Chapter

5

Input and String Handling Functions

In this chapter, you will learn how to use the following AutoLISP functions to World Class standards:

1. **Placing Text on Drawings as Easy as Lines and Circles**
2. **Starting the Notemaker Code by Launching the Visual LISP Editor**
3. **Saving the Object Snap Settings and Then Turning Them Off**
4. **Using Getpoint to Obtain a Point on the Graphical Display**
5. **Using Getreal to Obtain a Real Number from the Keyboard**
6. **Using Getstring to Obtain Text from the Keyboard**
7. **Using Getint to Obtain an Integer from the Keyboard**
8. **Using Initget to Setup Keywords for the Getkword Function**
9. **Using Getkword to Obtain Text from the Keyboard**
10. **Using the If Function to Make Decisions in a Program**
11. **Using Strcat to Concatenate Two or More Text Strings**
12. **Using Strcase to Change the Case Size of a Text String**
13. **Using Command "Text" to Insert Notes**
14. **Ending the Program**
15. **Saving the Program**
16. **Loading the Program**

`Placing Text on Drawings as Easy as Lines and Circles

On many drawings, we can see standard text which describes the part's material, surface finish, and protection coating from corrosion, coating color and texture and the tolerance range. Many organizations placed their notes in a common area and use standard phrasing which makes the manufacturing process easier for the technician or construction worker since they can locate the information and they are used to the terminology. The Visual AutoLISP programming method will work well determining and creating text for any working drawing. As with previous coding, we will ask the user a few questions and based upon their answers, we will formulate a series of numbered notes. Again using a program will present the designer with inquiries that need to be answered, so when we write the code during slower times of the engineering production year, we can use that potentially saved time stored in our extremely fast code to make drawings faster at our company in the peak segment of their work calendar.

In this exercise, we will create a program called Notemaker. The program will create standard text similar to the notes we have seen in previous textbooks, World Class CAD, Fundamentals of 2D Drawing and Fundamentals of 3D Drawing. The note we will place on the drawing or as follows:

Notes:
1. **Materials: aluminum**
2. **Remove all sharp edges and burrs.**
3. **Paint with one coat of primer and with 1 coat(s) of flat white enamel.**
4. **Tolerances unless otherwise specified:**
   Fraction: ±1/16
   1 decimal: ±0.06
   2 decimal: ±0.03
   3 decimal: ±0.010
   Angular: ±0.5º

Figure 5.1 – Standard Notes on a Part Drawing

The questions we will ask concentrate on the name of the material from which the part will be made. The company does not ask a question for the second note, since all parts will have sharp edges or imperfections removed. Another is the number of coats of enamel, the color of the enamel and the texture of the enamel. After those questions, we will prompt the designer whether they wish to use the standard tolerances or input their own. We are showing the standard tolerances in Figure 5.1. If we do not want the standard tolerances, we will have a prompt for fraction, 1 decimal, 2 decimal, 3 decimal and angular tolerances. An option like this shows the adaptability in the code, especially when we use a little imagination. The **if** function will enable us to toggle between the standard and custom notes for tolerances.

At the beginning of the notemaker.LSP program, a modified form of the Construction code, we will place our regular series of comments, which are the program name, explanation and copyright information. Next will be the alert message which gives pertinent information from the code lighter and the keyboard characters to execute the program.

As with the first eight Construction code exercises, we need to capture the Objects Snap settings from the user's drawing, store the settings and in a variable called OSM and then turn the Object Snap settings off. This will keep any Object Snap such as Endpoint, Center or Midpoint from interfering with the placement of the notes when the programming is running automatically. The Notemaker program will follow all the steps of the Construction code process except Step 4, Do the Math. We will not have to compute any complex geometry to insert the notes, so we will skip this section.

Step 1	**Start the program**
Step 2	**Drawing setup**
Step 3	**User input**
Step 4	**Do the math**
Step 5	**Point assignments**
Step 6	**Lets draw**
Step 7	**End the program**

Almost every drawing made will have text in the form of notes, in the title block, bill of materials and revision block. This first program which handles text strings only requires the user to select a single point and then every other line of text is referenced from the first point. In future routines that place text in a drawing, the placement of each word or phrase will be even more precise, such aligning the wording in between lines of the title block. So this code is just as important to a designer as drawing lines and dimensions. Now let us start to build the program, notemaker.LSP.

Starting the Notemaker Code by Launching the Visual LISP Editor

Open the Visual LISP Editor and on the first line type the comment

```
;;;    notemaker.lsp
```

The program name is always on the first line of the code. The semicolons cause the statement to become a comment so the line of code will not be read.

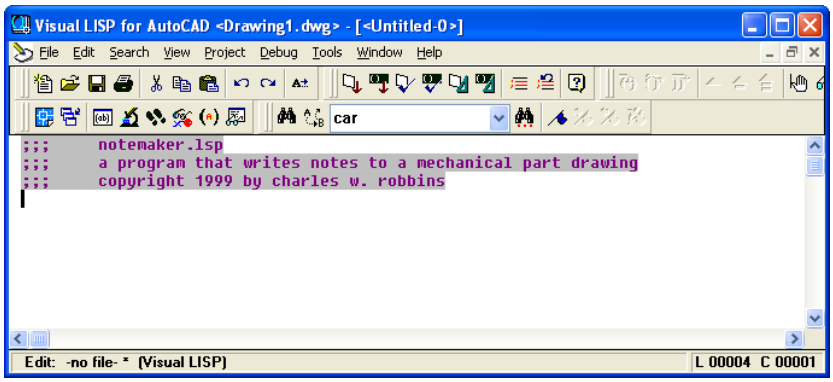

Figure 5.2 – Starting the Notemaker Program

The next comments in the program will be the details concerning what the routine will do. In this program, there are comments after almost every line of code.

Next we will create an AutoCAD Message by taking the information listed in the comments and placing the text in the **alert** function. On the first line of the alert expression, the program and the copyright information is keyed.

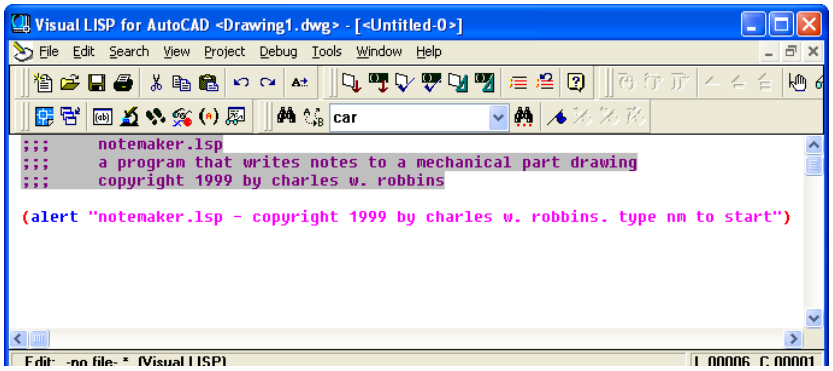

Figure 5.3 – Adding the Alert Expression

Add a new comment

;;; start the program

Then we start the program with the **defun** function, which means define function. Begin with the open parenthesis then **defun,** then a **c:** which will allow the program to run on the AutoCAD command line.

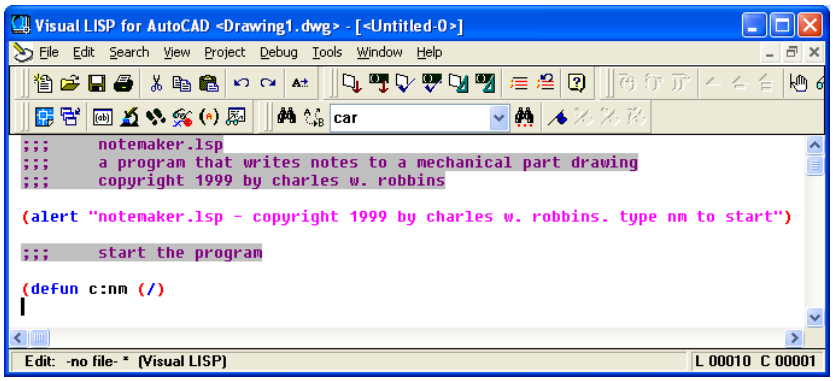

Figure 5.4 – The Defun Expression

Practice typing the following examples of the **alert** function at the command line of AutoCAD.

Function	Name	Description
alert	**AutoCAD Message**	**The alert function will create an AutoCAD message window appear on the graphical display with an OK button to close the message window.**
Examples		
At the beginning of the program	**(alert "notemaker.lsp - copyright 1999 by charles w. robbins. type nm to start")**	Window appears on the graphical display
As an error prompt	**(alert "Error: Type units in inches")**	Window appears on the graphical display

Next type **nm** which will be the execution symbol to start the program. Keep in mind the alert message that stated "type nm to start". The alert message text and the **defun** symbol must match. The open and closed parenthesis "()"following the **nm** enclosing nothing means there will not be any defined arguments or local variables for this program. After that, we need to make changes to the AutoCAD System Variables that may interfere with the running of the code and automatically drawing the lines and arcs perfectly.

Practice typing the following examples of the **defun** function at the command line of AutoCAD.

Function	Name	Description
defun	**Define Function**	**The define function leads off the beginning of the program**
Examples		
Place a **c:** in front of the program, hello. Allows **hello** to be typed at the keyboard to execute the code	**(defun c:hello (/) (print "Hi Ya All"))**	Answer: **C:HELLO** Type: **hello** Returns: **"Hi Ya All"**
When the code is used inside another program, do not place the c: in front of the program name	**(defun hello (/) (print "Hi Ya All"))**	Answer: **HELLO** Type: **(hello)** Returns: **"Hi Ya All"**

Saving the Object Snap Settings and then Turning Them Off

In the next section of the code, we will turn off the drawing Object Snaps so they cannot possibly interfere with the insertion of the drawing notes. In order to accomplish this task, we you need to understand the **getvar** and the **setvar** functions. The **getvar** function will obtain a drawing setting, so we can save the number or text string for future use. The **setvar** function will allow us to change a system variable, like turning off the Object Snaps.

Start with a new comment

;;; drawing setup

And type the code

(setq osm (getvar "osmode"))

; gets osnap settings and assigns to osm

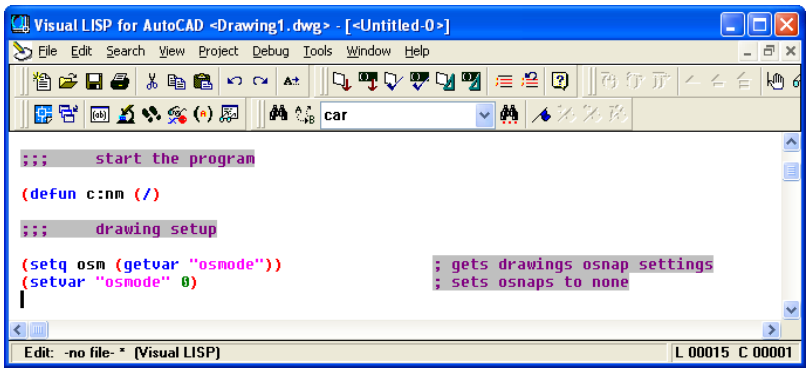

Figure 5.5 – Saving and Turning Off Object Snaps

Next, we will turn off the drawing's object snaps by setting the system variable **"osmode"** to **0** using this line of code. Add the comment as shown.

(setvar "osmode" 0) ; turns osnap settings off

Let's talk about the expression, **(setq osm (getvar "osmode"))**. The function **setq** means set quotient and we will use the function to create a variable **osm** which stands for object snap mode, a variable name that we just made up. The variable **osm** will hold the integer

representing the "osmode" system variable's setting. To get the number use the function **getvar** followed by the name of system variable inside a set of quotes.

To turn off a system variable in many cases in setting the variable to zero. In the expression, **(setvar "osmode" 0)**, the function **setvar** followed by a system variable inside a set of quotes like **"osmode"** then a **0** will result in turning off the Object Snap settings.

Practice typing the following examples of the **setq, getvar** and **setvar** functions at the command line of AutoCAD.

Function	Name	Description
setq	**Set Quotient**	**Allows the user to assign a real number, integer, string or list to a variable**
Examples		
Set the variable **a** the text string **World Class CAD**	**(setq a "World Class CAD")**	Answer: **"World Class CAD"**
Set the variable **counter** the integer **0**	**(setq counter 0)**	Answer: **0**
Set the text height variable **txtht** the real number **0.125**	**(setq txtht 0.125)**	Answer: **0.1250**
Set the point variable **sp** the list of 0,0,0	**(setq sp (list 0.0 0.0 0.0))**	Answer: **(0,0,0)**

Function	Name	Description
getvar	**Get a variable**	**Allows the user to obtain a system variable setting from an AutoCAD drawing**
Examples		
Turn on the endpoint, midpoint, quadrant, intersection and perpendicular Object Snaps	**(setq osm (getvar "osmode"))**	Answer: **179**
Get the AutoCAD version number	**(setq osm (getvar "acadver"))**	Answer: **"16.2s (LMS Tech)"**

Function	Name	Description
setvar	**Get a variable**	**Allows the user to obtain a system variable setting from an AutoCAD drawing**
Examples		
Turn off the Object Snaps	**(setvar "osmode" 0))**	Answer: **0**

Using Getpoint to Obtain a Point on the Graphical Display

In the User Input section of the Construction Code, we need to expand into new areas besides just requesting the starting point and the getting a measurement using the **getreal** function as we did in the first eight programs.

The first function we will examine together is **getpoint**. This tool will allow the program user to select a point on the graphical display with their mouse. Following **getpoint** is a text string usually written is a commanding or questioning format.

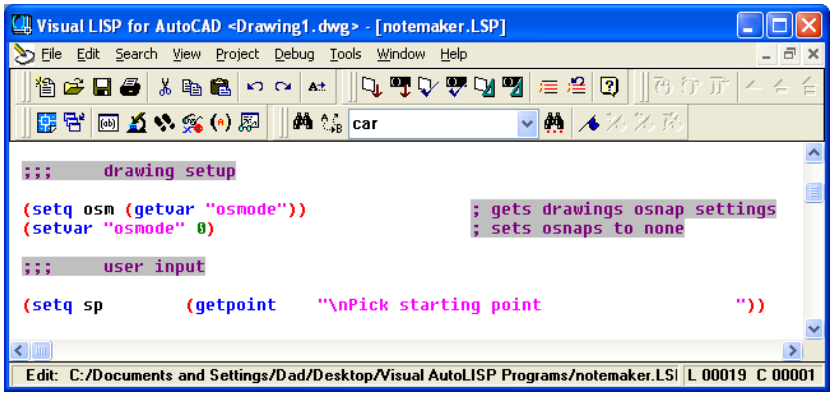

Figure 5.6 – Using the Getpoint Function

The user input of selecting a point begins with a comment.

;;; user input

Then type the following code:

(setq sp (getpoint "\nPick the starting point "))

We use the **setq** expression to assign the three point list (X, Y and Z) to the variable **sp** representing the starting point. After the function **getpoint**, a programmer has the option, in which we have chosen, to add a line of text prompting the user to **"Pick the starting point"** and we also modified the prompt in a small way. Notice that in front of the capital P in the word Pick, a "\n" is added. That will place the command **"Pick the starting point"** without containing those two characters to start on a new command line in the AutoCAD program. Placing command statements or questions on a new command line allows for a cleaner look to the user when following a command or answering the question.

Periodically we will work at an organization that wants their notes in the exact location of their drawing. When we face a programming problem such as this the starting point expression will change. First instance, let us make believe that the notes at this company start at the X and Y coordinates 14, 10. Then in this note making code, we would change the starting point expression to:

(setq sp (list 14 10 0))

This will place the beginning of the notes in an exact position for every occasion.

Practice typing the following examples of the **setq** and **getpoint** functions at the command line of AutoCAD.

Function	Name	Description
setq	**Set Quotient**	**Allows the user to assign a real number, integer, string or list to a variable**
Examples		
Set the variable **a** the text string **World Class CAD**	**(setq a "World Class CAD")**	Answer: **"World Class CAD"**
Set the variable **counter** the integer **0**	**(setq counter 0)**	Answer: **0**
Set the text height variable **txtht** the real number **0.125**	**(setq txtht 0.125)**	Answer: **0.1250**
Set the point variable **sp** the list of 0,0,0	**(setq sp (list 0.0 0.0 0.0))**	Answer: **(0,0,0)**

Function	Name	Description
getpoint	**Get a Point**	**Allows the user to obtain a point on the graphical display by selecting with a mouse**
Examples		
Get a starting point	**(setq sp (getpoint "\nPick starting point"))**	Answer: **Pick starting point** Then select a point and the will return a list like: **(30.471 28.4052 0.0)**

Using Getreal to Obtain a Real Number from the Keyboard

To ask the question, "What is the text height", we will use the **getreal** function. We use **getreal** to allow the LISP program user to type a number containing decimals with their keyboard. The **getreal** expression is set within the **(setq txtht …….)** code.

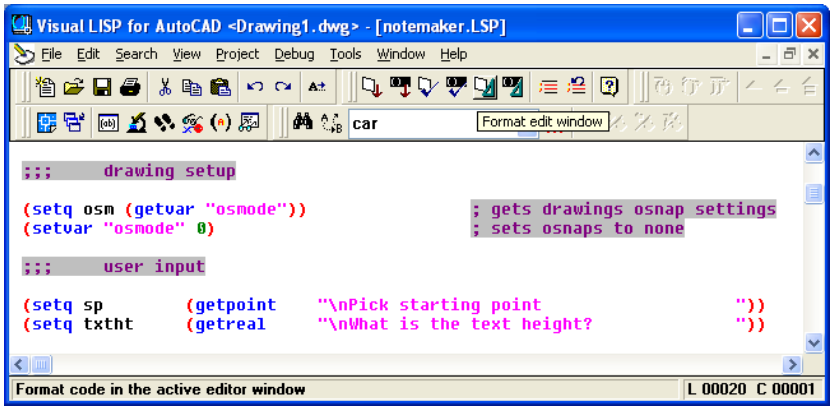

Figure 5.7 – Using the Getreal Function

So type the following compound expression:

(setq txtht (getreal "\nWhat is the text height))

The information that the user types with the keyboard is stored in the variable name **txtht**. We will never pick a variable name that matches an AutoCAD command.

Whenever we are not quite sure whether the answer is going to be a whole number or a decimal, we will use the **getreal** function. Using another function which will only allow whole numbers will never allow the acceptance of a decimal.

If you look at the Visual LISP Editor in Figure 5.7, you will notice that we dressed the last two expressions so that the questions line up perfectly. You will pick up on this characteristic when the program is running and the typed answers to the questions line up neatly.

Practice typing the following examples of the **getreal** function at the command line of AutoCAD.

Function	Name	Description
getreal	**Get a Real Number**	**Allows the user to obtain a real number by allowing the user to type at the keyboard**
Examples		
Get a number	**(setq txtht (getreal "\nWhat is the text height?"))**	Answer: **What is the text height?** Then type: **0.125** **3.2**
Ask for a number, user types a whole number and the reply is changed to a real number	**(setq txtht (getreal "\nWhat is the text height?"))**	Answer: **What is the text height?** Then type: **1** **1.0**
Ask for a number, user types a fraction and the reply is changed to a real number	**(setq txtht (getreal "\nWhat is the text height?"))**	Answer: **What is the text height?** Then type: **1/8** **0.125**

Using Getstring to Obtain a Text from the Keyboard

When we want the user to type a text string like the type of material at the command line, we use the **getstring** function. This tool allows the program user to type any characters with their keyboard, and in our note making program scenario, we need words or short phrases to complete the text.

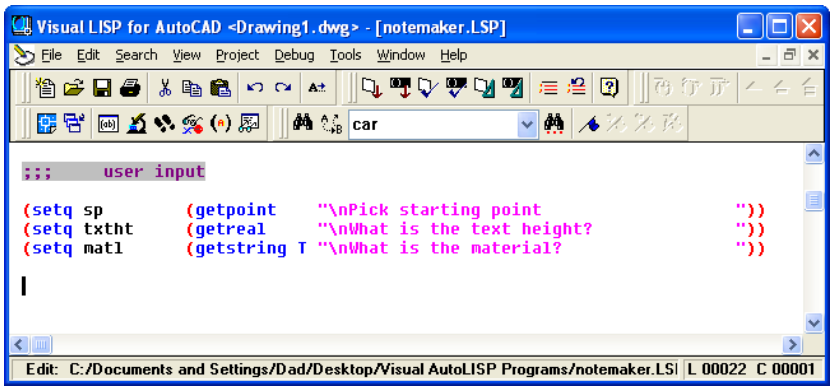

Figure 5.8 – Using the Getstring Function

So type the following compound expression:

(setq matl (getstring T "\nWhat is the material? "))

The information that the user types with the keyboard is stored in the variable name **matl**. Again the user input commands and questions are aligned in the LISP routine to allow the commands and questions to line up neatly on the command line in AutoCAD when the program is running. Aligning the expressions as shown in Figure 5.8 also gives us, the programmers the ability to easily check for a missing parenthesis.

We might wonder why there is a capital **T** following **getstring.** As we already know that when we are in the AutoCAD program, the space bar acts as an **Enter** key on the keyboard. So if we do not place the capital **T** behind **getstring** then if the material is cast iron, the user will type cast and strike the space bar, and the program will continue to the next command or question in the user input. Placing the capital **T** behind **getstring** allows the space bar to operate as a space bar, adding spaces in the text string wherever the user wants them.

Sometimes companies use the same materials because they specialize in making that single product. If our organization uses just one or two standard materials, then we would want to change the material **getstring** expression to a **getkword** coded phrase. We would identify the few materials using keywords and the getkword function will only accept those words during the user input section of the Construction code. We will learn about the **getkword** function later in this chapter.

Practice typing the following examples of the **getstring** function at the command line of AutoCAD.

Function	Name	Description
getstring	**Get a Text String**	**Allows the user to obtain a text string by allowing the user to type at the keyboard**
Examples		
Looking for a single word response	**(setq matl (getstring "\nWhat is the material?"))**	Answer: **What is the material?** Then type: **Aluminum** **"Aluminum"**
What happens when two or more words are typed	**(setq matl (getstring "\nWhat is the material?"))**	Answer: **What is the material?** Then type: **Stainless Steel** **"Stainless"**
Fix the space bar problem with a **T** after **getstring**	**(setq matl (getstring T "\nWhat is the material?"))**	Answer: **What is the material?** Then type: **Stainless Steel** **"Stainless Steel"**

Using Getint to Obtain an Integer from the Keyboard

We use **getint** to allow the LISP program user to type a whole with their keyboard. The getreal expression is set within the **(setq coats …….)** code. If the user does not type a whole number, the AutoCAD program returns with "Requires an integer value" and will repeat the original question.

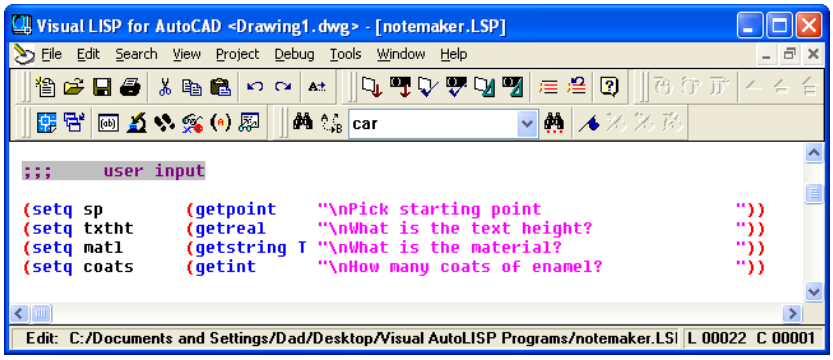

Figure 5.9 – Using a Getint Function

So type the following compound expression:

(setq coats (getint "\nHow many coats of enamel? "))

Practice typing the following examples of the **getint** function at the command line of AutoCAD.

Function	Name	Description
getint	**Get an Integer**	**Allows the user to obtain an integer by allowing the user to type at the keyboard**
Examples		
Asking a question	(setq coats (getint "\nHow many coats of enamel? "))	Answer: **How many coats of enamel?** Then type: **1** Returns: **1**
Asking a question and inputting a real number	(setq coats (getint "\nHow many coats of enamel? "))	Answer: **How many coats of enamel?** Then type: **1.0** Returns: **Requires an integer value** and **How many coats of enamel?**
Asking a question and inputting an incorrect answer	(setq coats (getint "\nHow many coats of enamel? "))	Answer: **How many coats of enamel?** Then type: **One** Returns: **Requires an integer value.** and **How many coats of enamel?**

From time to time, we will use a get function more than once in a program. In the note making routine, we need to enquire what is the enamel's color and texture, so we will use the **getstring** function two more times and assign their responses to a variable using the **setq** function.

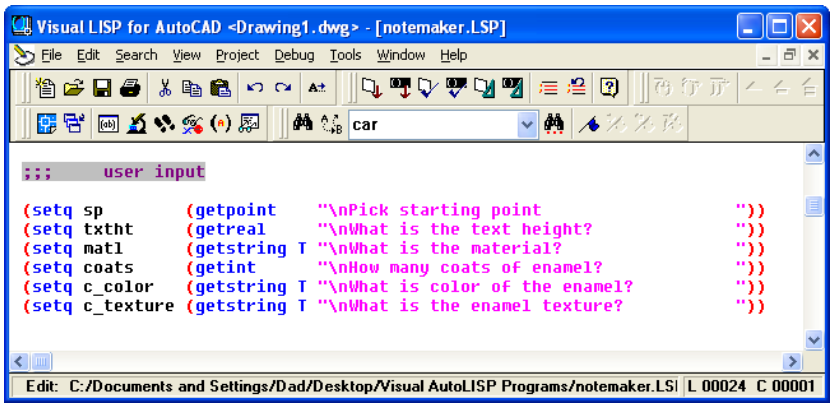

Figure 5.10 – More Getstring Functions

So type the following compound expressions:

```
(setq c_color   (getstring T "\nWhat is color of the enamel?      "))
(setq c_texture (getstring T "\nWhat is the enamel texture?      "))
```

We can use the underscore _ in variable names where we might want to place a space.

The very next expression in the program is most likely uses the **getkword** function which will use the keyword definition in the initget function. Whenever we want to change the keywords, use the **initget** function to redefine them before another get a keyword expression.

Using Initget to Setup Keywords for the Getkword Function

Whenever we want to use the **getkword** function which uses keywords for the reply to the program's command prompt, we need to use the **initget** function. We will use the function to determine what types of tolerances are placed in the series of notes on the drawing when the program is run.

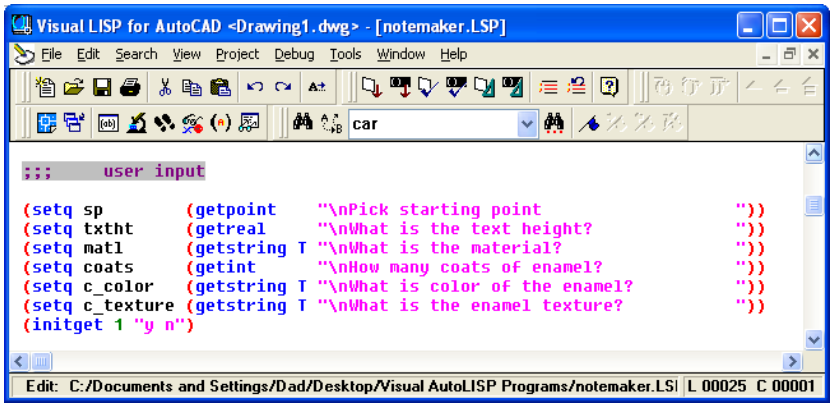

Figure 5.11 – Adding an Initget Expression

So type the following compound expression:

```
(initget 1 "y n")
```

The number behind the **initget** function determines the type of entry that will be accepted. Most of the time, we will use a **1**, meaning that the user cannot type a null entry. There are other bit codes used less often, shown in the table below.

Bit	Meaning
1	Prohibits a NULL input
2	Prohibits input of zero (0)
4	Prohibits negative values.
8	Allows the user to enter a point outside the drawing limits
16	Not used
32	Shows dashed lines when displaying rubber-band lines or boxes
64	Ignores Z coordinate
128	Allows unpredictable input

After the placing the input option control bit, the keywords are placed inside quotes. In our problem, we want either a **y** for yes or **n** for no, so the section of the code is written **"y n"**. We could have written **"yes y no n")** and each of these possible keyboard entry can be accepted.

Practice typing the following examples of the **initget** function at the command line of AutoCAD.

Function	Name	Description
initget	**Input Options for User Input Functions**	**Define the keywords for the getkword function**
Example		
Allow the user to type yes (y) or no (n) at the keyboard	**(initget 1 "y n")**	Returns: **nil**
Allow the user to type 1, 2 or 3 at the keyboard	**(initget 1 "1 2 3")**	Returns: **nil**

Using Getkword to Obtain Text from the Keyboard

When we want the user to type a text string that matches a certain list we present in the question, we use the **getkword** function. The user can type any entry at the keyboard, but the only one matching one of the keywords will be accepted. The **getkword** expression is set within the **(setq q1…….)** code.

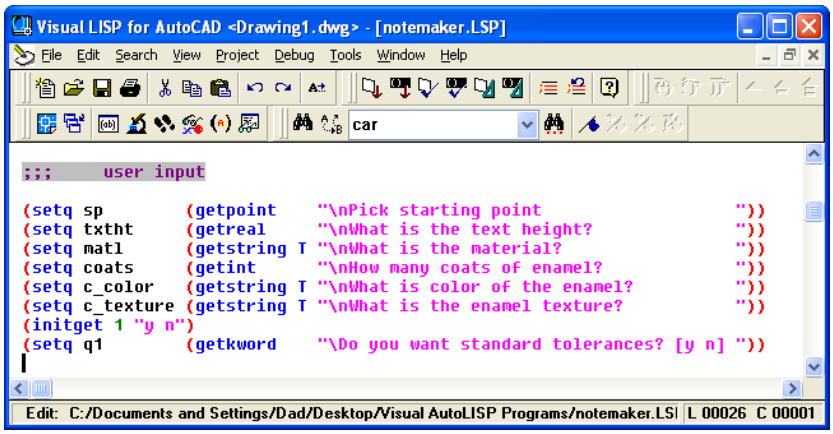

Figure 5.12 – Using the Getkword Function

So type the following compound expression:

(setq q1 (getkword "\nDo you want standard tolerances? [y n] "))

The information that the user types with the keyboard is stored in the variable name **q1**. Again the user input commands and questions are aligned in the LISP routine to allow the commands and questions to line up neatly on the command line in AutoCAD when the program is running. Aligning the expressions as shown in Figure 5.12 also gives us, the programmers the ability to easily check for syntax errors.

At the end of the question, place the possible answers to the question in brackets, so the user does not have to guess what the proper response is. In our case only **y** or **n** will work.

Practice typing the following examples of the **getkword** function at the command line of AutoCAD.

Function	Name	Description
getkword	**Get a Key Word**	**Allows the user to obtain a keyword text string by allowing the user to type at the keyboard**
Examples		
Responding the proper keyword	**(setq q1 (getkword "\nDo you want tolerances? [y n] "))**	Answer: **Do you want tolerances?** Then type: **y** **"y"**
Not responding the proper keyword	**(setq q1 (getkword "\nDo you want tolerances? [y n] "))**	Answer: **Do you want tolerances?** Then type: **no** **Invalid option keyword** and Repeats the initial question

Using the If Function to Make Decisions in a Program

Whenever we are confronted with a making a choice between two or more options in computer programming, the **if** function is a very popular solution to this challenge. The **if** function will execute the statements within the then section of the **if** expression when the logical test is true.

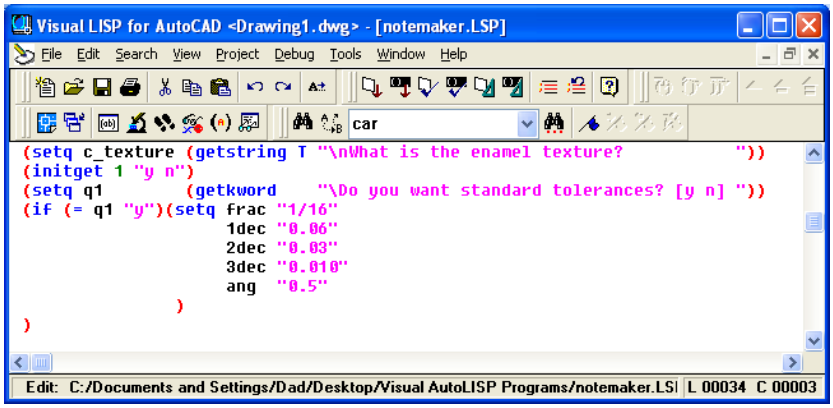

Figure 5.13 – Using the If Function with a Yes Answer

The **if** function also will execute the else section of the if expression when the logical test is false.

The **if** function is arranged to work in a more complex fashion than other AutoLISP tools. If is typed directly after the open parenthesis. Then an expression containing the logical test is written right after the **if**. The logical expression tests for a true or false response. When a program user answers the question "Do you want tolerances? [y n]", then the y or n is stored in the variable **q1.** So in the logical test we are asking does the text string "y" equal "y". Of course the answer is true and the **if** function will execute the expression in the next set of open and closed parentheses.

So type the following compound expression for the notemaker routine:

```
(if (= q1 "y") (setq frac "1/16"
                     1dec "0.06"
                     2dec "0.03"
                     3dec "0.010"
                     ang  "0.5"
               )
)
```

The setq function can be written as shown in Figure 5.13 where there is a series of variables with a definition following each holder of changeable data. Therefore if the logical test of **(= q1 "y")** is true, then **frac** will contain **"1/16", 1dec** will contain **"0.06", 2dec** will contain **"0.03", 3dec** will contain **"0.010"** and **ang** will contain **"0.5"**. This if expression does not contain an or else phrase for a logical test result of false.

Instead of using an or else expression in the first **if** statement, we are going to copy and paste the first if statement and paste the lines of code below the closed parenthesis of the first statement. After in the new lines of code as shown in Figure 5.14, then change the text strings to **getstring** with questions, so the program user can assign their own tolerances for each question.

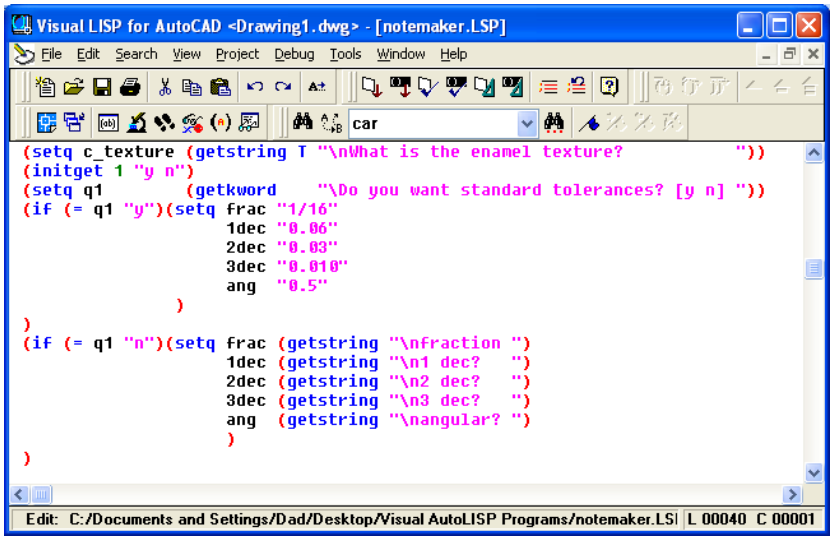

Figure 5.14 – Using the If Function with a No Answer

The setq function can be written as shown in Figure 5.14 where there is a series of variables with a question following each holder of changeable data. Therefore if the logical test of **(= q1 "n")** is true, then **frac** will contain a text string from the answer to the question "Fraction?" **1dec** will contain a text string from the answer to the question "1 dec?" **2dec** will contain a text

string from the answer to the question "2 dec?" **3dec** will contain a text string from the answer to the question "3 dec?" The variable **ang** will contain a text string from the answer to the question "Angular?" Again this if expression does not contain an or else phrase for a logical test result of false.

Make sure the second **if** expression looks like the one below:

```
(if (= q1 "n") (setq frac "\nFraction? "
                     1dec "\n1 dec?     "
                     2dec "\n2 dec?     "
                     3dec "\n3 dec?     "
                     ang  "\nAngular? "
               )
)
```

The optional "or else" expression in the **if** statement can be written avoiding a second **if** statement in the program. The technique we used in the notemaker code does not utilize the "or else" expression, keeping the code simple to use and reinforces the most typical method we will see the **if** statement written, the "then" expression. In beginning programming, the KISS rule "**K**eep **I**t **S**uper **S**imple" applies. Just because there are many options in every function does not override the fact that a program needs to have a clean look for checking. Experienced programmers have seen code writers use complex, messy looking and uncommented routines to the degree that no one else in the organization can read their work. The profession of programming has been through those years where the code writer was the holder of secret looking text, that this type of engineering is non-productive to the entire group.

Practice typing the following examples of the **if** and **=** functions at the command line of AutoCAD.

Function	Name	Description
if	**If Statement**	**The if function will execute the functions within the then section of the if expression when the logical test is true and within the else section of the if expression when the logical test is false**
Example		
If statement with just a then section with a logical test equally true	**(setq q1 "y")** **(if (= q1 "y") (alert "Hello"))**	Answer: **"Hello"**
If statement with just a then section with a logical test equally false	**(setq q1 "n")** **(if (= q1 "y") (alert "Hello"))**	Answer: **nil**
If statement with a then and or else section with a logical test equally false	**(setq q1 "n")** **(if (= q1 "y") (alert "Hello")** **(alert "Good-bye"))**	Answer: **"Good-bye"**

Function	Name	Description
=	Equal To	Logical test determining whether the first value is equal to the second value
Examples		
Using integers	(= 8 8)	Answer: **T (true)**
Using decimals	(= 4.5 4.5)	Answer: **T (true)**
Using text strings	(setq q1 "y") (= q1 "y")	Answer: **T (true)**
Using text strings	(setq q1 "n") (= q1 "y")	Answer: **F (false)**

Using Strcat to Concatenate Text Strings

The next section of the Construction code is making a point assignment. Instead of assigning a X,Y,Z coordinate, we will create a relative movement string to be placed in each command line. In order to accomplish this feat, we will learn two new functions in AutoLISP, **strcat** and **rtos**. Both functions handle or make text strings.

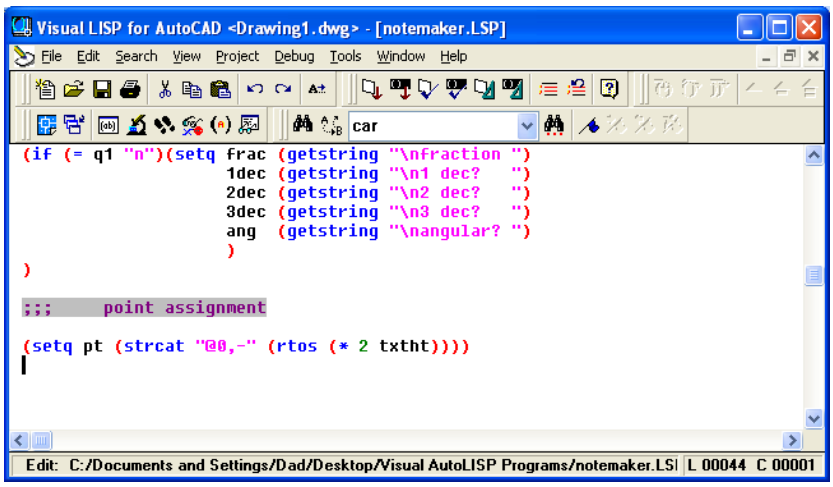

Figure 5.15 – Creating a Relative Displacement String

So type the following compound expression:

(setq pt (strcat "@0,-") (rtos (* 2 txtht))))

The variable **pt** will contain a text string made from two separate text strings. When the computer user answers the question for text height, that real number is placed in the variable **txtht**. The program works the inner most expression first and the asterisks * in **(* 2 txtht)** means we will multiply the **2** times the **txtht** (See Chapter 4). The function **rtos**, meaning convert the real number to a text string will change the real number representing the distance between two notes into a string. The **strcat** function will concatenate the text **"@0,-"** and the number in the form of a string. Concatenation or bringing together will create a text string like **"@0,-0.25"** if the computer user typed 0.125 for the height of the text in the notes. Since the @ symbol is in front of the X and Y coordinates, each new note will move 0.25 down in the negative Y-direction.

Now we are ready to write the code to insert the notes into the drawing.

Practice typing the following examples of the **rtos** and **strcat** functions at the command line of AutoCAD.

Function	Name	Description
rtos	**Real Number to a String**	**Will convert a real number to a text string**
Example		
Change a real number represented by the variable filename to a text string	**(setq filename 1000))** **(rtos filename)**	Answer: **"1000.0000"**
Change a integer 1000 to a text string "1000"	**(rtos 1000)**	Changes **1000.0000** to **"1000.0000"**

Function	Name	Description
strcat	**String Concatenation**	**Concatenates or brings together two or more text strings**
Examples		
Bringing three text string together, one which is the space	**(strcat "Good" " " "morning")**	Answer: **"Good morning"**
Bringing five text string together, two which is the space	**(strcat "How" " " "are" " " "you?")**	Answer: **"How are you?"**

Using Command "Text" to Insert Notes

As in the first eight programs, starting with the Boxcircle and Anglemaker, the **command** function is the key to drawing lines, circles, arcs, dimensions and placing text into a drawing. In the Notemaker program, we will execute a command that we may not have used in regular AutoCAD, **"text"**.

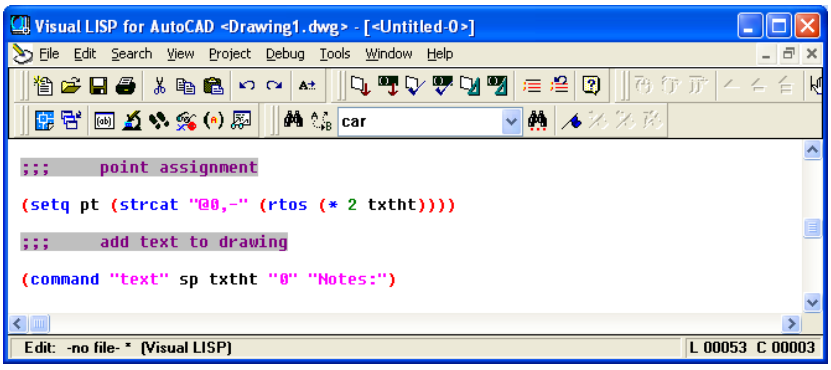

Figure 5.16 – Insert a Line of Text for "Notes:"

We have used the Mtext and Dtext tools on the Drawing toolbar in placing text on AutoCAD drawings in other World Class CAD textbooks. The Text command is the original single line text tool in the first years of computer aided designing using a Personal Computer (PC). The

tool comes in handy when programming since the command does not call up dialogue boxes and only contains simple attributes to the process of inserting notes.

So type the following expression:

(command "text" sp txtht "0" "Notes:")

The **command** function follows the open parenthesis, followed by the **"text"** tool. The next part of the expression is the text insertion point which we are assigning to the coordinates stored in the variable **sp**. Next, we see the text height which is being kept in **txtht**. All of our notes are horizontal so the rotation angle is **"0"**. Lastly, the text string is **"Notes:"**.

The next nine lines of code use the same basic **command "text"** expression with a few small exceptions. The insertion point changes from **sp** to **pt** for the relative movement. Standard notes like note 2 and the beginning of note 4 looks like the very first **command "text"** expression. The other seven statements contain the **strcat** function to concatenate strings and the **strcase** to change the text to lower case.

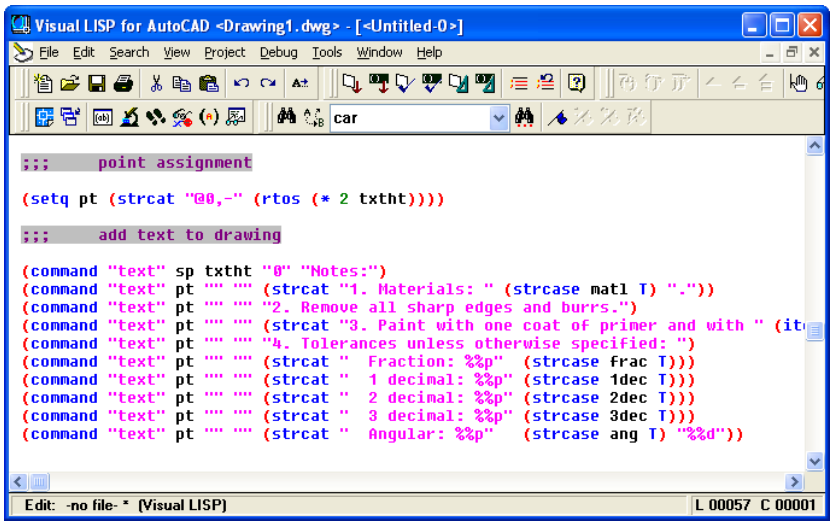

Figure 5.17 – Insert Notes with Command "Text"

So type the following expression:

```
(command "text" sp txtht "0" "Notes:")
(command "text" pt "" "" (strcat "1. Materials: " (strcase matl T) "."))
(command "text" pt "" "" "2. Remove all sharp edges and burrs.")
(command "text" pt "" "" (strcat "3. Paint with one coat of primer and with " (itoa coats)
" coat(s) of " (strcase c_texture T) " " (strcase c_color T) " enamel.")
(command "text" pt "" "" "4. Tolerances unless otherwise specified: ")
(command "text" pt "" "" (strcat "  Fraction: %%p"  (strcase frac T)))
(command "text" pt "" "" (strcat "  1 decimal: %%p" (strcase 1dec T)))
(command "text" pt "" "" (strcat "  2 decimal: %%p" (strcase 2dec T)))
(command "text" pt "" "" (strcat "  3 decimal: %%p" (strcase 3dec T)))
(command "text" pt "" "" (strcat "  Angular: %%p"   (strcase ang T) "%%d"))
```

In note number one, the **strcase** function can change the case of the text. When the expression is written **(strcase matl),** the note will insert with all the text in capital letters. When the expression is written **(strcase matl T),** the note will insert with all the text in lower case letters. After the material is changed to lower case, we concatenate the text string to **"1. Materials: "** using the **strcat** function. The note insertion point is relative to the starting point by the distance in the variable **pt**.

Most of the **command "text"** lines of code are the same except for these special considerations we see in our exercise. When we use the text **"%%p"** and **"%%d"** in the tolerance notes will place the plus and minus symbol (±) and the degree symbol (°) in the note. The **itoa** function will change an integer or whole number to a string. Remember to leave spaces after segments of text strings, so that when they are concatenated, the note has a space between each word. Sometimes programmers place a set of quotes with a single space **" "** in between to add a space in the sentence.

Practice typing the following examples of the **command "text"**, **strcase** and itoa functions at the command line of AutoCAD.

Function	Name	Description
Command "Text"	**Text Command**	**Will place text in an AutoCAD file based upon the insertion point, text height, rotation and text string.**
Examples		
Type in each LISP expression	**(setq sp (list 0 0 0))** **(setq txtht 0.125)** **(command "text" sp txtht "0" "Notes:")**	Places the text **Notes:** at the drawing origin

Function	Name	Description
strcase	**String Case**	**Changes the case of any text string**
Examples		
Change to upper case text	**(strcase "hello")**	Answer: **"HELLO"**
Change to lower case text	**(strcase "HELLO" T)**	Answer: **"hello"**

Function	Name	Description
itoa	**Integer to a String**	**Will convert a whole number (integer) to a text string**
Example		
Change a integer represented by the variable filename to a text string	**(setq filename 1000))** **(itoa filename)**	Answer: **"1000"**
Change a integer 1000 to a text string "1000"	**(itoa 1000)**	Changes **1000** to **"1000"**

Ending the Program

To end the program, we will set the object snap mode back to the original settings by using the **setvar** function followed by the variable **osm** which holds the original integer containing the Osnap settings. Type the following code.

(setvar "osmode" osm)

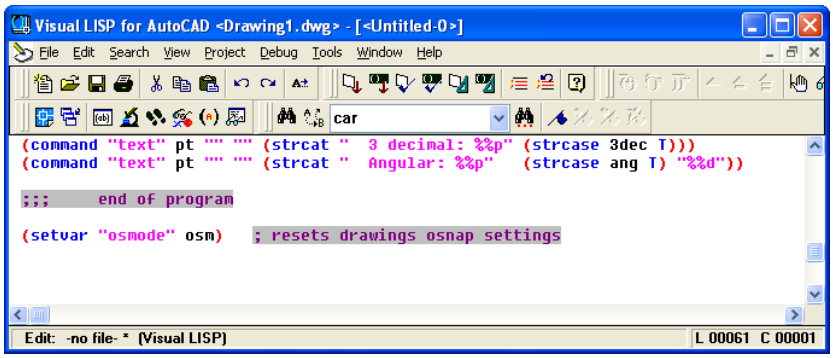

Figure 5.18 – End of Program

To end the program, we will need to place a parenthesis at the end of the code to close the **defun** **c:nm** function. Type the following code.

(princ)

)

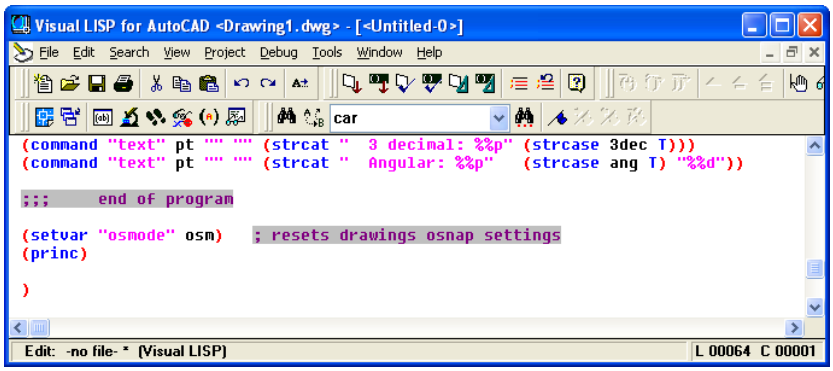

Figure 5.19 – Using the Princ Function

The **princ** function used in this routine will allow the program to end without printing the last line of the program to the command line. Without this function the command line can show a number or text that may not make sense to the use. This function is used to keep your code neat.

Practice typing the following examples of the **princ** function at the command line of AutoCAD.

Function	**Name**	**Description**
princ	**Princ Function**	**Will allow the program to run without printing the last line of the code to the command line**
Example		
Typing an expression at the command line without the princ function	**(setq a "Hello")**	Answer: **"Hello"**
Typing an expression at the command line without the princ function	**(setq a "Hello")(princ)**	Answer: **nothing**

Saving the Program

Now that the program is finished, we need to double check our typing with the text in this manual and then save our program to our folder named "Visual AutoLISP Programs".

Make sure the Look in list box is displaying the Visual LISP Programs folder and then select the program "notemaker" and press the Load button. At the bottom – left corner of the Load / Unload Applications window you will see a small text display that was blank initially but now displays the text as shown in Figure 5.20,

"notemaker.LSP successfully loaded"

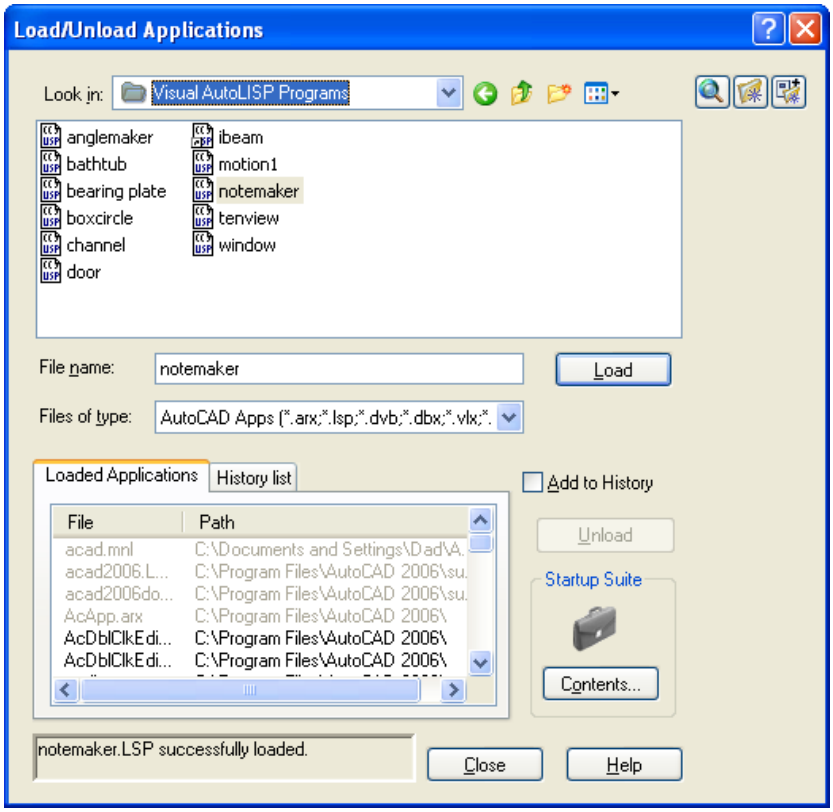

Figure 5.20 – Loading the Notemaker Program

After noting that the program is loaded, press the Close button and now when you are in the AutoCAD program, an AutoCAD message window appears in the middle of the graphics display. The copyright and information to start the program is shown.

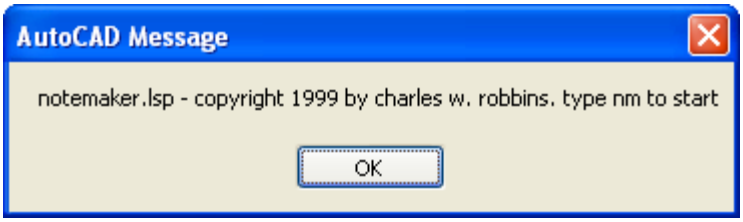

Figure 5.21 – The Alert Message

Running the Program

Press the OK button if you agree with the message and follow your own instructions by typing **nm** at the command line. The message "Pick starting point" appears on the command line and the we should select a point at the lower left hand corner of the AutoCAD graphics display.

The initial command line text is shown in Figure 5.22. Answer the questions to the prompts such as "0.125" for the text height, "Aluminum" for the material, "2" coats of enamel, "White" color, "Semi gloss" texture and standard tolerances. The notes will appear on the graphical display as shown in Figure 5.23.

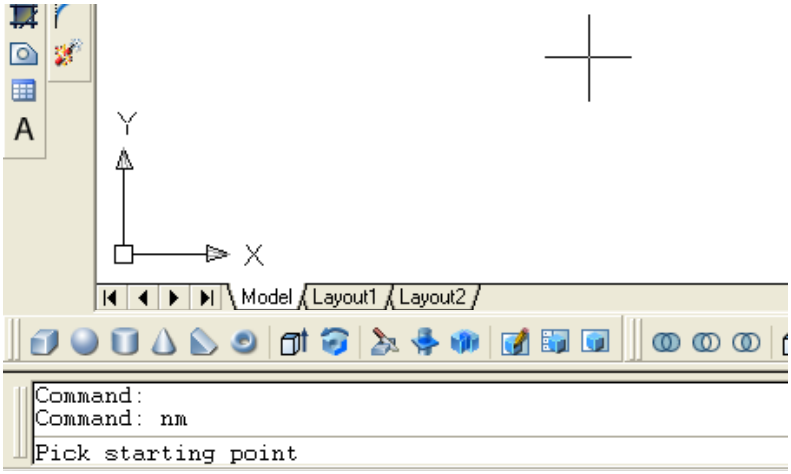

Figure 5.22 – Starting the Program

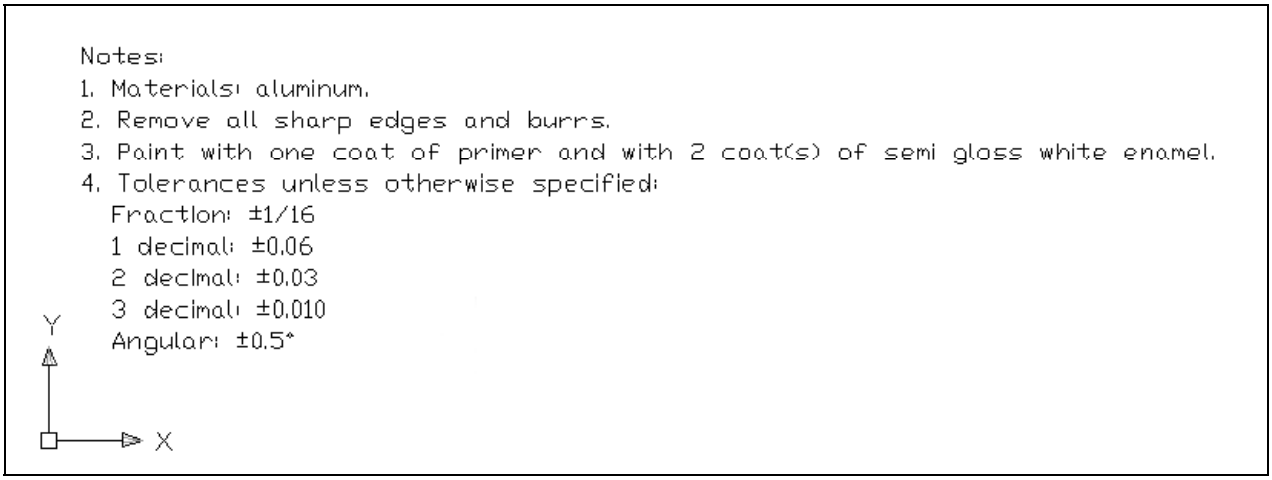

Figure 5.23 – The Notes in AutoCAD

Programs creating and placing text on a drawing are very easy to write once we have achieved writing the first program with these new functions. There are addition exercises for text based routines in the appendixes of this manual. Written below is the entire notemaker.LSP code for your benefit.

*** World Class CAD Challenge 04-14* - Open a New AutoCAD file and type the entire set of mechanical notes on proper layers, using proper dimensions and finally placing the points and x and y grid on the drawing. Save your drawing as notemaker.dwg. Open the Visual AutoLISP editor and code the notes problem using the Construction coding method. Save the code as notemaker.lsp.**

Send your copy of your code for verification to the authors of these problems to have your name and location posted. See the web site for instructions at

www.worldclasscad.com

```
;;; notemaker.lsp
;;; a program that writes notes to a mechanical part drawing
;;; copyright 1999 by charles w. robbins

(alert "notemaker.lsp - copyright 1999 by charles robbins. type nm to
start")

;;; start the program

(defun c:nm (/)

;;; drawing setup

(setq osm (getvar "osmode”))      ; gets drawings osnap settings
(setvar "osmode” 0)               ; sets osnaps to none

;;; user input

(setq sp        (getpoint    "\nPick starting point                  "))
(setq txtht     (getreal     "\nWhat is the text height?              "))
(setq matl      (getstring T "\nWhat is the material?                 "))
(setq coats     (getint      "\nHow many coats of enamel?             "))
(setq c_color   (getstring T "\nWhat is color of the enamel?          "))
(setq c_texture (getstring T "\nWhat is the enamel texture?           "))
(initget 1 "y n")
(setq q1        (getkword    “\nDo you want standard tolerances? [y n] “))
(if (= q1 "y")(setq frac "1/16"
           1dec "0.06"
           2dec "0.03"
           3dec "0.010"
           ang  "0.5%%d"
               )
)
(if (= q1 "n")(setq frac (getstring "\nfraction ")
           1dec (getstring "\n1 dec? “)
           2dec (getstring "\n2 dec? “)
           3dec (getstring "\n3 dec? “)
           ang  (getstring "\nangular? ")
           )
)

;;; point assignment

(setq pt (strcat "@0,-" (rtos (* 2 txtht))))

;;; add text to drawing

(command "text" sp txtht "0" "Notes:")
(command "text" pt "" "" (strcat "1. Materials: " (strcase matl T) "."))
(command "text" pt "" "" "2. Remove all sharp edges and burrs.")
(command "text" pt "" "" (strcat "3. Paint with one coat of primer and with
" (itoa coats) " coat(s) of " (strcase c_texture T) " " (strcase c_color T)
" enamel.”)
(command "text" pt "" "" "4. Tolerances unless otherwise specified: ")
```

```
(command "text" pt "" "" (strcat "  Fraction: %%p"  (strcase frac T)))
(command "text" pt "" "" (strcat "  1 decimal: %%p" (strcase 1dec T)))
(command "text" pt "" "" (strcat "  2 decimal: %%p" (strcase 2dec T)))
(command "text" pt "" "" (strcat "  3 decimal: %%p" (strcase 3dec T)))
(command "text" pt "" "" (strcat "  Angular: %%p"   (strcase ang T) "%%d"))

;;;    end of program

(setvar "osmode" osm)     ; resets drawings osnap settings
(princ)

)
```