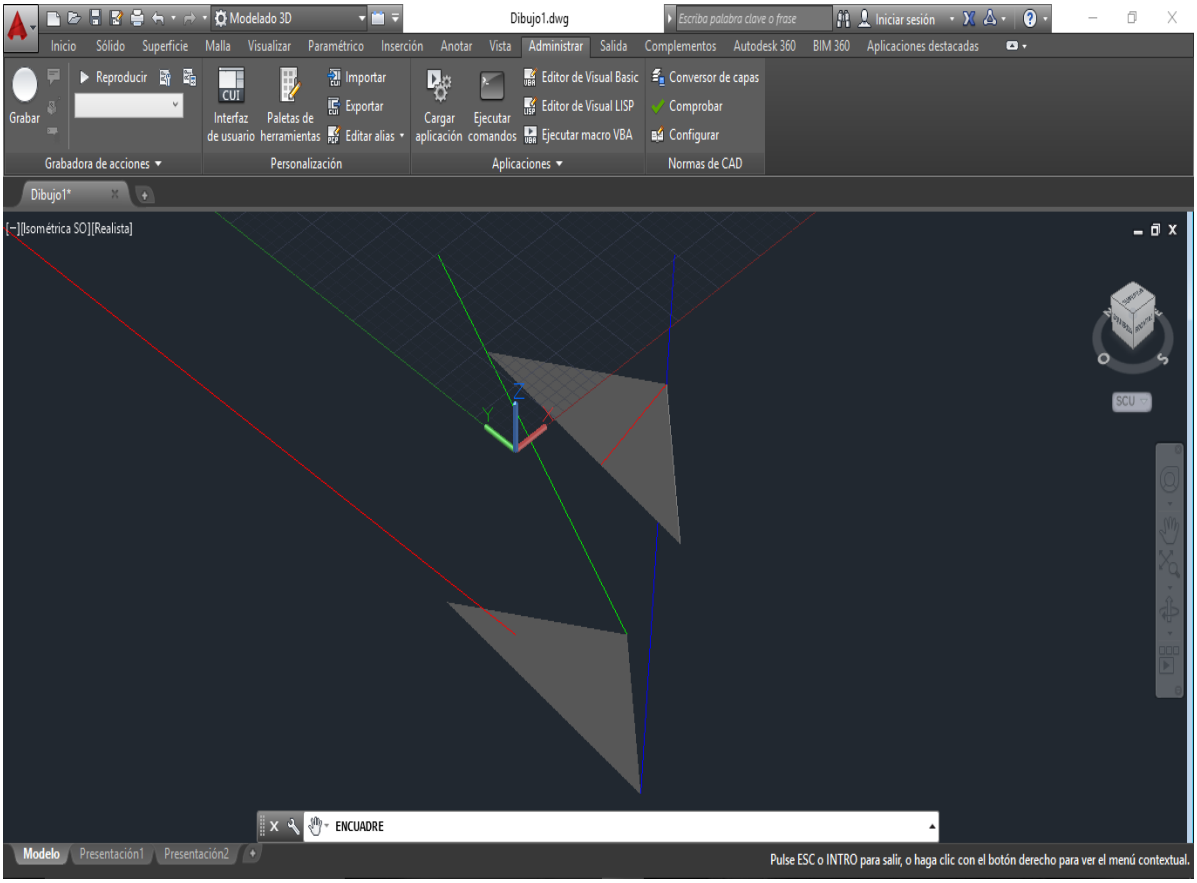


Figura 5-9. Visualización de los pozos AutoCAD nº4

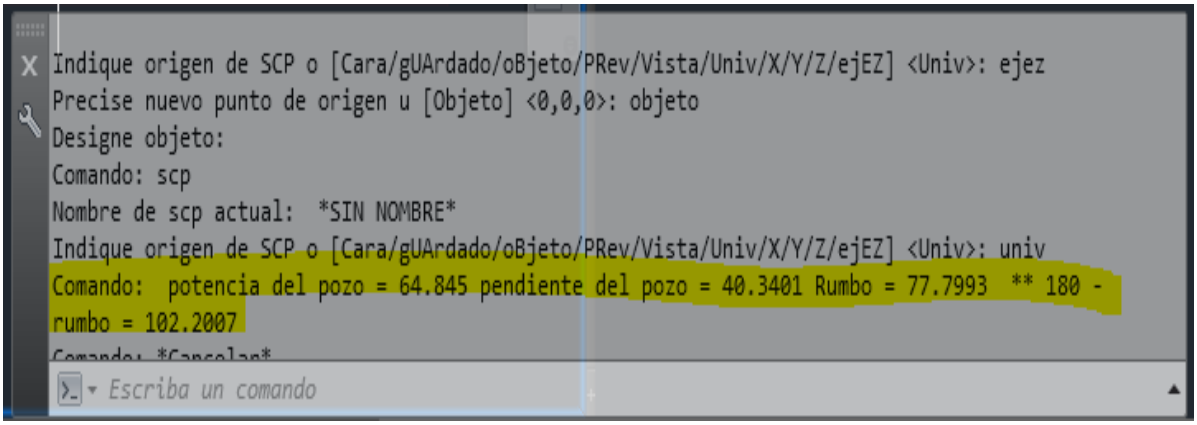


Figura 5-10. Datos obtenidos por el programa nº1

6 Conclusiones

Después de la realización de este trabajo, puedo sacar una serie de conclusiones que expongo a continuación:

- AutoCAD es una herramienta esencial para los ingenieros, la cual permite realizar cualquier tipo de dibujo, plano o mecanismo industrial.
- A pesar de ser un programa completo, presenta una serie de limitaciones a la hora de realizar diferentes dibujos que siguen un patrón y tenemos que realizar uno por uno, con el consiguiente gasto de tiempo y pérdida de productividad.
- Las limitaciones antes mencionadas se pueden solucionar mediante la combinación de la herramienta AutoCAD y los lenguajes de programación.
- Los lenguajes de programación amplían la capacidad de los programas de dibujo, pudiendo programar rutinas que faciliten la realización de esos dibujos e implementarlas en el programa de una manera sencilla.

7 Anexo 1: Entorno de AutoLISP.

7.1 AutoLISP. Conceptos previos y definiciones

7.1.1 Introducción

AutoCAD es un software de diseño asistido por computadora utilizado para dibujo 2D y modelado 3D, es decir, es un programa de dibujo técnico desarrollado por Autodesk para el uso de ingenieros, técnicos y otros profesionales de carreras de diseño.

7.1.2 Lenguaje de programación LISP

El Lisp (o LISP) es una familia de lenguajes de programación de computadora de tipo multiparadigma con una larga historia y una sintaxis completamente entre paréntesis.

Especificado, originalmente, en 1958 por John McCarthy y sus colaboradores en el Instituto Tecnológico de Massachusetts, el Lisp es el segundo lenguaje más antiguo de programación de alto nivel de extenso uso hoy en día, solamente el FORTRAN es más viejo.

El Lisp fue creado, originalmente, como una notación matemática práctica para los programas de computadora, basada en el cálculo lambda de Alonzo Church. Se convirtió rápidamente en el lenguaje de programación favorito en la investigación de la inteligencia artificial (AI). Como uno de los primeros lenguajes de programación, el Lisp fue pionero en muchas ideas en ciencias de la computación, incluyendo las estructuras de datos de árbol, el manejo de almacenamiento automático, tipos dinámicos y el compilador auto contenido.

Su nombre proviene de LISt Processing (Procesado de Listas), puesto que la base de su funcionamiento es el manejo de listas en vez de datos numéricos.

Una lista es un conjunto de símbolos que pueden ser nombres de variables, datos concretos numéricos o textuales, funciones definidas por el propio usuario, etc. El símbolo es, pues, la unidad básica con un contenido o un significado para el programa en LISP.

La lista es el mecanismo que junta una serie de símbolos y los evalúa, es decir, los procesa obteniendo un resultado. El lenguaje LISP procesa directamente las listas en cuanto se encuentran formadas y obtiene o "devuelve" el resultado de ese proceso.

Esta característica del manejo de listas otorga al LISP una gran versatilidad y le distingue de otros lenguajes de programación orientados a la manipulación de números.

Las ventajas que supone la utilización de un lenguaje basado en LISP para programar desde AutoCAD se podrían resumir en los siguientes puntos:

- Facilidad para manejar objetos heterogéneos: números, caracteres, funciones, entidades de dibujo, etcétera. Para LISP, basta representar cualquiera de esos objetos con un "símbolo" y no hay necesidad de definir previamente qué tipo de datos va a contener ese símbolo.
- Facilidad para la interacción en un proceso de dibujo.
- Sencillez de aprendizaje y comprensión.
- El hecho de que el LISP sea un lenguaje muy utilizado en investigación y desarrollo de Inteligencia Artificial y Sistemas Expertos.
- Sencillez de sintaxis.

El LISP es un lenguaje que es evaluado en vez de compilado o interpretado.

En los lenguajes interpretados por la computadora, cada palabra es convertida a lenguaje máquina, lo que hace que sean muy lentos. En cambio, los lenguajes compilados son mucho más rápidos porque en el proceso de compilación todo el programa se convierte en instrucciones de máquina.

Un lenguaje evaluado como el LISP es un paso intermedio entre los interpretados y los compilados. No es tan rápido como estos últimos pero resulta mucho más flexible e interactivo.

El mecanismo evaluador de LISP es la propia lista. Una lista es un conjunto de símbolos separados entre sí por, al menos, un espacio en blanco y encerrados entre paréntesis. Desde el momento en que existe una expresión encerrada entre paréntesis, el LISP la considera como una lista y la evalúa intentando ofrecer un resultado.

El LISP no es un lenguaje de programación único, sino que existen muchas versiones de LISP: MacLISP, InterLISP, ZetaLISP, Common LISP.

7.1.3 AutoLISP, una versión específica de lisp

AutoLISP es la herramienta más potente para optimizar la ejecución de AutoCAD. Le habilita para "automatizar" AutoCAD, incluso más allá de lo que puede llevar a cabo usando macros.

Los programas hechos en AutoLISP amplían los comandos y aplicaciones de AutoCAD creando, así, una solución óptima para cada problema en particular, desde el simple trazo de una línea hasta el diseño de un plano o pieza, llegando a cálculos complejos, convirtiéndose en gran ayuda para las aplicaciones de ingeniería.

Entre las aplicaciones más notables de AutoLISP se pueden citar:

- Dibujo de figuras bidimensionales con características específicas.
- Creación de objetos tridimensionales.
- Generación de gráficas de funciones basándose en ecuaciones.
- Cálculos de áreas y tablas de datos, combinación de comandos de dibujo para realizar determinados tipos de tareas.
- La creación de nuevas y únicas órdenes AutoCAD.
- La inserción de funciones especiales para dibujar y para calcular.
- Análisis detallados de gráficos y de dibujos dentro del editor de dibujos de AutoCAD.

Los programas en AutoLISP son simples archivos de texto, con la extensión obligatoria .LSP, donde el usuario escribe uno o varios programas contenidos en ese archivo. Una vez hecho esto, basta cargar el archivo.

Además, se pueden escribir directamente instrucciones en AutoLISP desde la línea de comandos del dibujo en AutoCAD, es decir, escribir conjuntos de símbolos encerrados entre paréntesis. AutoLISP evalúa inmediatamente esa lista y ofrece un resultado. Y, si la expresión contiene definiciones de funciones o variables, quedan cargadas en la memoria para su utilización posterior.

7.1.4 Elementos del lenguaje

Los objetos en AutoLISP representan todos los tipos de componentes de un programa. En esencia son dos, como ya se ha dicho, listas y símbolos.

Atendiendo a sus características se pueden definir varios tipos de objetos en AutoLISP:

- Lista: es un objeto compuesto de:
  - Paréntesis de apertura.
  - Uno o más elementos separados por, al menos, un espacio en blanco.
  - Paréntesis de cierre.
  - Los elementos de la lista pueden ser símbolos, valores concretos (“constantes" numéricas o textuales), o, también, otras listas incluidas.
- Elemento: cualquiera de los componentes de una lista.
- Átomo: representa una información indivisible, un valor concreto o un símbolo de variable que contiene un valor.
- Símbolo: cualquier elemento de la lista que no sea un valor concreto. El símbolo puede ser un nombre de variable, un nombre de función definida por el usuario o un nombre de comando de AutoLISP.

- Enteros: valores numéricos enteros, ya sean explícitos o contenidos en variables.
- Reales: valores numéricos con precisión de coma flotante.

7.1.5 Procedimientos de evaluación en AutoLISP

La base de todo intérprete de LISP es su algoritmo evaluador. Este analiza cada línea del programa y devuelve un valor como resultado. La evaluación en AutoLISP se realiza de acuerdo con las siguientes reglas:

- Los valores enteros, reales, cadenas de texto, descriptores de archivos, así como los nombres de subrutinas o comandos de AutoLISP, devuelven su propio valor como resultado.
- Los símbolos de variables participan con el valor que contienen (que les está asociado) en el momento de la evaluación
- Las listas se evalúan de acuerdo con el primer elemento. Si este es un nombre de función inherente o comando, los elementos restantes de la lista son considerados como los argumentos de ese comando. En caso contrario, se considera como un nombre de función definida por el usuario, también, con el resto de elementos como argumentos.

Cuando los elementos de una lista son a su vez otras listas, estas se van evaluando desde el nivel de anidación inferior (las listas más “interiores"). Puesto que cada lista contiene un paréntesis de apertura y otro de cierre, cuando existen listas incluidas en otras, todos los paréntesis que se vayan abriendo deben tener sus correspondientes de cierre.

También es posible evaluar directamente un símbolo (extraer, por ejemplo, el valor actual contenido en una variable) tecleando el signo de cerrar admiración seguido del nombre del símbolo.

- Por ejemplo: ! P1

7.1.6 Convenciones de AutoLISP

- Las expresiones contenidas en un programa de AutoLISP pueden introducirse directamente desde el teclado durante la edición de un dibujo de AutoCAD, escribirse en un archivo de texto ASCII o ser suministradas por una variable del tipo cadena.
- Los nombres de símbolos pueden utilizar todos los caracteres imprimibles (letras, números, signos de puntuación, etc.), salvo los prohibidos: ( ) . ' “ ;
- Los caracteres que terminan un nombre de símbolo o un valor explícito (una constante numérica o de texto) son: ( ) ' " ; (espacio en blanco) (fin de línea)

- Una expresión puede ser tan larga como se quiera, es decir, puede ocupar varias líneas del archivo de texto.
- Los espacios en blanco de separación entre símbolos son interpretados como un solo espacio en cada par de símbolos.
- Los nombres de símbolos no pueden empezar por una cifra. Las mayúsculas y minúsculas son diferentes para AutoLISP.
- Los valores explícitos (constantes) de números pueden empezar con el carácter + o, que es interpretado como el signo del número.
- Los valores de constantes de número reales deben empezar con una cifra significativa. El carácter se interpreta como el punto decimal. También se admite + o - para el signo y "e" o “E” para notación exponencial (científica).
- No es válida la coma decimal ni tampoco se puede abreviar, como en "6" (hay que poner Ø.6)

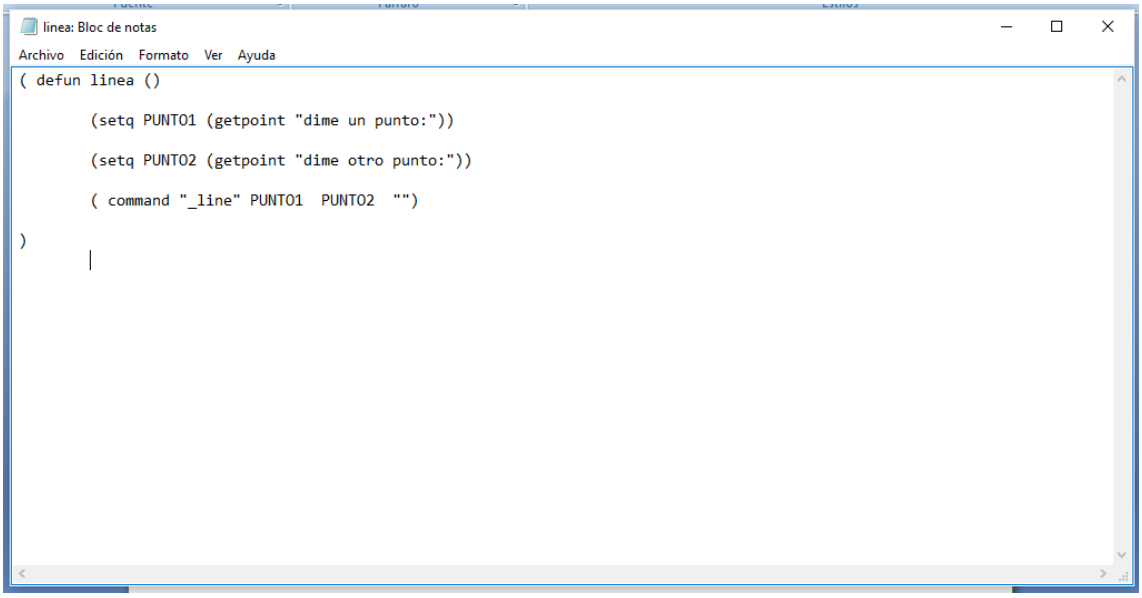
7.2 Programación en AutoLISP

7.2.1 Creación de un programa en AutoLISP.

Los programas en AutoLISP son archivos de texto con la extensión .LSP. Por tanto, se crean directamente con un Editor.

Nombre de archivo: linea.lsp

La función creada línea, al ser cargada dentro de AutoCAD, y ejecutada, provocará la creación de una línea entre dos puntos indicados.



7.2.2 Cargar programas

Existen principalmente 3 formas para cargar un programa en AutoCAD:

- Directamente sobre la barra de comandos.
- Cargando el fichero desde el exterior, mediante la pestaña “cargar aplicación”.
- A través del entorno visual, mediante la pestaña “editor visual de LISP”.

7.2.3 Definir una función/rutina/programa

En AutoLISP, una función es directamente un programa, pues su evaluación ofrece un resultado una vez cargado el archivo que contiene su definición. Así, un archivo .LSP puede contener muchos programa, según el número de funciones definidas en él. Para evaluarlas, no es necesario volver al archivo que las contiene, basta con cargarlo una sola vez.

Todas las funciones tienen esta estructura:

(DEFUN <simb> <lista argum> <expr>...)

- (DEFUN) se utiliza precisamente para definir funciones o programas de AutoLISP.
- <simb>, que es el símbolo o nombre de la función a definir. Conviene que los nombres de símbolos contengan como máximo seis caracteres, por razones de espacio ocupado en la memoria.
- Lista de argumentos (al ser una lista debe ir entre paréntesis), que puede estar vacía o contener varios argumentos y, opcionalmente, una barra inclinada y los nombres de uno o más símbolos locales de la función. Los argumentos o símbolos globales son variables que se almacenan en la memoria y permanecen en ella, lo mismo que los nombres de funciones definidas, de forma que pueden ser utilizados por otros programas o funciones de AutoLISP.

Para extraer el valor actual de esas variables, basta con introducir desde la Línea de Comando un signo “!” seguido del símbolo o nombre de la variable.

Los símbolos locales son opcionales. La barra inclinada debe estar separada del primer símbolo local y, si lo hubiera, del último global por, al menos, un espacio en blanco. Los símbolos locales se almacenan en la memoria solo temporalmente, mientras la función definida está en curso. En cuanto termina, desaparecen de la memoria, por lo que no pueden ser utilizados por otros programas o funciones. Se emplean cuando se necesitan, únicamente, dentro de la función definida, evitando que ocupen memoria inútilmente.

Si el símbolo local se encontrara ya creado antes de ser utilizado en la función definida, recuperará el valor que tenía al principio una vez terminada la función. Si no se especifican como locales al definir la función, todos los símbolos creados con SETQ dentro de ella se consideran como globales.

El último elemento de la función definidora DEFUN es la expresión en AutoLISP, que va a servir de definición de <simb>. Esta expresión puede ser tan compleja como se quiera, ejecutando otras funciones definidas, cargando archivos .LSP, etc.

Ejemplo: Command: **(DEFUN c: seno (x) (SETQ xr (* PI (/ x 18Ø.Ø))) (SETQ s (SIN xr)))**

La función definida "seno" utiliza una variable global que es "x". Lo que hace esa función es crear una variable llamada "xr", que transforma el valor del ángulo "x" dado en grados sexagesimales a radianes. Para ello, divide "x" entre 18Ø.Ø y multiplica ese resultado por PI. El valor obtenido en radianes se almacena en "xr".

A continuación, se crea una nueva variable llamada "s". SIN es el comando de AutoLISP que calcula el seno de un ángulo expresado en radianes (en este caso, el valor contenido en "xr"), y el valor de ese seno, que es el resultado final que se persigue, se almacena en la variable "s".

Las funciones definidas por el usuario mediante (DEFUN) pueden integrarse en AutoCAD como una orden más. Para ello, deben tener un nombre precedido por C: y comprender una lista de variables globales vacía, aunque pueden tener variables locales.

En el Anexo1 de este trabajo, explico todo el entorno de AutoLISP y lo amplío con la explicación de las funciones más importantes a la hora de programar con AutoLISP.

7.3 Funciones de AutoLISP

7.3.1 Funciones aritméticas

Estos cuatro comandos de AutoLISP efectúan las cuatro operaciones aritméticas básicas (Sumar, Restar, Multiplicar y Dividir) con los datos numéricos especificados a continuación.

- Sumar: (+ <nurn> <num>…) devuelve la suma de todos los <num> especificados a continuación del comando. Si todos los <num> son enteros, el resultado es entero. Si uno de ellos es real, el resultado es real. Los <num> pueden ser directamente números o bien variables pero en ese caso su contenido debe ser numérico.

Command: (SETQ nl 5 n2 7) (SETQ sum (+ 3 nl n2))

15

Command: ! sum

15

- Restar: (- <num> <num>...) devuelve el resultado de restar al primer <num> todos los demás. Es decir, se resta al primero la suma de todos los demás.

Command: (SETQ nl 12 n2 5) (SETQ res (- nl n2 3))

4

Command: ! res

4

- Multiplicar: (* <num> <num>...) evalúa el producto de todos los <num> indicados. Como siempre, si uno de los<num> es real el resultado es real.

Command: (SETQ nl 5 n2 6) (SETQ prod (* nl n2 3))

6Ø

Command: ! prod

6Ø

- Dividir: (/ <num> <num>…) divide el primer número por todos los demás. Es decir, divide el primer número por el producto de todos los demás.

Command: (SETQ nl 8Ø n2 5) (SETQ div (/ nl n2 3))

5

Command: (SETQ div (/ ni n2 3.Ø))

5.333

- (1 + <num>): incrementa una unidad al valor indicado en <num>. Equivale a (+ 1 <num>) pero de una forma más cómoda.

Command: (SETQ n (1+ n))

- (1 - <num>): resta una unidad al valor indicado en <num>. Equivale a (- <num> 1).

Command: (SETQ n (1 – n))

- (ABS <num>): devuelve el valor absoluto del número indicado en <num>.

Command: (ABS - 25.78) devuelve 25.78

- (REM <num1> <num2>): divide <num1> por <num2> y devuelve el resto de la división.

Command: (REM 2Ø 7) devuelve 6

Command: (REM 25 5) devuelve Ø

- (COS <ang>): devuelve el coseno del ángulo <ang> expresado en radianes.

Command: (COS Ø. Ø) devuelve 1.Ø

(COS PI) devuelve -1.Ø

(COS (/ PI 2)) devuelve Ø.Ø

- (SIN <ang>): devuelve el seno del ángulo <ang> indicado en radianes.

Command: (SIN (/ PI 2)) devuelve 1.Ø

(SIM 1.Ø) devuelve Ø.841471

- (ATAN <num1> |<num2>|): devuelve el arco tangente de <num1> en radianes.

Command: (ATAN 1.5) devuelve Ø.98

- (SQRT <num>): devuelve la raíz cuadrada del número <num>.

- (EXP <num>): devuelve "e" elevado a la potencia indicada en <num>.

Command: (SQRT 4) devuelve 2.ØØ

(SQRT 2) devuelve 1.4142

Command: (EXP 1) devuelve 2.71828

- (EXPT <base> <pot>): devuelve el número <base> elevado a la potencia <pot>. Si ambos números son enteros, el resultado es entero; en caso contrario, es real.

Command: (EXPT 2 4) devuelve 16

(EXPT 5.Ø 3) devuelve 125.ØØØØ

- (LOG <num>): devuelve el logaritmo neperiano del número indicado <num>.

Command: (LOG 4.5) devuelve 1.5Ø4Ø8

- (FLOAT <num1>): convierte el número indicado <num> en un número real.

Command: (FLOAT 5) devuelve 5.Ø

(FLOAT 5.25) devuelve 5.25

- (MAX <num> <num>…): devuelve el mayor de todos los números indicados. Estos pueden ser enteros o reales.

Command: (MAX 6.52 3 -7.5) devuelve 6.52

Command: (MAX -5 -7 -9) devuelve -5

- (MIN <num> <num>...): devuelve el menor de todos los números indicados. Estos pueden ser enteros o reales.

Command: (MIN 23.5 Ø 5.72) devuelve Ø.Ø

Command: (MIN Ø - 23 – 89) devuelve – 89

7.3.2 Funciones de asignación

7.3.2.1 Atribuir valores / SETQ y valores predefinidos.

Este comando tiene la siguiente estructura:

- **(SETQ <simb1><expr1> [<simb2> <expr2>...])**

Este comando atribuye el valor de la expresión <expr1> a la variable, cuyo nombre es <simb1>, el de <expr2> a <simb2>, y, así, sucesivamente. Si las variables no están previamente definidas, las crea. La función devuelve el valor atribuido a la última variable. Los nombres de símbolos no pueden ser un literal (no valdría, por ejemplo, "QUOTE x").

SETQ es la función básica de atribución de valores en AutoLISP.

Command: (SETQ x 5 y "Hola" z ' (a b) w 25.3)

25.3

Como ya se ha dicho, existen unas subrutinas de AutoLISP que no son propiamente comandos sino valores predefinidos. Estas son las siguientes:

- PI: es el valor del número real "pi" 3.1415926...
- T: es el símbolo de True, Cierto (1 lógico),
- Por último, el valor de Nada, Falso (Ø lógico), se representa en AutoLISP por nil.

Este valor aparece siempre en minúsculas y no se puede acceder a él. Por ejemplo:

Command: (SETQ nil 2Ø)

error: bad argument type

No es un símbolo propiamente sino un valor que representa Vacío o Nada.

7.3.2.2 Setvar

Cambia el valor de variables del sistema.

Sintaxis: (SETVAR <"nombre de la variable"> <nuevo valor>)

La variable del sistema no podrá ser solo de lectura y el nuevo valor asignado será de los que el sistema pueda aceptar.

Valor retornado: el nuevo valor de la variable del sistema.

(setvar "blipmode" 0) Devuelve 0 y desactiva las marcas auxiliares.

(setvar "pdmode" 34) Devuelve 34 y establece un tipo de punto

7.3.3 Funciones para gestión de listas

7.3.3.1 Creación de listas

7.3.3.1.1 List

Este comando reúne todas las expresiones indicadas en <expr> y forma con ellas una lista, que devuelve como resultado de la evaluación.

Se utiliza, sobre todo, para definir variables de un punto, una vez obtenidas las coordenadas por separado. Si, por ejemplo, como resultado de un programa se han almacenado en “x1" "y1" y "z1" las coordenadas de un punto y se quiere definir una variable "pt1" que contenga ese punto:

Command: (SETQ pt1 (x1 y1 z1))

Error: bad function

Command: (SETQ pt1 ' (x‟ y‟ z1))

(x1 y1 z1)

En el primer caso, el paréntesis que contiene las tres coordenadas es evaluado por AutoLISP y, como la función "x1" no está definida, produce un error. En el segundo caso, la variable “pt1" almacena el literal de la lista pero no con los valores de las coordenadas, que son los que interesan. Para conseguirlo, hay que utilizar el comando LIST.

Command: (LIST x1 y1 z1)

(3Ø.Ø 6Ø.Ø 9Ø.Ø)

Command: (SETQ pt1 (LIST x1 y1 z1))

(3Ø.Ø 6Ø.Ø 9Ø.Ø)

7.3.3.2 Extracción de elementos de una lista.

7.3.3.2.1 Car

Este comando devuelve el primer elemento de la lista especificada. Si <lista> es vacía "()", devuelve "nil".

Una de sus aplicaciones más importantes es extraer el primer valor de una lista que representa un punto, es decir, la coordenada X.

Por ejemplo, si un punto en 3D "pt1" es "(1Ø 25 5Ø)":

Command: (SETQ x1 (CAR pt1))

1Ø.Ø

El comando SETQ almacena en la variable "x1" el valor de la coordenada X de "pt1".

7.3.3.2.2 Cdr

Este comando devuelve una lista con los elementos de la lista que se indiquen desde el segundo hasta el final, es decir, todos los elementos menos el primero. Si <lista> es vacía, devuelve "nil".

Si <lista> es un punto en dos dimensiones, CDR devuelve el segundo elemento pero incluido en una lista. Para obtener el valor numérico de la coordenada Y, es preciso aplicar CAR a esa lista de un elemento.

Command: (SETQ pt1 ' (6 9))

(6 9)

Command: (SETQ ly1 (CDR pt1))

(9)

Command: (SETQ y1 (CAR ly1))

9

La variable "pt1" almacena en forma de lista las coordenadas X Y del punto. En "ly1" se almacena el resultado del comando CDR, que es una lista con el único elemento "9". Para obtener el valor numérico de la coordenada Y, ha habido que aplicar CAR a "ly1" y almacenar el resultado en "y1".

7.3.3.2.3 Cadr

Este comando equivale a (CAR (CDR <lista>)), por lo que es muy útil para obtener directamente la coordenada Y de una lista que represente un punto.

Command: (SETQ pt1 ' (25 36 40))

(25.Ø 36.Ø 4Ø.Ø)

Command: (SETQ y1 (CADR pt1))

36.Ø

Los comandos CAR y CDR se pueden combinar de manera similar a lo visto, hasta el cuarto nivel de anidación. Por ejemplo, si se tiene:

Command: (SETQ lis ' ((a b) (x y) 5)) f))

Las siguientes combinaciones equivaldrían a:

- (CAAR lis) a (CAR (CAR lis)) devuelve a
- (CDAR lis) a (CDR (CAR lis)) devuelve (b)
- (CADDR lis) a (CAR (CDR (CDR lis))) devuelve 5
- (CADAR lis) a (CAR (CDR (CAR lis))) devuelve b
- (CADDAR lis) a (CAR (CDR (CDR (CAR lis)))) devuelve nil

Y, así, todas las combinaciones posibles. Según lo visto, el comando para obtener la coordenada Z de un punto en 3d (tercer elemento de la lista) es CADDR.

Command: (SETQ pt1 ' (3Ø 6Ø 9Ø))

(3Ø 6Ø 9Ø)

Command: (CADDR pt1)

9Ø.Ø

7.3.4 Funciones de conversión y transformación

7.3.4.1 Angtos

- **(ANGTOS <ang>l<modo> [<prec>])**

Este comando toma el ángulo especificado en <ang>, que debe ser un número real en radianes, y lo devuelve editado como cada texto, según el formato especificado por <modo> y precisión <prec>. Estos valores corresponden a las Variables de Sistema AUNITS y AUPREC, respectivamente. Si no se suministran los argumentos <modo> y <prec>, se toman los valores actuales de esas dos Variables de Sistema.

El argumento <modo> es un número entero entre Ø y 4, cuyos formatos son los siguientes:

- Modo Ø Grados
- Modo 1 Grados/minutos/segundos.
- Modo 2 Grados centesimales g.
- Modo 3 Radianes.
- Modo 4 Geodesia.

El argumento <prec> especifica la precisión en decimales con la que se va a obtener la cadena de texto.

(ANGTOS ang Ø 2) devuelve "18Ø.ØØ"

(ANGTOS ang 1 4) devuelve "18ØdØ'Ø""

(ANGTOS ang 3 4) devuelve "3.1416r"

(ANGTOS, ang 4 3) devuelve "Ø"

7.3.4.2 Trans

En la interface de AutoCAD, se trabaja básicamente por coordenadas, por lo que en la pantalla gráfica, se representa, imaginariamente, por un eje X Y, Z para trabajar en 3 dimensiones. De esta forma, la designación se da por X Y y Z. En consecuencia, un punto en su designación consta siempre de estos tres valores, así sea 0 el valor para Z.

- **(TRANS <pto> <desde> <hasta> [<desplz>])**

Este comando convierte un punto o un vector de desplazamiento desde un Sistema de Coordenadas hasta otro. El valor del punto o del desplazamiento se indica en <pto>, como una lista de tres números reales. Si se indica el argumento optativo <desplz> y su valor es diferente de NIL, entonces, la lista de tres números reales se considera un vector de desplazamiento.

Los argumentos <desde> y <hasta> son los que indican en qué Sistema de Coordenadas se encuentra actualmente el punto o desplazamiento y en qué nuevo Sistema de Coordenadas debe expresarse el punto o desplazamiento devuelto.

Estos dos argumentos para expresar Sistemas de Coordenadas pueden ser los siguientes:

Un código que indique el Sistema de Coordenadas. Los tres códigos admitidos son:

Ø Sistema de Coordenadas Universal (SCU).

1 Sistema de Coordenadas Personal (SCP) actual.

2 Sistema de Coordenadas Vista actual (SCV).

El comando TRANS devuelve el punto o vector de desplazamiento como una lista de tres elementos expresada en el nuevo Sistema de Coordenadas indicado en <hasta>.

Por ejemplo, si el SCP actual se ha obtenido girando desde el SCU 9Ø grados sobre el eje Y:

(TRANS ' (1 2 3) 0 1) devuelve (-3.Ø 2.Ø 1.Ø)

(TRANS ' (-3 2 1) 1 0) devuelve (1.Ø 2.Ø 3.Ø)

En el primer caso, el punto (1 2 3) en el SCU (código Ø) se expresa en el SCP actual (código 1) como (-3 2 1). En el segundo caso, se hace la operación inversa desde el SCP actual (código 1) hasta el SCU (código Ø).

7.3.5 Funciones de relación

7.3.5.1 Operadores de comparación

Esta serie de comandos de AutoLISP efectúan operaciones de comparación de valores, ya sean numéricos o textuales. Los valores se pueden indicar expresamente o por las variables que los contienen.

- Igual: (= <átomo> <átomo>...) Compara los valores de todos los <átomo> especificados. Si son todos iguales, el resultado de la evaluación es "T'. En caso contrario, devuelve "nil".

Command: (SETQ x1 5) (SETQ y1 x1) (= x1 y1)

- No Igual: (/= <átomo> <átomo>…) Compara los dos valores y devuelve "T", si electivamente son distintos. La función solo está definida para dos argumentos.

Command: (SETQ a 7 b "Hola") (/ = a b)

T

- Menor: (< <átomo> <átomo> ..) Devuelve "T", si el valor del primer <átomo> es menor que el del segundo.

Si se indican más de dos <átomo>, el resultado será “T”, si cada uno de ellos es menor que el siguiente, es decir, se encuentran ordenados por valores crecientes de código ASCII.

Command: (SETQ v1 1Ø v2 3Ø v3 “A” v4 “A” v5 “f”)

Dadas esas variables con esos valores asignados, los resultados de emplear el comando "Menor" con varias combinaciones serían los siguientes:

(< v1 v2 v3 v4 v5) nil

(< v1 v2 v3 v5) T

(< v3 v5) T

(< v4 v5 v1) nil

- Menor o Igual: (<= <átomo> <átomo> ...) Similar al comando anterior, funciona de la misma manera pero comparando la condición menor o igual.

Command: (SETQ v1 3Ø v2 1Ø v3 3Ø v4 "A" v5 "a" v6 "A")

Algunas de las posibles combinaciones darían los siguientes resultados:

(<= v1 v3 v4 v6) T

(<= v2 v1 v4 v5) T

(<= v1 v3) T

(<= v5 v4 v6) nil

- Mayor: (> <átomo> <átomo>...) Compara todos los <átomo> especificados y devuelve "T", solo si cada valor es mayor que el siguiente, es decir, se encuentran ordenados de mayor a menor:

Command: (SETQ v1 2Ø v2 2Ø v3 "C" v4 "f" v5 "C")

Estos serían algunos de los resultados de la evaluación del comando con esas variables:

(> v1 v2 v3 v5 v4) nil

(> v1 v3 v4) nil

(> v4 v5 v3 v2 v1) nil

(> v4 v3 v2) T

- Mayor o Igual: (> = <átomo> <átomo> …) Similar al anterior, establece la comparación "Mayor o Igual". Devuelve "T", solo si cada <átomo> es mayor o igual que el siguiente.

Command: (SETQ v1 2Ø v2 2Ø v3 "C" v4 "f" v5 "C")

Estas variables ofrecerían los siguientes resultados:

(>= v5 v4 v3 v1) nil

(>= v4 v5 v3 v2 v1) T

(>= v4 v5 v2 v3 v1) nil

(>= v1 v2) T

7.3.5.2 Operadores lógicos

Existen, fundamentalmente, cuatro comandos que actúan como operadores lógicos: AND, OR, EQUAL y NOT. Estos comandos devuelven, únicamente, los resultados "T" (cierto) o "nil" (Falso).

- (AND <expr> ...) : "Y" lógico. Este comando realiza el "y" lógico de una serie de expresiones, que son las indicadas en <expr). Evalúa todas las expresiones y devuelve "T", si ninguna de ellas es "nil". En el momento en que encuentra alguna expresión con resultado "nil", abandona la evaluación de las demás y el comando devuelve “nil”. Su función es detectar si existe algún "nil" en las expresiones indicadas.

Command: (SETQ v1 1Ø v2 1Ø) (AND (<= v1 v2) (>= v1 v2)) devuelve T (AND (= v1 v2) (> v1 v2)) devuelve nil

- (OR <expr> ...) : "O" lógico . Realiza el “o" lógico de la serie de expresiones indicadas. Evalúa todas las expresiones y devuelve “nil”, si todas ellas son “nil”. En el momento en que encuentra un resultado diferente de “nil”, abandona la evaluación y devuelve "T".

Command: (SETQ v1 2Ø v2 2Ø)(OR (< vi v2) (> v1 v2)) devuelve nil

(OR (<= v1v2) (/= v1 v2)) devuelve T

- (EQUAL <expr1> <expr2> | <margen>|): Identidad lógica. Comprueba si las dos expresiones indicadas son idénticas, es decir, si el resultado de ambas evaluaciones es el mismo. En ese caso, devuelve "T" o "nil", si no es el mismo. A diferencia de "=", se puede utilizar para comparar cualquier expresión. Una aplicación muy sencilla es comparar si dos puntos son iguales:

Command: (SETQ pt1 ' (5Ø 25 Ø) pt2 ' (5Ø 25 Ø) pt3 ' (3Ø 2Ø 1Ø))

(EQUAL pt1 pt2) devuelve T

(EQUAL pt1 pt3) devuelve nil

- (NOT <expr>) : "NO" lógico. Admite una única <expr> como argumento. Devuelve el "no" lógico de esa expresión. Es decir, si el resultado de esa explosión es diferente de "nil", devuelve "nil", y si el resultado es "nil", devuelve "T".

Command: (SETQ v1 2Ø v2 3Ø)(NOT (> v1 v2)) devuelve T

(NOT (< v1 v2)) devuelve nil

7.3.6 Funciones de condición

7.3.6.1 Cond

- **(COND (<cond1 > <res1 >) (<cond2> <res2>) ...)**

Este comando permite establecer una serie de condiciones especificando diferentes actuaciones, según cuál sea la primera condición en cumplirse.

Los argumentos del comando COND son un número cualquiera de listas, tantas como condiciones se pretendan establecer. Cada lista contiene dos especificaciones: la condición <cond> y el resultado a ejecutar <res>, en el caso de que esa condición se cumpla (devuelva un valor diferente de "nil"). La condición <cond>, normalmente, es una lista con un comando de comparación o un operador lógico (para comprobar si la condición se cumple).

La segunda especificación de cada lista es el resultado a ejecutar. Este no tiene porqué ser una única lista, pueden ser varias, que se irán ejecutando todas, en caso de que la condición se cumpla.

Command:

(SETQ c 1)

(COND ((= c 1) (SETQ n T) (PROMPT "Correcto") (TERPRI)

(T (PROMPT "Incorrecto") (TERPRI))

)

En el ejemplo, el comando COND tiene dos listas como argumento, es decir, establece dos condiciones: en la primera, si el valor de "c" es igual a 1, la lista (= c l) devuelve "T" y, entonces, la condición se cumple y se ejecutan los comandos siguientes, que son SETQ, PROMPT y TERPRI.

La segunda condición no es una lista sino "T" (cierto), es decir, se obliga a que sea siempre cierta. El comando COND no ejecuta todas las condiciones establecidas, sino que va comprobándolas hasta encontrar una que devuelva un valor diferente de "nil" y, entonces, ejecuta las expresiones establecidas como resultado <res>.

Por tanto, COND evalúa las expresiones de la primera condición que encuentre que se cumpla (evaluación diferente de "nil"). Ya no sigue revisando las siguientes condiciones, aunque haya más que también se cumplan.

7.3.6.2 If

- **(IF <cond><acción-cierto> [<acción-falso>])**

Este comando es una de las estructuras básicas de programación: la secuencia alternativa. La función 1 evalúa la expresión indicada como condición en <cond>. Si el resultado no es "nil", entonces, pasa a evaluar la expresión contenida en <acción-cierto>. En este caso, devuelve el resultado de esa expresión.

Si <cond> es "nil", entonces, pasa a evaluar la expresión contenida en <acción-falso>, en el caso de que se haya indicado y devuelve su resultado. Si no se ha indicado <acciónfalso>, devuelve "nil".

Command:

(SETQ pt1 (GETPOINT "Primer punto: ")) (TERPRI)

(SETQ pt2 (GETPOINT "Segundo punto: ")) (TERPRI)

(IF (EQUAL pt1 pt2)

(PROMPT "Son iguales")

(PROMPT "No son iguales")

) (TERPRI)

El ejemplo anterior acepta dos puntos introducidos por el usuario y, después, compara si las dos listas que representan esos puntos son iguales con el comando de operación lógica EQUAL. Si son iguales, resultado de <cond> es "T" (cierto) y, entonces, IF devuelve PROMPT "Son iguales". Si no son iguales, el resultado de <cond> es "nil", con lo que IF devuelve PROMPT "No son iguales".

7.3.7 Funciones para gestión de ciclos

7.3.7.1 While

- **(WHILE <cond> <acción> ,..)**

Es otro comando para establecer secuencias repetitivas en el programa. En este caso, el ciclo no se repite un determinado número de veces, "N", sino mientras se cumpla la condición especificada <cond>. Esta condición sirve, pues, como elemento de control de la repetitiva.

Mientras el resultado de evaluar <cond> sea diferente de "nil", el comando WHILE evalúa la expresión indicada a continuación <acción>, que puede ser tan compleja como se desee. Después, vuelve a evaluar <cond>. Esto se repite hasta que al evaluar <cond> resulte "nil". Entonces, deja de repetirse el ciclo y WHILE devuelve el resultado de la última expresión <acción> realizada.

Command:

```
( DEFUN pg15 ( / n ctrl pt1 pt2 )
( SETQ pt1 ( GETPOINT "Introducir punto: " ) )
( SETQ n ( GETINT "Numero de líneas: " ) )
( SETQ ctrl T)
( WHILE ctrl
( SETQ pt2 ( GETPOINT pt1 "Nuevo punto: " ) )
( COMMAND "line" pt1 pt2 " " )
( SETQ pt1 pt2 )
( SETQ n ( - n 1 ) )
( IF ( = n Ø ) ( SETQ ctrl nil ) )
)
)
```

Se define una variable "n", que almacena el número de líneas de la poligonal, valor introducido por el usuario. Cada vez que se ejecuta un ciclo, al valor de "n" se le resta 1, con lo que se lleva la cuenta del número de ciclos que faltan por ejecutar.

La variable "n" se inicializa fuera del ciclo antes de entrar en él, aceptando el valor introducido mediante GETINT. La variable de control "ctrl" se inicializa, también, fuera del ciclo con el valor "T". Este procedimiento de inicializar determinadas variables antes de entrar por primera vez en el ciclo es muy habitual en las estructuras repetitivas.

El comando IF chequea el valor de "n" y, cuando es Ø, significa que se acaba de realizar el último ciclo. En ese momento (= n Ø), devuelve "T" (cierto), con lo que se asigna el valor nulo "nil" a la variable de control "ctrl". Esto provoca el final de la repetitiva establecida por WHILE.

7.3.8 Funciones de entrada interactiva

7.3.8.1 Getpoint

- **(GETPOINT [<pto>] [<"mensaje">])**

Este comando aguarda a que el usuario introduzca un punto, bien directamente por teclado o señalando en pantalla, y devuelve las coordenadas de ese punto en el SCP actual. Estas coordenadas se devuelven como una lista con los tres valores reales de X Y Z.

Command: (SETQ pt1 (GETPOINT " Introducir un punto: "))

Introducir un punto:

(Se señala un punto en pantalla o se digita las coordenadas)

(15Ø.Ø 85.Ø Ø.Ø)

Command: !pt1 (15Ø.Ø 85.Ø Ø.Ø)

7.3.8.2 Getint

- **(GETINT [<"mensaje">])**

Este comando o función inherente de AutoLISP acepta un número entero introducido por el usuario y devuelve ese número. El valor debe encontrarse entre -32768 y +32767. Si el usuario introduce un número no entero o un dato no numérico, se produce un mensaje de error y AutoLISP solicita que pruebe otra vez.

Command: (SETQ nent (GETINT "Introducir numero: "))

Introducir número: 25 (Return)

25

Command: ! nent

7.3.9 Funciones de lectura y escritura

7.3.9.1 Prin1

Imprime una expresión AutoLISP en el área de órdenes o la escribe en un archivo abierto en modo de escritura.

Sintaxis: (PRIN1 [expresión] [descriptor de fichero])

Valor retornado: el argumento de la expresión.

Es posible llamar a la función prin1 sin argumentos, en cuyo caso devuelve (e imprime) la cadena nula.

(Setq a 123 b '(a))

(prin1 'a) imprime A y devuelve A

(prin1 a) imprime 123 y devuelve 123

7.3.9.2 Princ

Imprime una expresión AutoLISP en el área de órdenes o la escribe en un archivo. La diferencia con PRIN1 es que PRINC evalúa los caracteres de control que aparezcan en la expresión.

Sintaxis: (PRINC [expresión] [descriptor de fichero])

Valor retornado: el argumento de la expresión.

7.3.9.3 Print

Imprime un salto de línea, una expresión AutoLISP y un espacio en el área de órdenes o lo escribe en un archivo.

Sintaxis: (PRINT [expresión] [descriptor de fichero])

Valor retornado: el argumento de la expresión.

7.3.9.4 Prompt

Presenta una cadena de caracteres en el área de órdenes. (Evalúa caracteres de control)

Sintaxis: (PROMPT <”cadena”>)

Valor retornado: nil

En las configuraciones de pantalla dual de AutoCAD, PROMPT muestra la cadena en ambas pantallas, por lo que resulta más útil princ.

(prompt "Nuevo valor: ") muestra Nuevo valor: en la(s) pantalla(s)

7.3.10 Funciones relativas a nombres de entidades.

7.3.10.1 Entget

Este comando busca, en la Base de Datos, el nombre de entidad indicado y devuelve la lista completa correspondiente a esa entidad. Se observa que el comando requiere indicar el nombre de la entidad y, por tanto, es preciso obtener previamente este nombre (con SSNAME, ENTNEXT, ENTLAST etc.).

Por ejemplo, para obtener la lista correspondiente a un círculo dibujado desde un programa en AutoLISP se haría de la siguiente manera:

Command:

(COMMAND "circulo" "1Ø,1Ø" “5”)

(SETQ ent (ENTGET (ENTLAST)))

7.3.10.2 Entlast

Este comando devuelve el nombre de la última entidad principal no eliminada de la Base de Datos. Se utiliza fundamentalmente para obtener el nombre de las entidades recién añadidas a la Base de Datos con COMMAND. Presenta la gran ventaja de que obtiene el nombre de la entidad aunque no sea visible en pantalla o se encuentre en una capa inutilizada.

Command:

(COMMAND "línea" p1 p2 "")(SETQ lin (ENTLAST))

La variable "lin" almacena el nombre de la línea recién dibujada, sin necesidad de tener que designarla. Con ese nombre ya es posible acceder a la Base de Datos y realizar todas las operaciones con la línea en cuestión directamente.

7.3.10.3 Entsel

El comando ENTSEL espera que el usuario designe una única entidad mediante un punto y devuelve una lista cuyo primer elemento es el nombre de la entidad y su segundo elemento el punto de designación. De esta forma, se tienen asociados ambos valores para procesarlos posteriormente.

Command:

(SETQ ent (ENTSEL "Designar una entidad mediante punto: "))

Designar una entidad mediante punto:

(<Entity name: 6ØØØØ32A> (1Ø.Ø 2Ø.Ø Ø.Ø))

En el ejemplo, el nombre de la entidad designada es "6ØØØØ32A" y el punto de designación (1Ø.Ø 2Ø.Ø Ø.Ø).

7.3.10.4 Entdel

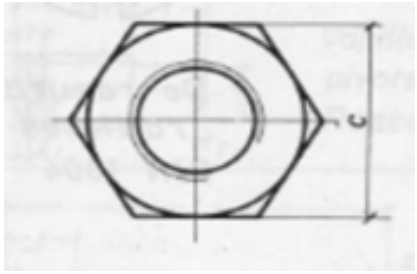
La entidad, cuyo nombre se indica, es borrada, si existe en la Base de Datos del dibujo actual o recuperada, si había sido previamente borrada en la actual sesión de dibujo. Esto quiere decir que las entidades borradas con ENTDEL pueden ser, después, recuperadas con el mismo ENTDEL, antes de salir de la actual edición del dibujo.

Las entidades borradas con ENTDEL se eliminan definitivamente al salir del Editor de Dibujo, por lo que ya no podrían ser ya recuperadas.

Command: (COMMAND "línea" "5,5" “2Ø,2Ø" "")(ENTDEL (ENTLAST))

8 Anexo 2: Pasos para elementos roscados.

- Paso 1: En el primer paso, se realizará la cabeza del tornillo dibujando un rectángulo, cuya altura es la altura de la cabeza en mm (h= 0,8 d) y cuya anchura es el doble del diámetro nominal de la rosca.
- Paso 2: En este paso, obtenemos la medida c, a partir de la realización de un hexágono regular de lado el diámetro nominal de la rosca.

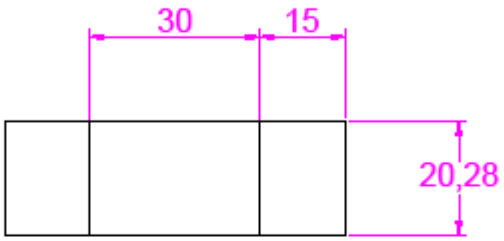


Con esta medida, obtenemos el centro para trazar las aristas del bisel y ese centro lo unimos con los vértices del rectángulo para, después, mediante paralelas, obtener los otros dos centros para poder trazar las aristas del bisel.

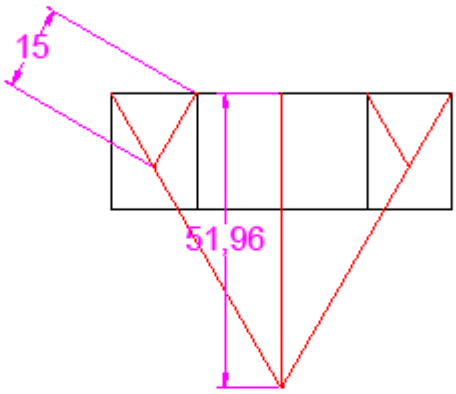
- Paso 3: Trazamos las aristas y borramos las partes sobrantes.

- Paso 4: En el cuarto paso, trazamos el cuerpo del tornillo. Conocida la distancia total del tornillo, se toma como origen la cabeza del tornillo y, a partir de ese punto, se traslada la distancia hasta la cual debemos trazar dos paralelas separadas el diámetro nominal del tornillo.

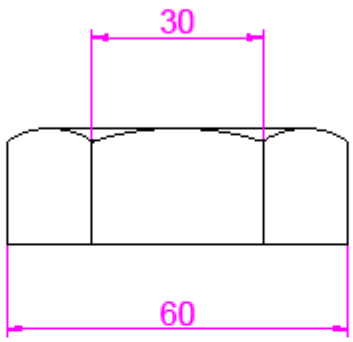
- Paso 5: Por último, dibujamos la longitud roscada del tornillo, trasladando la distancia desde el final del tronillo. Después, solo falta dibujar las líneas que representan la profundidad de la rosca, las cuales serán paralelas a las líneas de contorno y estarán separadas una distancia que será la profundidad de la rosca.



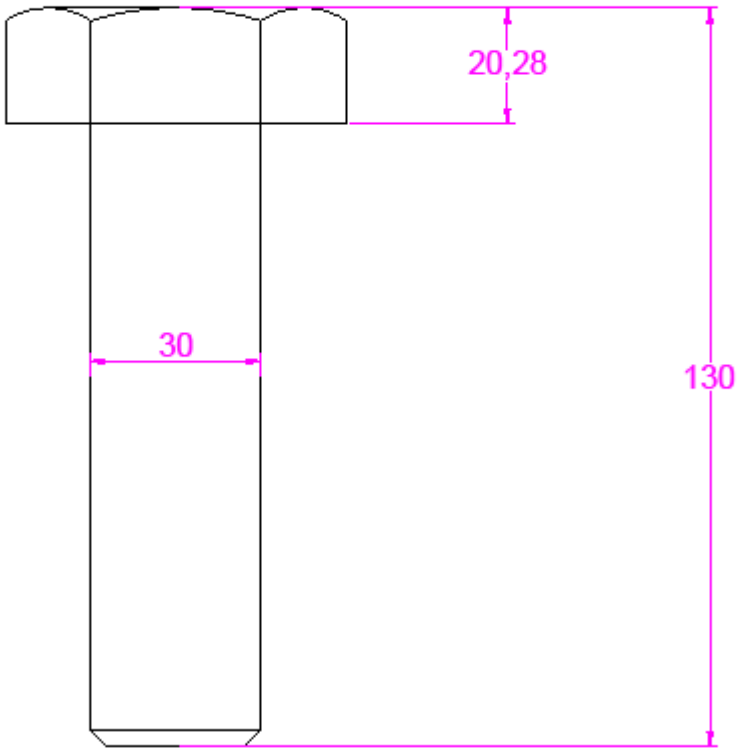
UNIVERSIDAD DE LEÓN **ESCUELA SUPERIOR Y TÉCNICA DE INGENIEROS DE MINAS**			
GRADO EN INGENIERÍA DE LA ENERGÍA			
PROYECTO DE			
PLANO DE	SOLUCIÓN GRÁFICA ELEMENTOS ROSCADOS PASO1		
ESCALA	1/1		PLANO Nº
FECHA	28/6/16	Fdo.:.....	



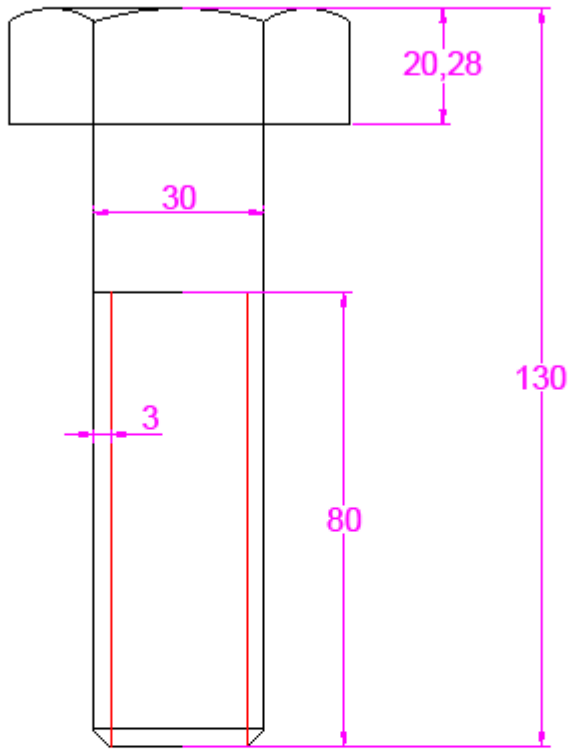
UNIVERSIDAD DE LEÓN **ESCUELA SUPERIOR Y TÉCNICA DE INGENIEROS DE MINAS**			
GRADO EN INGENIERÍA DE LA ENERGÍA			
PROYECTO DE			
PLANO DE	SOLUCIÓN GRÁFICA ELEMENTOS ROSCADOS PASO 2		
ESCALA	1/1		PLANO Nº
FECHA	28/6/16	Fdo.:.....	



UNIVERSIDAD DE LEÓN **ESCUELA SUPERIOR Y TÉCNICA DE INGENIEROS DE MINAS**			
GRADO EN INGENIERÍA DE LA ENERGÍA			
PROYECTO DE			
PLANO DE	SOLUCIÓN GRÁFICA ELEMENTOS ROSCADOS PASO3		
ESCALA	1/1		PLANO Nº
FECHA	28/6/16	Fdo.:.....	



UNIVERSIDAD DE LEÓN **ESCUELA SUPERIOR Y TÉCNICA DE INGENIEROS DE MINAS**	
GRADO EN INGENIERÍA DE LA ENERGÍA	
PROYECTO DE	
PLANO DE	SOLUCIÓN GRÁFICA ELEMENTOS ROSCADOS PASO4
ESCALA	1/1
FECHA	28/6/16
Fdo.:.....	PLANO Nº



UNIVERSIDAD DE LEÓN **ESCUELA SUPERIOR Y TÉCNICA DE INGENIEROS DE MINAS**			
GRADO EN INGENIERÍA DE LA ENERGÍA			
PROYECTO DE			
PLANO DE	SOLUCIÓN GRÁFICA ELEMENTOS ROSCADOS PASO 5		
ESCALA	1/1		PLANO Nº
FECHA	28/6/16	Fdo.:.....	

9 Anexo 3: Rutina para la realización de elementos roscados.

```
(defun C:Tornillo ()
(setq PI (getpoint "\Punto de inserción del tornillo: "))
(setq D (getreal "Distancia entre aristas del hexágono de la cabeza: "))
(setq H (getreal "Altura de la cabeza del tornillo: "))
(setq DIAMROS (getreal "Diámetro de la rosca: "))
(setq LONGROS (getreal "Longitud de la rosca: "))
(setq LONGTOR (getreal "Longitud del tornillo: "))
(setq PIx (car PI))
(setq PIy (car (cdr PI)))
(setq P1 (list (- PIx (/ D 2)) (+ PIy (- H (* 0.0428932 D)))))
(setq P2 (list (- PIx (/ D 4)) (+ PIy (- H (* 0.0428932 D)))))
(setq P3 (list (+ PIx (/ D 4)) (+ PIy (- H (* 0.0428932 D)))))
(setq P4 (list (+ PIx (/ D 2)) (+ PIy (- H (* 0.0428932 D)))))
(setq PC1 (list (- PIx (* 0.375 D)) (+ PIy H)))
(setq PC2 (list PIx (+ PIy H)))
(setq PC3 (list (+ PIx (* 0.375 D)) (+ PIy H)))
(setq PB1 (list (- PIx (/ D 2)) PIy))
(setq PB2 (list (- PIx (/ D 4)) PIy))
(setq PB3 (list (+ PIx (/ D 4)) PIy))
(setq PB4 (list (+ PIx (/ D 2)) PIy))
(setq PT1 (list (- PIx (/ DIAMROS 2)) PIy))
(setq PT2 (list (- PIx (/ DIAMROS 2)) (- PIy (- LONGTOR 1))))
(setq PT3 (list (- PIx (/ (- DIAMROS 2) 2)) (- PIy LONGTOR)))
(setq PT4 (list (+ PIx (/ (- DIAMROS 2) 2)) (- PIy LONGTOR)))
(setq PT5 (list (+ PIx (/ DIAMROS 2)) (- PIy (- LONGTOR 1))))
(setq PT6 (list (+ PIx (/ DIAMROS 2)) PIy))
(setq PT7 (list (- PIx (/ DIAMROS 2)) (- PIy (- LONGTOR LONGROS))))
(setq PT8 (list (+ PIx (/ DIAMROS 2)) (- PIy (- LONGTOR LONGROS))))
(setq PT9 (list (- PIx (/ (- DIAMROS 2) 2)) (- PIy (- LONGTOR LONGROS))))
(setq PT10 (list (+ PIx (/ (- DIAMROS 2) 2)) (- PIy (- LONGTOR LONGROS))))
(setq PE1 (list PIx (+ PIy H (/ H 6))))
(setq PE2 (list PIx (- PIy (+ LONGTOR (/ H 6)))))
```

```
(command "_color" 4)
(command "_line" PE1 PE2)
(command "")
(command "_color" "_bylayer")
(command "_arc" P1 PC1 P2)
(command "_arc" P2 PC2 P3)
(command "_arc" P3 PC3 P4)
(command "_line" PC1 PC3 )
(command "")
(command "_line" PB1 PB4 )
(command "")
(command "_line" PB1 P1 )
(command "")
(command "_line" PB2 P2 )
(command "")
(command "_line" PB3 P3 )
(command "")
(command "_line" PB4 P4 )
(command "")
(command "_line" PT1 PT2 )
(command "")
(command "_line" PT2 PT3 )
(command "")
(command "_line" PT3 PT4 )
(command "")
(command "_line" PT4 PT5 )
(command "")
(command "_line" PT5 PT6 )
(command "")
(command "_line" PT2 PT5 )
(command "")
(command "_line" PT7 PT8)
(command "")
```

```
(command "_color" 2 "")
(command "_line" PT9 PT3)
(command "")
(command "_line" PT10 PT4)
(command "")
(command "_color" "_bylayer" "")
)
```

10 Anexo 4: Pasos para pozos mineros.

- Paso 1: Situamos en el dibujo la cabeza del pozo A, la cual tendrá una altitud. Seguidamente, se traza una línea, que sale de ese punto con la dirección del rumbo que se especifica para, después, poder trasladar sobre esa recta los grados de la pendiente del pozo y trazar otra recta sobre la cual trasladaremos las distancias de las perforaciones.

  Cuando ya tengamos marcadas las perforaciones para llegar al techo y al muro del estrato, tiramos desde esos puntos perpendiculares hasta la primera línea trazada obteniendo, así, el muro del pozo a y el techo del pozo a.

  La distancia de esas perpendiculares, teniendo en cuenta la escala del dibujo, es la altitud que descendemos y la cual hay que restar a la altitud inicial para saber la altitud de los dos puntos.

- Paso 2: El siguiente paso es situar en el dibujo la cabeza del pozo b con las medidas que nos facilitan, las cuales tienen como punto de origen la cabeza del pozo a.

- Paso 3: Repetimos el mismo proceso que realizamos en el paso 1, esta vez para poder obtener el techo del pozo b y el muro del pozo b.

- Paso 4: Se borran las líneas y trazos sobrantes para tener una mayor claridad en los siguientes pasos.

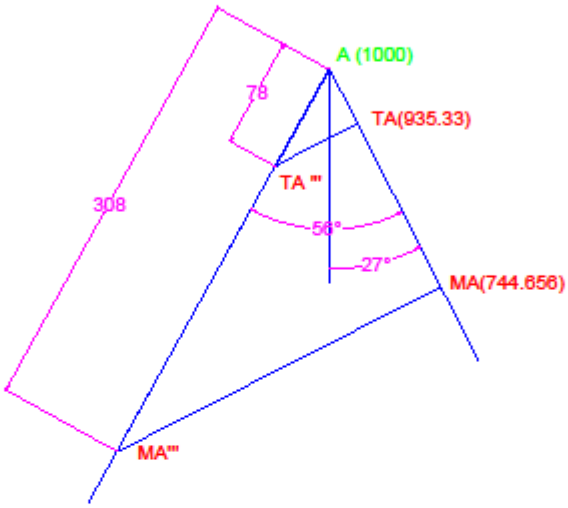
- Paso 5: Trazamos dos rectas: una, desde el techo del pozo a hasta el techo del pozo b, y otra, desde el muro del pozo a hasta el muro del pozo b. Estas rectas serán paralelas en el espacio y estarán contenidas en los planos del techo y del muro del estrato.

- Paso 6: Para poder determinar el plano del techo, necesitamos 3 puntos contenidos en él y, para ello, lo que hacemos es trasladar la recta contenida en el muro hasta el punto b del techo. Esto se puede hacer, ya que consideramos y simplificamos que los dos planos son paralelos.

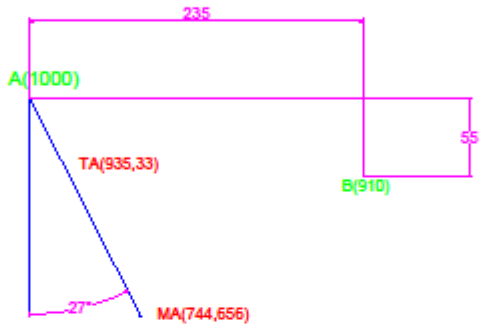
  El nuevo punto tendrá la misma diferencia de altitud con el punto b del techo, la misma que tenían el punto a y el punto b del muro.

- Paso 7: Se unen los puntos a del techo y el tercer punto del techo mediante una recta, la cual servirá para realizar un abatimiento.

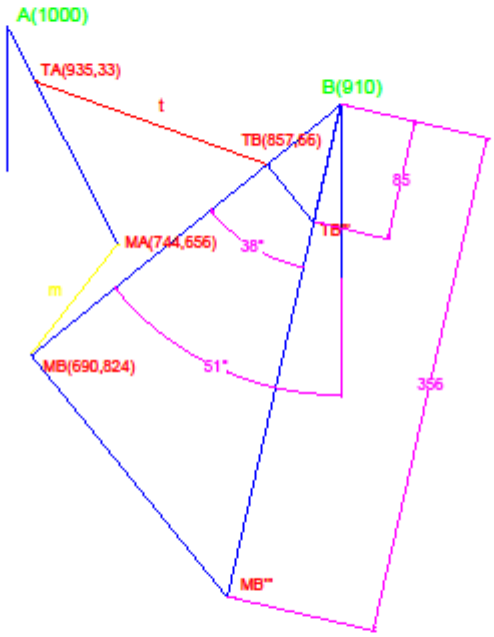
- Paso 8: Mediante un abatimiento, encontramos en la recta, que une el punto a del techo y el tercer punto del techo, un punto que tiene la misma altitud que el punto b del techo, con el fin de trazar una horizontal contenida en el plano del techo y, con esa horizontal, calculamos el buzamiento del pozo.
- Paso 9: Realizamos un cambio de plano para poder ver el techo proyectante, así, vemos de manera directa la pendiente y la potencia del pozo.
- Paso 10: Trasladamos todos los puntos pertenecientes a los planos para verlos proyectantes para, así, calcular la potencia y la pendiente.



UNIVERSIDAD DE LEÓN **ESCUELA SUPERIOR Y TÉCNICA DE INGENIEROS DE MINAS**			
GRADO EN INGENIERÍA DE LA ENERGÍA			
PROYECTO DE			
PLANO DE	RESOLUCIÓN GRÁFICA DE POZOS MINEROS PASO 1		
ESCALA	1/4000		PLANO Nº
FECHA	28/6/16	Fdo.:.....	



UNIVERSIDAD DE LEÓN **ESCUELA SUPERIOR Y TÉCNICA DE INGENIEROS DE MINAS**			
GRADO EN INGENIERÍA MINERA			
PROYECTO DE			
PLANO DE	RESOLUCIÓN GRÁFICA DE POZOS MINEROS PASO 2		
ESCALA	1/4000		PLANO Nº
FECHA	28/6/16	Fdo.:.....	

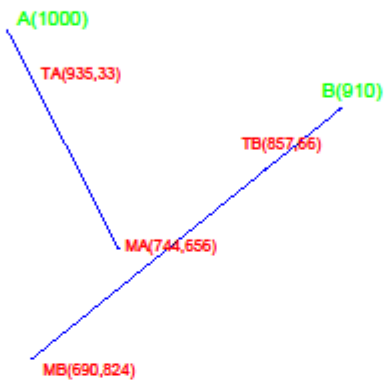


UNIVERSIDAD DE LEÓN
ESCUELA SUPERIOR Y TÉCNICA DE INGENIEROS DE MINAS

GRADO EN INGENIERÍA DE LA ENERGÍA

PROYECTO DE

PLANO DE	RESOLUCIÓN GRÁFICA DE POZOS MINEROS PASO 3		
ESCALA	1/4000		PLANO Nº
FECHA	28/6/16	Fdo.:.....	

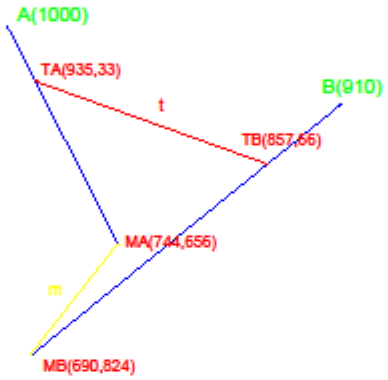


UNIVERSIDAD DE LEÓN
ESCUELA SUPERIOR Y TÉCNICA DE INGENIEROS DE MINAS

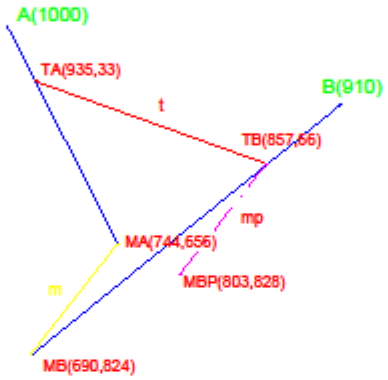
GRADO EN INGENIERÍA DE LA ENERGÍA

PROYECTO DE

PLANO DE	RESOLUCIÓN GRÁFICA DE POZOS MINEROS PASO 4		
ESCALA	1/4000		PLANO Nº
FECHA	28/6/16	Fdo.:.....	



UNIVERSIDAD DE LEÓN **ESCUELA SUPERIOR Y TÉCNICA DE INGENIEROS DE MINAS**			
GRADO EN INGENIERÍA DE LA ENERGÍA			
PROYECTO DE			
PLANO DE	RESOLUCIÓN GRÁFICA DE POZOS MINEROS PASO 5		
ESCALA	1/4000		PLANO Nº
FECHA	28/6/16	Fdo.:.....	

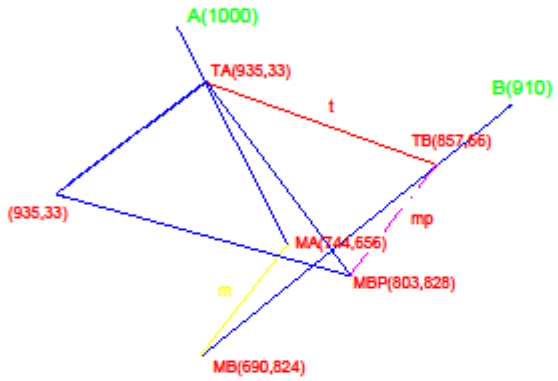


UNIVERSIDAD DE LEÓN
ESCUELA SUPERIOR Y TÉCNICA DE INGENIEROS DE MINAS

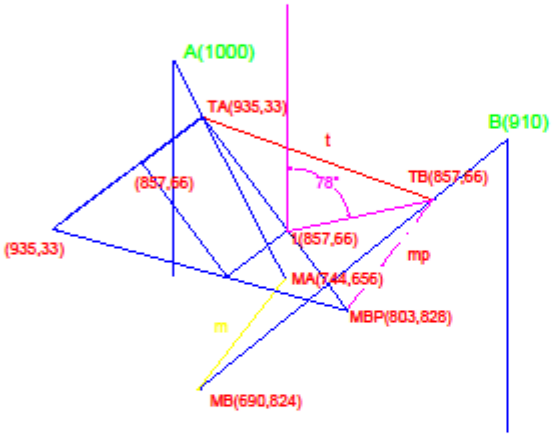
GRADO EN INGENIERÍA DE LA ENERGÍA

PROYECTO DE

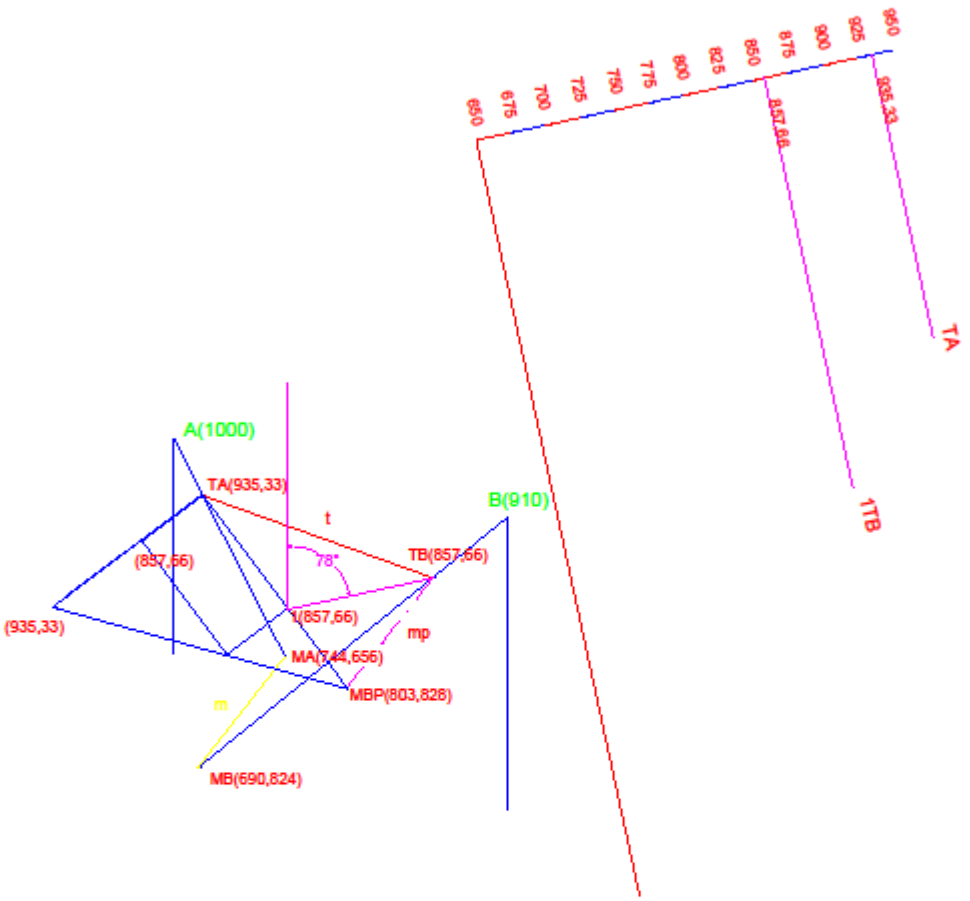
PLANO DE	RESOLUCIÓN GRÁFICA DE POZOS MINEROS PASO 6		
ESCALA	1/4000		PLANO Nº
FECHA	28/6/16	Fdo.:.....	



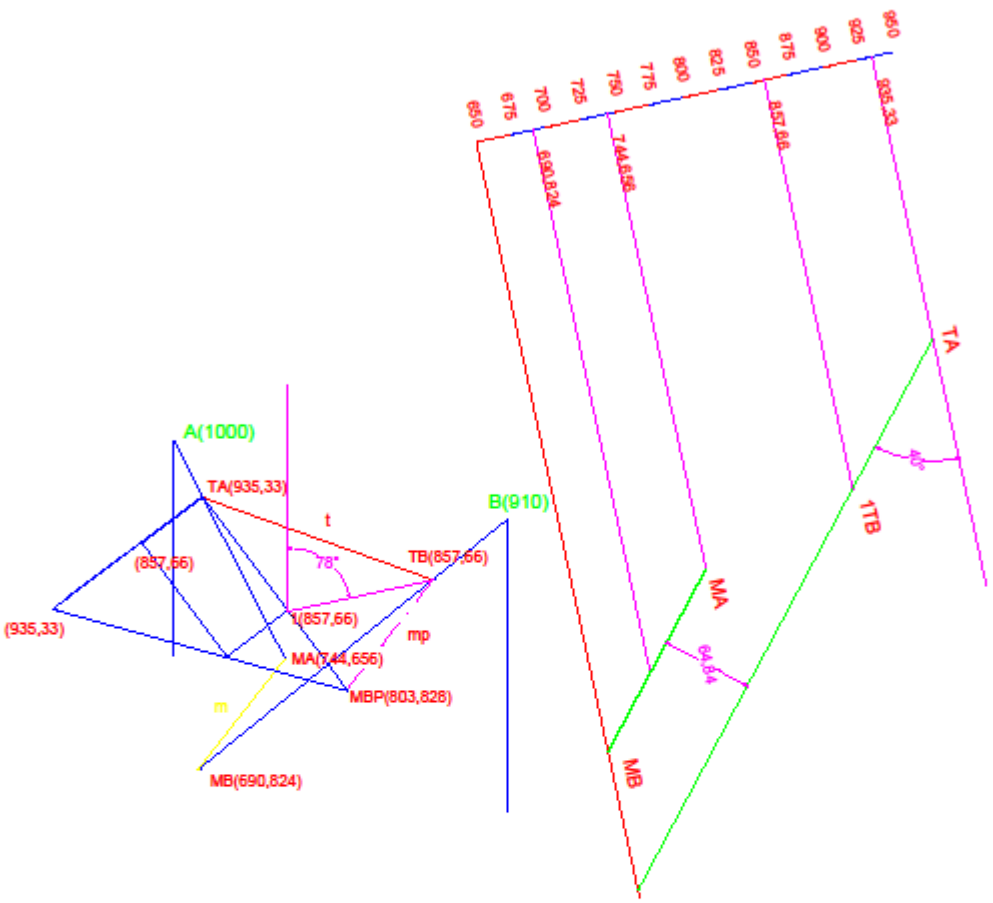
UNIVERSIDAD DE LEÓN **ESCUELA SUPERIOR Y TÉCNICA DE INGENIEROS DE MINAS**			
GRADO EN INGENIERÍA DE LA ENERGÍA			
PROYECTO DE			
PLANO DE	RESOLUCIÓN GRÁFICA DE POZOS MINEROS PASO 7		
ESCALA	1/4000		PLANO Nº
FECHA	28/6/16	Fdo.:.....	



UNIVERSIDAD DE LEÓN ESCUELA SUPERIOR Y TÉCNICA DE INGENIEROS DE MINAS			
GRADO EN INGENIERÍA DE LA ENERGÍA			
PROYECTO DE			
PLANO DE	RESOLUCIÓN GRÁFICA DE POZOS MINEROS PASO 8		
ESCALA	1/4000		PLANO Nº
FECHA	28/6/16	Fdo.:.....	



UNIVERSIDAD DE LEÓN ESCUELA SUPERIOR Y TÉCNICA DE INGENIEROS DE MINAS			
GRADO EN INGENIERÍA DE LA ENERGÍA			
PROYECTO DE			
PLANO DE	RESOLUCIÓN GRÁFICA DE POZOS MINEROS PASO 9		
ESCALA	1/4000		PLANO Nº
FECHA	28/6/16	Fdo.:.....	



UNIVERSIDAD DE LEÓN **ESCUELA SUPERIOR Y TÉCNICA DE INGENIEROS DE MINAS**			
GRADO EN INGENIERÍA DE LA ENERGÍA			
PROYECTO DE			
PLANO DE	RESOLUCIÓN GRÁFICA DE POZOS MINEROS PASO 10		
ESCALA	1/4000		PLANO Nº
FECHA	28/6/16	Fdo.:.....	

11 Anexo 5: Rutina para la realización de pozos mineros.

```
(defun c:pozosmineros ()
(setq pa (getpoint "dime la ubicacion de la boca del pozo A:"))
(setq pax (car pa))
(setq pay (car(cdr pa)))
(setq paz (caddr pa))
(setq pendiente1 (getint "dime la pendiente del pozo A:"))
(setq rumboa (getint "dime el rumbo del pozo a:") )
(command "color" 3)
(cond
(  ( and ( > rumboa 0) ( < rumboa 90 ))
(setq rumbo1 (/ (* rumboa PI) 180 ))
(setq perforacion1 (getint "dime la perforacion1:"))
(setq h (* perforacion1 (cos ( / ( * pendiente1 PI) 180))))
(setq z (* perforacion1 (sin ( / ( * pendiente1 PI) 180))))
(setq techo1 (list (+ pax (* h (cos rumbo1))) (+ pay (* h (sin rumbo1))) (- paz z)))
(command "_line"  pa techo1 "")
(setq perforacion2 ( getint "dime la perforacion2:"))
(setq h1 (* perforacion2 (cos ( / (* pendiente1 PI) 180))))
(setq z1 (* perforacion2 (sin ( / (* pendiente1 PI) 180))))
(setq muro1 (list (+ pax (* h1 (cos rumbo1))) (+ pay (* h1 (sin rumbo1))) (- paz z1)))
(command "_line"  pa muro1 "")
)

(  ( and ( > rumboa  90 ) ( < rumboa 180 ))
(setq rumbo1 (/ (* (- 180 rumboa ) PI) 180 ))
(setq perforacion1 (getint "dime la perforacion1:"))
(setq h (* perforacion1 (cos ( / (* pendiente1 PI) 180 ))))
(setq z (* perforacion1 (sin ( / (* pendiente1 PI) 180))))
(setq techo1 (list (- pax (* h (cos rumbo1))) (+ pay (* h ( sin rumbo1)))  (- paz z) ))
(command "_line" pa techo1 "")
(setq perforacion2 (getint "dime la perforacion2:"))
(setq h1 (* perforacion2 ( cos ( / (* pendiente1 PI) 180))))
```

```
(setq z1 (* perforacion2 ( sin ( / (* pendiente1 PI) 180))))
(setq muro1 (list (- pax (* h1 (cos rumbo1))) (+ pay (* h1 (sin rumbo1))) (- paz z1)))
(command "_line" pa muro1 "")
)
( (and ( > rumboa 180)(< rumboa 270))
(setq rumbo1 (/ (* (- 270 rumboa) PI) 180 ))
(setq perforacion1 (getint "dime la perforacion1:"))
(setq h (* perforacion1 (cos ( / (* pendiente1 PI) 180))))
(setq z (* perforacion1 (sin ( / (* pendiente1 PI) 180))))
(setq techo1 (list (- pax (* h (sin rumbo1))) (- pay (* h ( cos rumbo1)))  (- paz z) ))
(command "_line" pa techo1 "")
(setq perforacion2 (getint "dime la perforacion2:"))
(setq h1 (* perforacion2 (cos ( / (* pendiente1 PI) 180))))
(setq z1 (* perforacion2 (sin ( / (* pendiente1 PI) 180))))
(setq muro1 (list (- pax (* h1 (sin rumbo1))) (- pay (* h1 (cos rumbo1))) (- paz z1)))

(command "_line" pa muro1 "")
)
( ( and (>  rumboa 270) (< rumboa 360))
(setq rumbo1 (/ (* (- 360 rumboa ) PI) 180 ))
(setq perforacion1 (getint "dime la perforacion1:"))
(setq h (* perforacion1 (cos (/ (* pendiente1 PI) 180))))
(setq z (* perforacion1 (sin (/ (* pendiente1 PI) 180))))
(setq techo1 (list (+ pax (* h (cos rumbo1))) (- pay (* h (sin rumbo1))) (- paz z)))
(command "_line" pa techo1 "")
(setq perforacion2 (getint "dime la perforacion2:"))
(setq h1 (* perforacion2 (cos (/ (* pendiente1 PI) 180))))
(setq z1 (* perforacion2 (sin (/ (* pendiente1 PI) 180))))
(setq muro1 (list (+ pax (* h1 (cos rumbo1))) (- pay (* h1 (sin rumbo1))) (- paz z1)))
(command "_line" pa muro1 "")
)
( ( and (= rumboa 0) ( = 0 0) )
(setq perforacion1 (getint "dime la perforacion1:"))
```

```
(setq h (* perforacion1 (cos (/ (* pendiente1 PI) 180))))
(setq z (* perforacion1 (sin (/ (* pendiente1 PI) 180))))
(setq techo1 (list (+ pax h ) (+  pay  0 ) (- paz z)))
(command "_line" pa techo1 "")
(setq perforacion2 (getint "dime la perforacion2:"))
(setq h1 (* perforacion2 (cos (/ (* pendiente1 PI) 180))))
(setq z1 (* perforacion2 (sin (/ (* pendiente1 PI) 180))))
(setq muro1 (list (+ pax  h1 ) (+ pay 0) (- paz z1)))
(command "_line" pa muro1 "")
)
( ( and ( = rumboa 90)( = 90 90 ) )
(setq perforacion1 (getint "dime la perforacion1:"))
(setq h (* perforacion1 (cos (/(* pendiente1 PI) 180))))
(setq z (* perforacion1 (sin (/(* pendiente1 PI) 180))))
(setq techo1 (list ( + pax 0 ) (+ pay h ) (- paz z)))
(command "_line" pa techo1 "")
(setq perforacion2 (getint "dime la perforacion2:"))
(setq h1 (* perforacion2 (cos (/ (* pendiente1 PI) 180))))
(setq z1 (* perforacion2 (sin (/ (* pendiente1 PI) 180))))
(setq muro1 (list ( + pax 0) (+ pay h1 ) (- paz z1)))
(command "_line" pa muro1 "")
)
( (and (= rumboa 180)( = 180 180) )
(setq perforacion1 (getint "dime la perforacion1:"))
(setq h (*  perforacion1 (cos (/ (* pendiente1 PI) 180))))
(setq z (*  perforacion1 (sin (/ (* pendiente1 PI) 180))))
(setq techo1 (list ( - pax h ) (+ pay 0 ) (- paz z)))
(command "_line" pa techo1 "")
(setq perforacion2 (getint "dime la perforacion2:"))
(setq h1 (* perforacion2 (cos (/ (* pendiente1 PI) 180))))
(setq z1 (* perforacion2 (sin (/ (* pendiente1 PI) 180))))
(setq muro1 (list (- pax h1) ( + pay 0  ) (- paz z1)))
```

```
(command "_line" pa muro1 "")
)
( (and (= rumboa 270)(= 270 270))
(setq perforacion1 (getint "dime la perforacion1:"))
(setq h (* perforacion1 (cos (/ (* pendiente1 PI) 180))))
(setq z (* perforacion1 (sin (/ (* pendiente1 PI) 180))))
(setq techo1 (list ( + pax 0 ) (- pay  h ) (- paz z)))
(command "_line" pa techo1 "")
(setq perforacion2 (getint "dime la perforacion2:"))
(setq h1 (* perforacion2 (cos (/ (* pendiente1 PI) 180))))
(setq z1 (* perforacion2 (sin (/ (* pendiente1 PI) 180))))
(setq muro1 (list (+ pax 0 ) (-  pay h1  ) (- paz z1)))
(command "_line" pa muro1 "")
)
( ( and ( = rumboa 1000) (= 1000 1000))
(setq perforacion1 (getint "dime la perforacion1:"))
(setq techo1 (list (+ pax 0  ) (+ pay 0 ) (- paz perforacion1 )))
(command "_line" pa techo1 "")
(setq perforacion2 (getint "dime la perforacion2:"))
(setq muro1 (list ( + pax 0  ) (+ pay 0 ) (- paz perforacion2)))
(commad "_line" pa muro1)
)
)
(command "color" 5)

(setq distanciax (getint "dime la distancia x :"))
(setq distanciay (getint "dime la distancia y :"))
(setq distanciaz (getint "dime la distancia Z :"))
(setq pb (list (+ pax distanciax) (+ pay distanciay) (+ paz distanciaz)))
(setq pbx (car pb))
(setq pby (car(cdr pb)))
(setq pbz (caddr pb))
(setq pendiente2 (getint "dime la pendiente del pozo b :"))
```

```
(setq rumbob (getint "dime el rumbo del pozo b:") )
( cond
(   (and (> rumbob 0) (< rumbob 90) )
(setq rumbo2 (/ (* rumbob PI) 180 ))
(setq perforacion3 (getint "dime la perforacion3:"))
(setq x (* perforacion3 (cos (/ (* pendiente2 PI) 180))))
(setq k (* perforacion3 (sin (/ (* pendiente2 PI) 180))))
(setq techo2 (list (+ pbx (* x (cos rumbo2))) (+ pby (* x (sin rumbo2))) (- pbz k)))
(command "_line" pb techo2 "")
(setq perforacion4 (getint "dime la perforacion4:"))
(setq x1 (* perforacion4 (cos (/ (* pendiente2 PI) 180))))
(setq k1 (* perforacion4 (sin (/ (* pendiente2 PI) 180))))
(setq muro2 (list (+ pbx (* x1 (cos rumbo2))) (+ pby (* x1 (sin rumbo2))) (- pbz k1)))
(command "_line" pb muro2 "")
)


(          ( and (> rumbob 90) (< rumbob 180 ) )
(setq rumbo2 (/ (* (- 180 rumbob) PI) 180 ))
(setq perforacion3 (getint "dime la perforacion3:"))
(setq x (* perforacion3 (cos (/ (* pendiente2 PI) 180))))
(setq k (* perforacion3 (sin (/ (* pendiente2 PI) 180))))
(setq techo2 (list (- pbx (* x (cos rumbo2))) (+ pby (* x (sin rumbo2))) (- pbz k)))
(command "_line" pb techo2 "")
(setq perforacion4 (getint "dime la perforacion4:"))
(setq x1 (* perforacion4 (cos (/ (* pendiente2 PI) 180))))
(setq k1 (* perforacion4 (sin (/ (* pendiente2 PI) 180))))
(setq muro2 (list (- pbx (* x1 (cos rumbo2))) (+ pby (* x1 (sin rumbo2))) (- pbz k1)))
(command "_line"  pb muro2 "")
)
( (and ( > rumbob 180 ) ( < rumbob 270 ) )
(setq rumbo2 (/ (* (- 270 rumbob ) PI) 180 ))
(setq perforacion3 (getint "dime la perforacion3:"))
```

```
(setq x (* perforacion3 (cos (/ (* pendiente2 PI) 180))))
(setq k (* perforacion3 (sin (/ (* pendiente2 PI) 180))))
(setq techo2 (list (- pbx (* x (sin rumbo2))) (- pby (* x (cos rumbo2))) (- pbz k)))
(command "_line" pb techo2 "")
(setq perforacion4 (getint "dime la perforacion4:"))
(setq x1 (* perforacion4 (cos (/ (* pendiente2 PI) 180))))
(setq k1 (* perforacion4 (sin (/ (* pendiente2 PI) 180))))
(setq muro2 (list (- pbx (* x1 (sin rumbo2))) (- pby (* x1 (cos rumbo2))) (- pbz k1)))

(command "_line" pb muro2 "")
)
( ( and ( > rumbob 270) (< rumbob 360) )
(setq rumbo2 (/ (* (- 360 rumbob) PI) 180 ))
(setq perforacion3 (getint "dime la perforacion3:"))
(setq x (* perforacion3 (cos (/ (* pendiente2 PI) 180))))
(setq k (* perforacion3 (sin (/ (* pendiente2 PI) 180))))
(setq techo2 (list (+ pbx (* x (cos  rumbo2))) (- pby (* x (sin rumbo2))) (- pbz k)))
(command "_line" pb techo2 "")
(setq perforacion4 (getint "dime la perforacion4:"))
(setq x1 (* perforacion4 (cos (/ (* pendiente2 PI) 180))))
(setq k1 (* perforacion4 (sin (/ (* pendiente2 PI) 180))))
(setq muro2(list (+ pbx (* x1 (cos rumbo2))) (- pby (* x1 (sin rumbo2))) (- pbz k1)))
(command "_line" pb muro2 "")
)
( ( and ( = rumbob 0) (= 0 0) )
(setq perforacion3 (getint "dime la perforacion3:"))
(setq x (* perforacion3 (cos (/ (* pendiente2 PI) 180))))
(setq k (* perforacion3 (sin (/ (* pendiente2 PI) 180))))
(setq techo2 (list (+ pbx  x ) (+ pby 0 ) (- pbz k)))
(command "_line" pb techo2 "")
(setq perforacion4 (getint "dime la perforacion4:"))
(setq x1 (* perforacion4 (cos (/ (* pendiente2 PI) 180))))
(setq k1 (* perforacion4 (sin (/ (* pendiente2 PI) 180))))
```

```
(setq muro2 (list (+ pbx  x1 ) ( + pby 0) (- pbz k1)))
(command "_line" pb muro2 "")
)
( ( and (= rumbob 90 ) (= 90 90) )
(setq perforacion3 (getint "dime la perforacion3:"))
(setq x (* perforacion3 (cos (/ (* pendiente2 PI) 180))))
(setq k (* perforacion3 (sin (/ (* pendiente2 PI) 180))))
(setq techo2 (list ( + pbx 0 ) ( + pby  x ) (- pbz k)))
(command "_line" pb techo2 "")
(setq perforacion4 (getint "dime la perforacion4:"))
(setq x1 (* perforacion4 (cos (/ (* pendiente2 PI) 180))))
(setq k1 (* perforacion4 (sin (/ (* pendiente2 PI) 180))))
(setq muro2 (list ( + pbx 0 ) (+ pby x1 ) (- pbz k1)))
(command "_line" pb muro2 "")
)
( ( and ( = rumbob 180 ) ( = 180 180) )
(setq perforacion3 (getint "dime la perforacion3:"))
(setq x (* perforacion3 (cos (/ (* pendiente2 PI) 180))))
(setq k (* perforacion3 (sin (/ (* pendiente2 PI) 180))))
(setq techo2 (list ( - pbx x  ) ( + pby 0 ) (- pbz k)))
(command "_line" pb techo2 "")
(setq perforacion4 (getint "dime la perforacion4:"))
(setq x1 (* perforacion4 (cos (/ (* pendiente2 PI) 180))))
(setq k1 (* perforacion4 (sin (/ (* pendiente2 PI) 180))))
(setq muro2 (list ( - pbx x1 ) ( + pby 0 ) (- pbz k1)))

(command "_line" pb muro2 "")
)
( (and (= rumbob 270) ( = 270 270) )
(setq perforacion3 (getint "dime la perforacion3:"))
(setq x (* perforacion3 (cos (/ (* pendiente2 PI) 180))))
(setq k (* perforacion3 (sin (/ (* pendiente2 PI) 180))))
(setq techo2 (list ( + pbx  0 ) ( - pby x ) (- pbz k)))
```

```
(command "_line" pb techo2 "")
(setq perforacion4 (getint "dime la perforacion4:"))
(setq x1 (* perforacion4 (cos (/ (* pendiente2 PI) 180))))
(setq k1 (* perforacion4 (sin (/ (* pendiente2 PI) 180))))
(setq muro2 (list ( + pbx 0 ) ( - pby  x1 ) (- pbz k1)))
(command "_line" pb muro2 "")
)
( (and ( = rumbob 1000) ( = 1000 1000))
(setq perforacion3 (getint "dime la perforacion3:"))
(setq techo2 (list ( + pbx 0 ) ( + pby 0 ) (- pbz perforacion3)))
(command "_line" pa techo1 "")
(setq perforacion4 (getint "dime la perforacion4:"))
(setq muro2 (list (+ pbx 0 ) ( + pby 0 ) (- pbz perforacion4)))
(commad "_line" pa muro1)
)
)



(setq muro1x (car muro1))
(setq muro1y (car(cdr muro1)))
(setq muro1z (caddr muro1))
(setq muro2x (car muro2))
(setq muro2y (car(cdr muro2)))
(setq muro2z (caddr muro2))
(setq techo1x (car techo1))
(setq techo1y (car (cdr techo1)))
(setq techo1z (caddr techo1))
(setq techo2x (car techo2))
(setq techo2y (car (cdr techo2)))
(setq techo2z (caddr techo2))
(command "color" 8)
(command "_line" muro2 muro1 "")
(setq objetolinea (entlast))
```

```
(command "_move" objetolinea "" muro1 techo2 "")
(setq techo3 (list (+ techo2x (- muro2x muro1x)) (+ techo2y (- muro2y muro1y)) (+ techo2z (- muro2z muro1z)) ) )
(setq techo3x (car techo3))
(setq techo3y (car (cdr techo3)))
(setq techo3z (caddr techo3))
(command "_3dface"  techo2 techo3 techo1 techo2 "" )
(setq objetotecho (entlast))
(command "_copy" objetotecho "" techo2 muro1 "")
(command "scp" "objeto" objetotecho)
(setq puntoaux (trans muro1 0 1))
(command "scp" "univ")
(setq potenciadelpozo (caddr puntoaux))
(command "color" 1)
(cond
( (and    ( > techo2z techo3z  )   ( < techo2z techo1z))
(setq puntoplano1 (list (+ 0 0) (+ 0 0) (- techo2z 0)))
(setq puntoplano2 (list (+ 0 1000) (+ 0 1000) (- techo2z 0) ))
(command "rectang" puntoplano1 puntoplano2 "")
(setq objetoplano (entlast))
(setq puntoplano3 (list (+ 0 0) (+ 0 1000) (- techo2z 0)))
(command "_line" puntoplano1 puntoplano3 "")
(setq objetolineaplano (entlast))
(command "superfplana" "objeto" objetoplano  "")
(setq superficieplana1 (entlast))
(command "superfplana" "objeto" objetotecho "")
(setq superficieplana2 (entlast))
(command "intersec" superficieplana1 superficieplana2 "")
(setq objetointerseccion (entlast))
)
( (and ( < techo2z techo3z ) ( > techo2z techo1z) )
(setq puntoplano1 (list (+ 0 0) (+ 0 0) (- techo2z 0)))
(setq puntoplano2 (list (+ 0 1000) (+ 0 1000) (- techo2z 0) ))
(command "rectang" puntoplano1 puntoplano2 "")
```

```
(setq objetoplano (entlast))
(setq puntoplano3 (list (+ 0 0) (+ 0 1000) (- techo2z 0)))
(command "_line" puntoplano1 puntoplano3 "")
(setq objetolineaplano (entlast))
(command "superfplana" "objeto" objetoplano  "")
(setq superficieplana1 (entlast))
(command "superfplana" "objeto" objetotecho "")
(setq superficieplana2 (entlast))
(command "intersec" superficieplana1 superficieplana2 "")
(setq objetointerseccion (entlast))
)
( (and (> techo1z techo2z ) (< techo1z techo3z))
(setq puntoplano1 (list (+ 0 0) (+ 0 0) (- techo1z 0)))
(setq puntoplano2 (list (+ 0 1000) (+ 0 1000) (- techo1z 0) ))
(command "rectang" puntoplano1 puntoplano2 "")
(setq objetoplano (entlast))
(setq puntoplano3 (list (+ 0 0) (+ 0 1000) (- techo1z 0)))
(command "_line" puntoplano1 puntoplano3 "")
(setq objetolineaplano (entlast))
(command "superfplana" "objeto" objetoplano  "")
(setq superficieplana1 (entlast))
(command "superfplana" "objeto" objetotecho "")
(setq superficieplana2 (entlast))
(command "intersec" superficieplana1 superficieplana2 "")

(setq objetointerseccion (entlast))
)
( (and (< techo1z techo2z) (> techo1z techo3z))
(setq puntoplano1 (list (+ 0 0) (+ 0 0) (- techo1z 0)))
(setq puntoplano2 (list (+ 0 1000) (+ 0 1000) (- techo1z 0) ))
(command "rectang" puntoplano1 puntoplano2 "")
(setq objetoplano (entlast))
```

```
(setq puntoplano3 (list (+ 0 0) (+ 0 1000) (- techo1z 0)))
(command "_line" puntoplano1 puntoplano3 "")
(setq objetolineaplano (entlast))
(command "superfplana" "objeto" objetoplano  "")
(setq superficieplana1 (entlast))
(command "superfplana" "objeto" objetotecho "")
(setq superficieplana2 (entlast))
(command "intersec" superficieplana1 superficieplana2 "")
(setq objetointerseccion (entlast))
)
( (and (> techo3z techo2z)( < techo3z techo1z) )
(setq puntoplano1 (list (+ 0 0) (+ 0 0) (- techo3z 0)))
(setq puntoplano2 (list (+ 0 1000) (+ 0 1000) (- techo3z 0) ))
(command "rectang" puntoplano1 puntoplano2 "")
(setq objetoplano (entlast))
(setq puntoplano3 (list (+ 0 0) (+ 0 1000) (- techo3z 0)))
(command "_line" puntoplano1 puntoplano3 "")
(setq objetolineaplano (entlast))
(command "superfplana" "objeto" objetoplano  "")
(setq superficieplana1 (entlast))
(command "superfplana" "objeto" objetotecho "")
(setq superficieplana2 (entlast))
(command "intersec" superficieplana1 superficieplana2 "")
(setq objetointerseccion (entlast))
)
( (and (< techo3z techo2z) (> techo3z techo1z) )
(setq puntoplano1 (list (+ 0 0) (+ 0 0) (- techo3z 0)))
(setq puntoplano2 (list (+ 0 1000) (+ 0 1000) (- techo3z 0) ))
(command "rectang" puntoplano1 puntoplano2 "")
(setq objetoplano (entlast))
(setq puntoplano3 (list (+ 0 0) (+ 0 1000) (- techo3z 0)))
(command "_line" puntoplano1 puntoplano3 "")
(setq objetolineaplano (entlast))
```

```
(command "superfplana" "objeto" objetoplano  "")
(setq superficieplana1 (entlast))
(command "superfplana" "objeto" objetotecho "")
(setq superficieplana2 (entlast))
(command "intersec" superficieplana1 superficieplana2 "")
(setq objetointerseccion (entlast))
)
)
(command "_erase" superficieplana1 objetoplano "")

(command "scp" "ejez" "objeto" objetointerseccion )
( setq L1 (list objetointerseccion))
( setq      LW1 (entget (car L1)) )
( setq      P11 (cdr (assoc 10 LW1)) )
( setq     P21 (cdr (assoc 11 LW1)) )
( setq     W1 (angle P11 P21) )
( setq   L2 (list objetolineaplano) )
( setq    LW2 (entget (car L2)) )
( setq   P12 (cdr (assoc 10 LW2)))
( setq   P22 (cdr (assoc 11 LW2)) )
( setq     W2 (angle P12 P22))
( setq    W (abs (- W1 W2)))
(if (> W pi)(setq W (- W pi)))
(setq W (* W (/ 180 pi))
WC (- 180 W)
A (min W WC)
C (max W WC))
(setq punto100 (trans techo1 0 1))
(setq punto100x (car punto100))
(setq punto100y(car (cdr punto100)))
(setq punto100z (caddr punto100))
(setq punto300(trans techo3 0 1))
(setq punto300x(car punto300))
```

```
(setq punto300y(car (cdr punto300)))

(setq punto300z (caddr punto300))
(setq x ( - punto300y punto100y ))
(setq y ( - punto300x punto100x ))
(setq pendiente (atan x y ))
(command "scp" "univ")
(setq pendientedelpozo (/ (* 180 pendiente) PI))
(prompt " potencia del pozo = ")
(prin1 (ABS potenciadelpozo))
(prompt " pendiente del pozo = ")
(prin1 (ABS pendientedelpozo) )
(prompt " Rumbo = ")
(prompt (rtos A 2 4))
(prompt "  ** 180 - rumbo = ")
(prompt (rtos C 2 4))
(princ)
)
```

Lista de referencias

1. “Manual de AutoCAD 2d y 3d “

Universidad de oriente núcleo de Anzoátegui departamento de ingeniería y ciencias aplicadas departamento de ingeniería industrial.

Autor: Josep Stalín loján paladínes

2.” Elementos mecánicos”

Empresa Ternium

Revisado por Ricardo Sepúlveda

3. “programación grafica: AutoLISP”

Autor: José Gustavo barros g.

4. “Manual de referencia rápida de lisp”

Universidad de concepción facultad de ingeniería departamento de ingeniería informática y ciencias de la computación.

Autor: Rolando Burgos Cárdenas

5.” Programación AutoLISP (personalización de AutoCAD)”

Autores: Milagros canga Villegas y José Andrés Díaz Severiano

6. Elementos de maquinas ii “Elementos de unión roscados” Mérida 2010

Autor: desconocido

7. PowerPoint “Elementos básicos de diseño mecánico: de unión desmontable y de unión fija.”

Autor: desconocido

8. http://www.afralisp.net/reference/autolisp-functions.php

9. http://www.arquba.com/rutinas-lisp-para-autocad/

10. http://www.cadenlinea.com/autolisp.htm

11. http://www.togores.net/vl/curso/lisp/introduccion

12. Curso AutoLISP 01. Introducción y operaciones matemáticas

https://www.youtube.com/watch?v=tgbquikyoqo

13. Curso AutoLISP 02. Funciones relacionales y operadores lógicos

https://www.youtube.com/watch?v=l3n-otyoek0

14. Curso AutoLISP 03. Variables

https://www.youtube.com/watch?v=tnd0lz6_lwy

15. Curso AutoLISP 04. Captura de datos e initget

https://www.youtube.com/watch?v=wkni3e2_acm

16. Curso AutoLISP 05. Estructuras de programación

https://www.youtube.com/watch?v=0vuny6bdcyy

17. Curso AutoLISP 06. Estructuras de programación ii.

https://www.youtube.com/watch?v=c2gsx-qp87a

18. Curso AutoLISP 07. Crear y manejar listas.

https://www.youtube.com/watch?v=ct96lxyd3ys

19. Curso AutoLISP 08. Conversión de datos.

https://www.youtube.com/watch?v=zoiokwowmbm

20. Curso AutoLISP 09. Cadenas de texto, ángulos y distancias.

https://www.youtube.com/watch?v=snemywb4clc

21. Curso AutoLISP 10. Acceso base de datos y edición de listas

https://www.youtube.com/watch?v=4idhkikslp0

22. Curso AutoLISP 11. Crear entidades desde código y conjuntos de selección ssget

https://www.youtube.com/watch?v=shp60-mpeaq

23. Curso AutoLISP 12. Rutinas finales i.

https://www.youtube.com/watch?v=2mtnl8ykxls

24. Curso AutoLISP 13. Rutinas finales ii.

https://www.youtube.com/watch?v=5c_zr1bond0

25. Curso AutoLISP 14. Rutinas finales iii y xdata.

https://www.youtube.com/watch?v=vxb2cr7f5uq

26. Curso AutoLISP 15. Entorno VisualLISP.

https://www.youtube.com/watch?v=7zpoefwteg0