Chapter

10

Creating a Motion Program for Animations

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

1. **The Advantage of Using While Loops and Animation Code**
2. **Starting the Motion Code by Launching the Visual LISP Editor**
3. **Learning to Use Selection Sets**
4. **Using a While Loop in a LISP Routine**
5. **Moving an Entity in a LISP Routine**
6. **Copying Code and Making Changes**
7. **Adding a While Loop and Nesting Loops**

The Advantage of Using While Loops and Animation Code

Many years ago I brought a class of students through the steps of creating while loops in their computer programs. In that exercise, I had the students create a basement stairs completely from scratch using Visual AutoLISP. The problem involved some mathematics, the knowledge of selections sets, and of course the while loops. I would have to say that most of the students really struggled through the exercise with me. My approach was too difficult. My challenge was to find a technique to train powerful programming functions and simultaneously allowing the programming student to concentrate on coding.

The next group going through the lesson plan for while loops, I still use a step with a run of 10 inches in a rise of the 8 inches. We repeat the single step ten times to construct a simple looking stairs. As you learned and chapter one, I never waste a student's time by writing a routine that they would put away and never use the code again. Initially this problem appears to be too simple to have any practical application. The next step in the lesson plan is to draw a seven inch diameter circle on the midpoint of the top step. My question to the class and the essence of the exercise is to bounce the ball down the stairs and to create the 3D animation to present at the conclusion of the lesson.

Since that day my students modify and enhance motion code to create fly around animations of buildings and product. After making three dimensional structures, they again use their motion scripts to make walkthrough animations without having to purchase a third party software package.

So in this exercise, we need to begin by drawing a single step as shown in Figure 10.1. Select the Line tool on the Draw toolbar and pick a point on the upper left corner of the graphical display, draw a line 10 inches to the right and a second line 8 inches down. Press Enter to close out the Line command. Now pick the Copy tool on the Modify toolbar, select the first two lines and press Enter to proceed to the second part of the Copy command. At the command prompt, specify based point or displacement or multiple, we are going to choose the multiple option. After typing "M" for multiple, pick the left Endpoint of the horizontal line for your base point. Now you copy the horizontal and vertical lines ten times by picking the lowest Endpoint of the vertical line. After adding nine steps, draw two additional lines to create the finished stairs as shown in Figure 10.2.

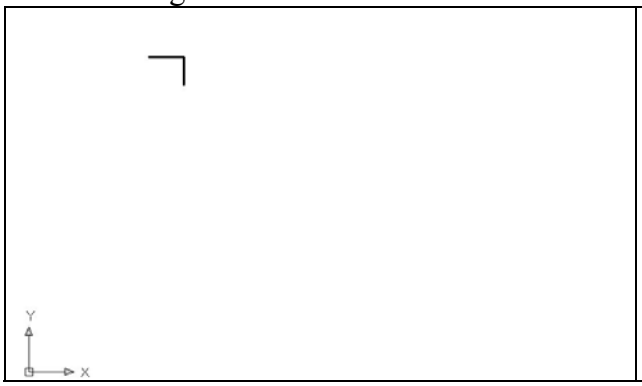

Figure 10.1 – The First Step

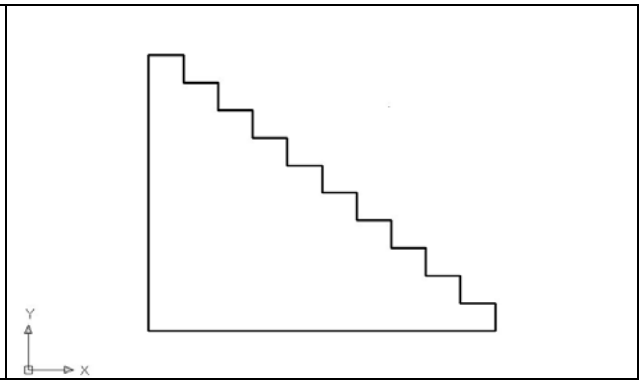

Figure 10.2 – The Finished Stairs

Select the Circle tool on the Draw toolbar and pick the Midpoint point on the upper horizontal line. Type **3.5** for the radius of the circle. Now pick the Move tool on the Modify toolbar, select the first circle and press Enter to proceed to the second part of the Move command. At the command prompt, specify based point or displacement or multiple, we are going to choose the south Quadrant of the circle and at the second point of displacement, pick the Midpoint of the top horizontal line. The ball will appear as shown in Figure 10.3.

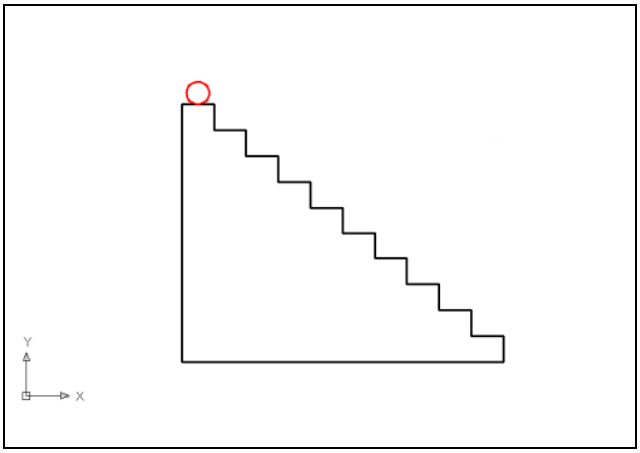

Figure 10.3 – Adding a Ball

Go ahead and save the file in your Visual AutoLISP folder, naming the file, Stairs. As in normal drawing practices, saving the file is critical to retain information and not to lose time drawing the same entity twice. By saving the Stairs drawing with the ball at the top step, we will find the process easy to reset when testing the code you are going to write in this chapter.

The strategy we will use in the programming code is to automatically select the ball, and move the entity to the right, stopping at each increment to take a picture. The question is "what is the distance we need to move to the right for each picture?" we can determine that by knowing a little bit about motion and about film. For a practical exercise, take a ball and roll the ball down a certain number of stairs timing the motion with a secondhand. The ball rolls and

When you watch a Saturday morning cartoons, the animator can show their film at twelve frames per second for very rough looking presentations. On the other side of the animation quality scale we have companies producing presentations at 28 frames per second for very fine displays. After timing a ball rolling down a stairs, we found that the ball took 8 seconds to descend. We choose 12 frames per second for our first animation so we are planning for around 96 frames. With 96 frames divided by 10 steps, we figure 9.6 frames per step. By simple math we decide to move the ball 5 times at 2 inches each displacement to clear the 10 inch step, then 4 times at 2 inches down to reach the next step. This will give us an equal displacement of 2 inches for each movement and a total of 90 frames in the movie. This is very close to the 96 frames we were planning on using from our actual experiment. When we finish the animation, we will see a presentation that closely represents what would be believable. That is important when bouncing a ball down a stairs or walking through a building in an animation.

In this program, we will use three while loops, a horizontal movement while loop, a vertical movement while loop, and finally a while loop that will repeat the process for ten steps. As you can see in Figures 6.4, the first while loop will the move the ball in a horizontal motion five times in 2-inch increments to the right, taking a picture in each time. The next vertical while loop, we will capture the vertical movement of the ball moving downward four times in two inch increments as shown in Figure 10.5.

The third loop will take the ball down all ten stairs. We will accomplish this assignment by nesting the first two loops inside the third loop. We need to proceed to see how this is done.

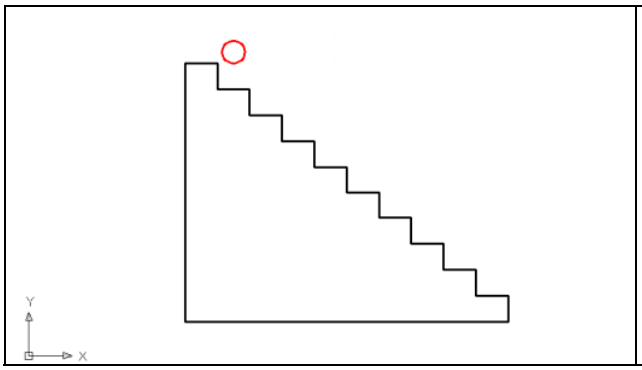

Figure 10.4 – Moving the Ball to the Right

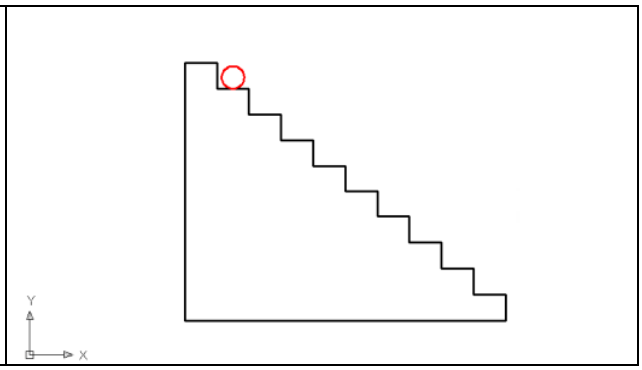

Figure 10.5 – Moving the Ball Down

Starting the Motion Code by Launching the Visual LISP Editor

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

```
;;;     motion1.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.

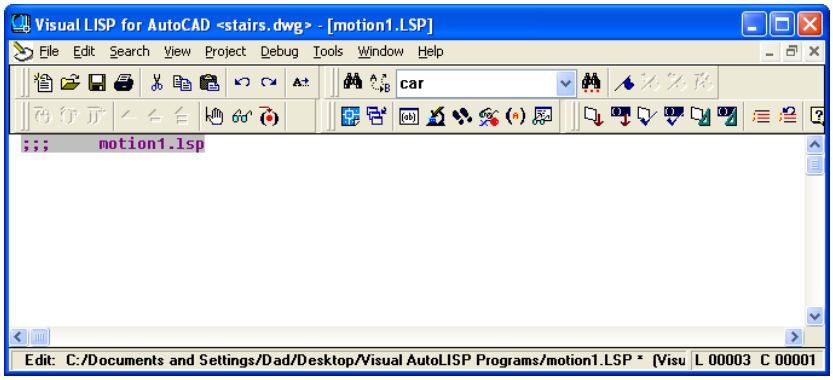

Figure 10.6 – Starting the Motion Program

The next line or lines 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. Follow the information provided and type the comments where needed. After a few programs, we will always remember to add comments.

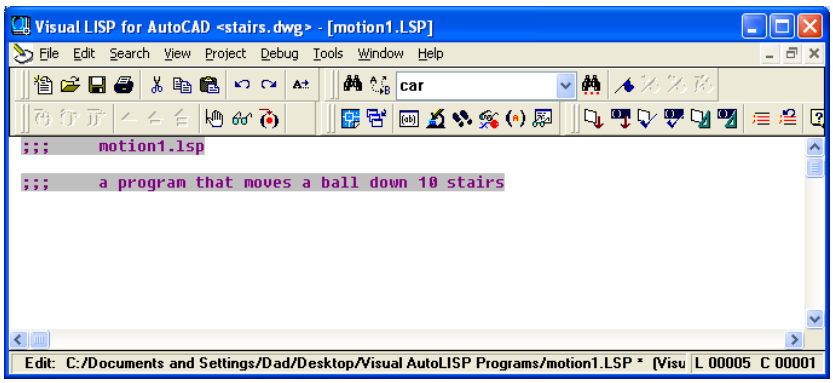

Figure 10.6 – Adding Comments

The copyright statement is the next line of code with the actual word copyright, the year initially written and the full and legal name of the author. If you make a change the next year, the copyright statement would say

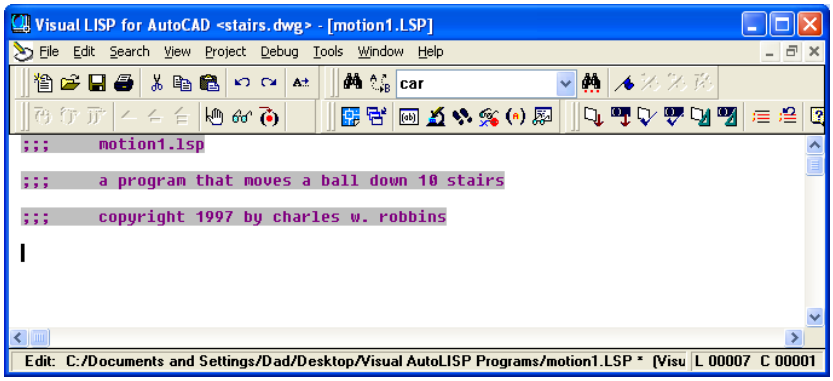

Figure 10.7 – Adding the Copyright Statement

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.

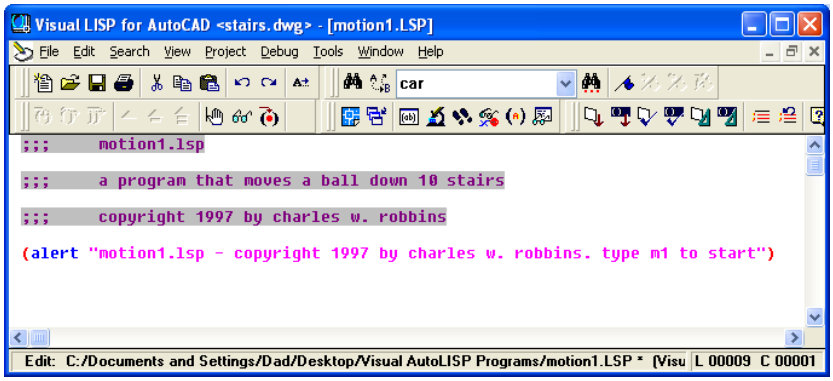

Figure 10.8 – Adding the Alert Expression

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.

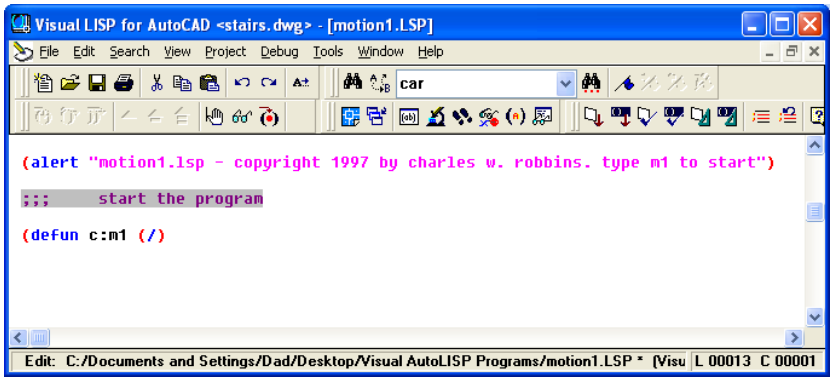

Figure 10.9 – The Defun Expression

Next type **m1** which will be the execution symbol to start the program. Keep in mind the alert message that stated "type m1 to start". The alert message text and the **defun** symbol must match. The open and closed parenthesis "()"following the **m1** 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.

Learning to Use Selection Sets

To automatically obtain the ball without picking the circle with the mouse, use the **ssget "x"** function. This is a very powerful tool in the list of Engineering programming functions. In many programs, the writer will just obtain information from a user's input at the keyboard or from a specific field in a database.

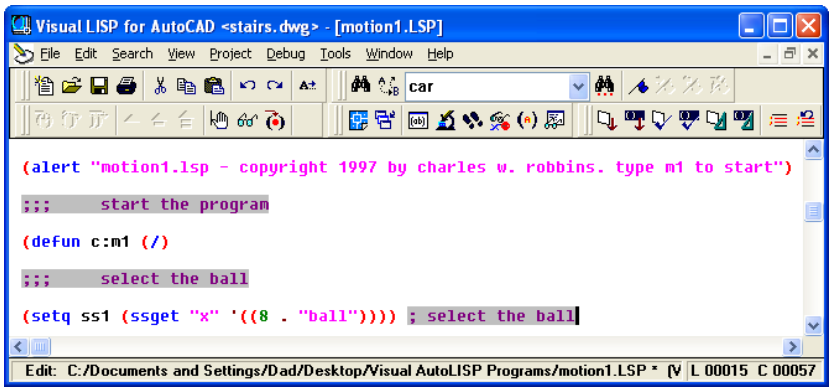

Figure 10.10 – The Filtered Selection Set Expression

The **ssget** function will extract information from any AutoCAD drawing no matter where that entity resides. If we use the **ssget** function and select all the entities with the expression shown below.

(setq ss1 (ssget "A"))

Then every entity is places in a list called **ss1** and we can pull that list apart to obtain data. This is not as efficient as using a filter where initially our list will only contain a smaller amount of pertinent information. Therefore we learn to use the **ssget "x"** function where we can filter all the records except what is listed at the end of the expression. In our movie program we write

(setq ss1 (ssget "x" '((8 . "ball"))))

After the **ssget "x"**, we write **'((8 . "ball"))** which describes the filter we desire. The number **8** is a group code which means layer name. All AutoCAD entities have data broken down into logical data fields such as entity name, layer name and insertion point. The text **"ball"** describes the name of the layer we are searching for entities. In the movie, the only entity residing on the ball layer is the circle that we are moving, so the **ssget** function will select the ball no matter where the object is located.

Before we continue to another expression, we need to take the time to understand the selection set function and how AutoCAD controls the drawing database. We can do this by drawing a single line on the graphical display. So go ahead and draw a single line as shown in Figure 10.11.

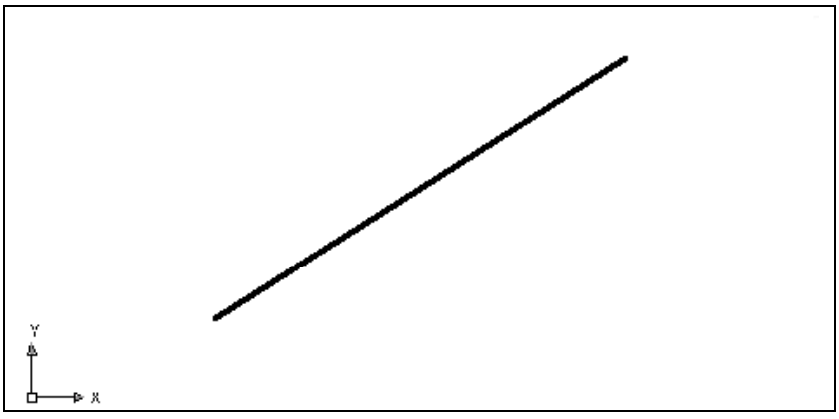

Figure 10.11 – A Single AutoCAD Line Entity

At the command line type the expression listed below. Notice that the group code is now **0**, which stands for entity type and the category we want is **"line”.**

(setq ss1 (ssget "x" '((0 . "line”))))

The AutoCAD program will return with comment on the command line with **<Selection set: 4>**. Anytime we capture a data in a selection set we will notice the number changing. To determine the quantity of entities in the selection set type the following code at the command line.

(setq quantity (sslength ss1))

We are using **quantity** as the variable name. The AutoLISP function **sslength** returns the number of entities in the selection set **ss1**. The command line will come back with **1**. The **sslength** function is important when we use a while loop to sort through a selection set automatically. Next type the following expression on the command line.

(setq entityname (ssname ss1 0))

We are using **entityname** as the variable name. The AutoLISP function **ssname** gives the name of an entity of the first item in the selection set **ss1**. The command line will come back with **<Entity name: 7ef60e98>**. An AutoCAD name has eight digits that can have 0 through 9 or A to Z in each digit. In a selection set, AutoCAD starts counting with the number **0**, so we have to remember to always start our counts in a program at zero and not one when using the **ssname** function. Now type the following expression on the command line.

(setq entitylist (entget entityname))

We are using **entitylist** as the variable name. The AutoLISP function **entget** will return the entire database list that describes the line entity on the graphical display and the first item in the selection set **ss1**. The command line will come back with following data. Your data will be different since your line is in a different location.

((-1 . <Entity name: 7ef60e98>) (0 . "LINE") (330 . <Entity name: 7ef60cf8>)
(5 . "8B") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (370 . 30)
(100 . "AcDbLine") (10 10.4573 9.24445 0.0) (11 27.8159 20.2655 0.0) (210 0.0 0.0 1.0))

For many AutoCAD expert operators this is a time of great discovery. The AutoCAD drawing file has header information which contains information regarding how to set up the environment for the actual drawing and then comes a complete collection of records which contain the data for every drawing entity which you see in the drawing. The AutoCAD file is actually three dimensional database. In later projects we will turn the monitor off and without any mouse or keyboard entries we will still draw in AutoCAD using a LISP program. For architectural and engineering experts, they finally see some computer and design power, and not drawing or designing everything manually.

Every data field inside of each set of parenthesis begins with a group code. There is a table of group in Appendix Z in this textbook. Common group codes that we use to extract data all the time are in the table below.

0	Entity Type
1	Text
2	Block Name
8	Layer Name
10	Start or Insertion Point
11	End Point

We will use selection sets in most of our programs, so this is good time spent reviewing what a single data record looks like in AutoCAD. Later we will learn how to extract starting and ending points of lines, and change text or blocks in a drawing automatically. All of these capabilities are made possible with a great understanding of selection sets, the functions that allow us to read the data, and finally the functions which will enable us to change the entity automatically without using a keyboard and mouse.

In the motion program, the selection set contains just that single entity, the circle. Anytime we wish to move the circle, we just call out the selection set **ss1.** Notice in the example shown below, we could have typed **(ssget "x" '(0 . "circle"))))** in the program and achieved the same result. In our program, we know there is only a single circle on the ball layer. We normally do not use entity type selections in motion programs since a designer may want to use circles in their drawing and the program may automatically select the wrong circle. The dedicated layer is an obvious choice for motion code.

Function	Name	Description
ssget "x"	**Obtain a Selection Set Using a Filter**	**Will automatically select entities using a filter such as layer name, entity type like circle.**
Examples		
Using layers	**(ssget "x" '(8 . "ball"))))**	Selection Set: 7
Using entity	**(ssget "x" '(0 . "circle"))))**	Selection Set: 9

The next expression in the motion program will set the variable for each file name where each frame is written.

(setq filename 1000)

We normally start with 1000 since the all of the frames will align in perfect numerical order when automatically placing them into an animation program.

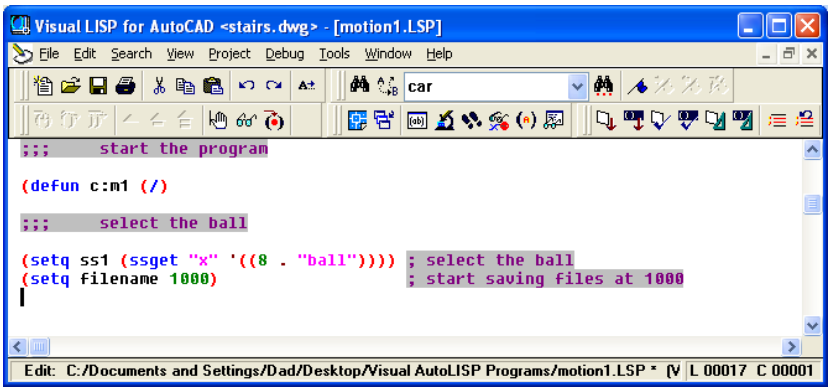

Figure 10.12 – Declaring the Filename to be 1000

When you write a file representing each frame starting with 1 and on up then the files will not be in order when you place them in an animation program. You will have files listed as 1, 10, 11, 12 ….. 19, 2, 20, 21. You can avoid this problem by using 1000 when you want up to 999 frames in the movie.

Using a While Loop in a LISP Routine

The comment for starting the horizontal movement is added and before we start any while loop, we need to set a condition to control the loop and in this case we will set a counter before entering the while loop. Using the setq function, we type the expression below to start the loop at zero.

(setq counter1 0)

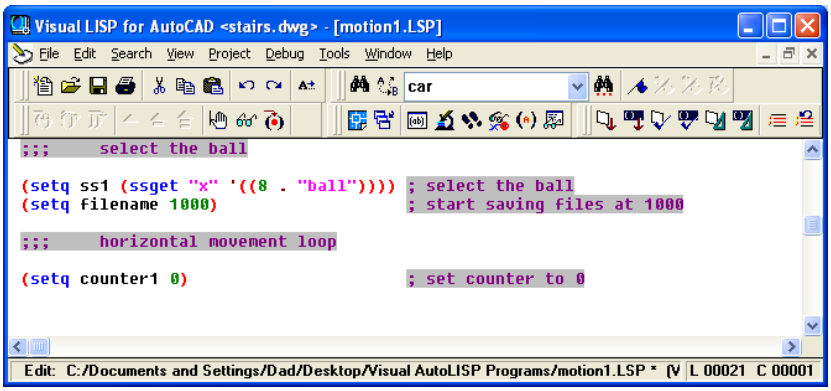

Figure 10.13 – Starting the Horizontal Loop

The next expression is the code is the entrance to the while loop.

(while (< counter1 5)

A condition statement trails the **while** function. Since the step is 10 inches in depth and the circle needs to travel that distance to clear the step.

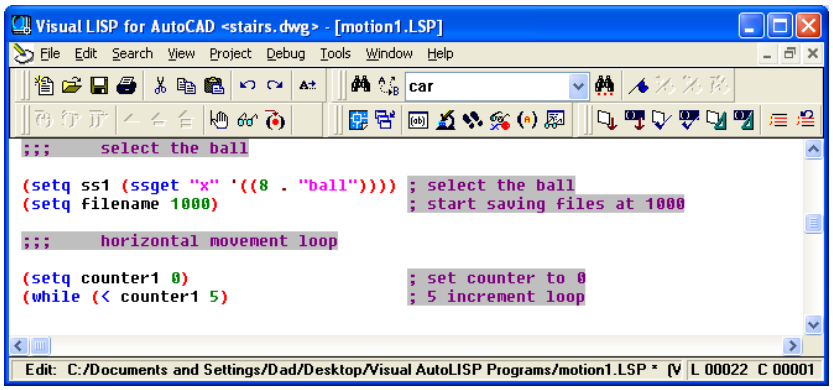

Figure 10.14 – Placing the While Loop Expression

We are moving the circle 2 inches in every displacement and with 5 loops in the **while** function, which means 2 times 5 equals 10. In this instance the condition function is the less than symbol, **<.** A while loop will run anytime the condition set returns a true response in the code. The first time into the loop the condition is (< 0 5) which is true so all of the expression inside the while loop will be read in the program. The second time into the loop the condition is (< 1 5) which is also true so all the loop continues to run. The third time into the loop the condition is (< 2 5) which is also true and the loop continues. The fourth time into the loop the condition is (< 3 5) which is also true and this seems to be going on and on. In a classroom we go through every step on our first while loop. By this time many students do not think this will ever end. The fifth time into the loop the condition is (< 4 5) which is also true, because 4 is less than 5. Now on sixth time into the loop the condition is (< 5 5) which is false and so the while loop will not execute and the next expression in the code will be read.

Notice that the expression below does not contain an ending parenthesis for the **while** function.

(while (< counter1 5)

The ending parenthesis comes at the end of the list of expression inside of the while loop.

Function	Name	Description
While	**While Loop**	**Will automatically select entities using a filter such as layer name, entity type like circle.**
Examples		
Using a counter	**(while (< counter 5)** **)**	Will continue 5 times
Using a question	**(while (= ball "y")** **)**	Will continue as long as ball equals "yes"

Moving an Entity in a LISP Routine

Now we want to move the circle 2 inches to the right. We use the **command** function to start the process. Now since the AutoCAD command is **"move"**, that comes next in the expression. The **ss1** variable contains the circle so we will list that next for the select objects part of the Move command.

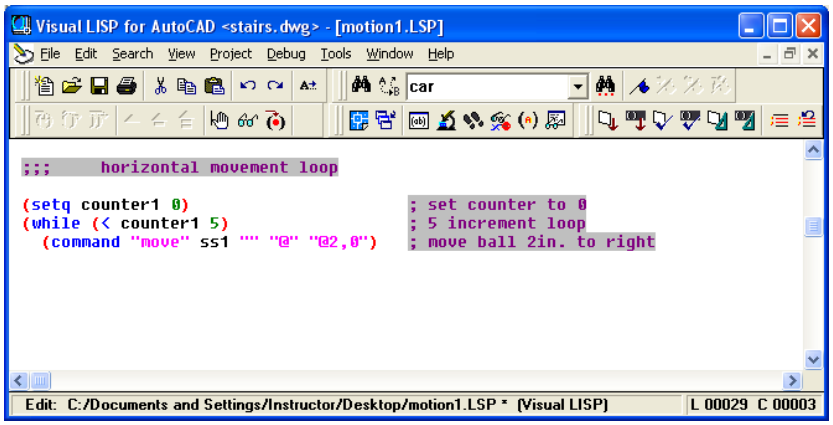

Figure 10.15 – Move the Ball Horizontally to the Right

Now that we have the circle in the Move command's selection set, the double quotes **""** means **Enter** so we are now in the next part of the Move command concerning displacement. The **"@"** symbol will pick a point on the graphical display. The **"@2,0"** will move the circle 2 inches to the right. The entire expression is shown below.

(command "move" ss1 "" "@" "@2,0")

Obviously the Copy command would run the same way as the Move command.

Function	Name	Description
Command "Move"	Move Command	Will move an entity in an AutoCAD file based upon the displacement criteria

Examples		
Moving a single selection set or a single entity using relative coordinates	(command "move" ss1 "" "@" "@2,0")	Moves **ss1**, 2 inches to the right
Moving multiple selection sets or two entities using relative coordinates	(command "move" ss1 ss2 "" "@" "@2,0")	Moves **ss1** and **ss2**, 2 inches to the right
Moving a single selection set or a single entity using a starting point **pt1** as the base point and **pt2** as the second point of displacement	(command "move" ss1 "" pt1 pt2)	Moves **ss1** The same distance as from **pt1** to **pt2**

The next expression in the code will save the bitmap image of the graphical display to file. The **saveimg** function needs to have the **c:** to the beginning of the command since this is an ARX command. Notice that when we type Saveimg or Rotate3d at the command line, the AutoCAD program loads the ARX function.

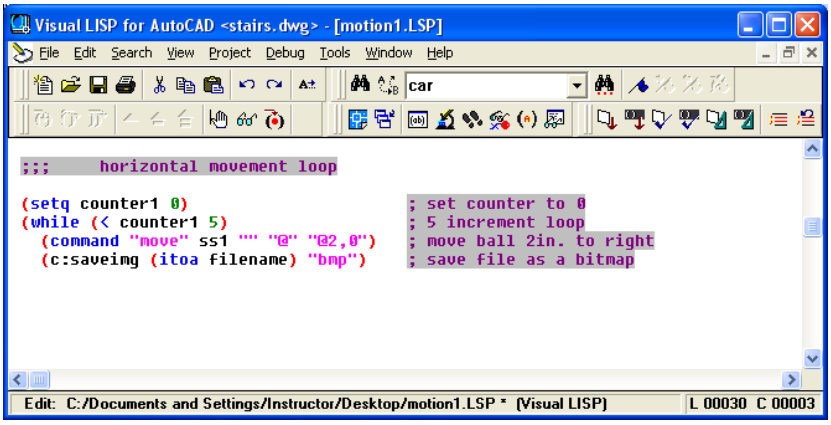

Figure 10.16 – Save the Image of the Graphical Display

After the **c:saveimg**, we need to call out the file name. The variable **filename** contains an integer, so we will convert the integer to a text string using the **itoa** function.

(c:saveimg (itoa filename) "bmp")

The **itoa** function is just one function that will change an integer to a text string. There are functions that convert between all of the different types of variables in AutoLISP. A variable can be a real number like 1.0, an integer like 1, a list like (1) or a text string like "1". There are AutoLISP functions that will allow the code writer to change between any of the four types of variables.

The last part of the expression tells the **saveimg** function what type of graphic file you wish to have. There are three choices for our graphical image: Bitmap (bmp), TGA (tga) and TIFF (tiff). We choose the Bitmap image to easily use in our animation program.

Now when we write the frames to files they will be in the same folder with the drawing file that we have our drawing file of the stairs.

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	(itoa filename)	Moves ss1, 2 inches to the right
Change a integer 1000 to a text string "1000"	(itoa 1000)	Changes 1000 to "1000"

Function	Name	Description
c:saveimg	Save Image	Will create a graphical image file of the graphical display
Example		
Makes a Bitmap image of the graphical display	(c:saveimg (itoa filename) "bmp")	Returns SAVEIMG Save Image done!
Makes a Bitmap image of the graphical display	(c:saveimg (itoa filename) "tga")	Returns SAVEIMG Save Image done!
Makes a Bitmap image of the graphical display	(c:saveimg (itoa filename) "tiff")	Returns SAVEIMG Save Image done!

The next expression we will place in the code will add one to the variable **filename**. We will use the 1+ function to add 1 to the 1000, so the variable **filename** will now be 1001.

(setq filename (1+ filename))

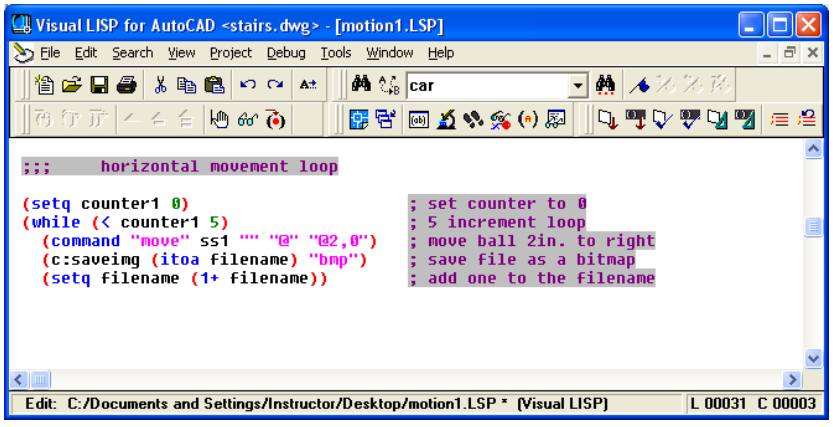

Figure 10.17 – Add One to the Filename

In chapter four, we discover that we can use the 1+ function to add one to a counter or we can utilize the 1- to remove one from a counter.

The next expression we will place in the code will add one to the variable **counter1**. We will use the 1+ function to add 1 to the 0, so the variable **counter1** will now be 1.

```
(setq counter1 (1+ counter1))
```

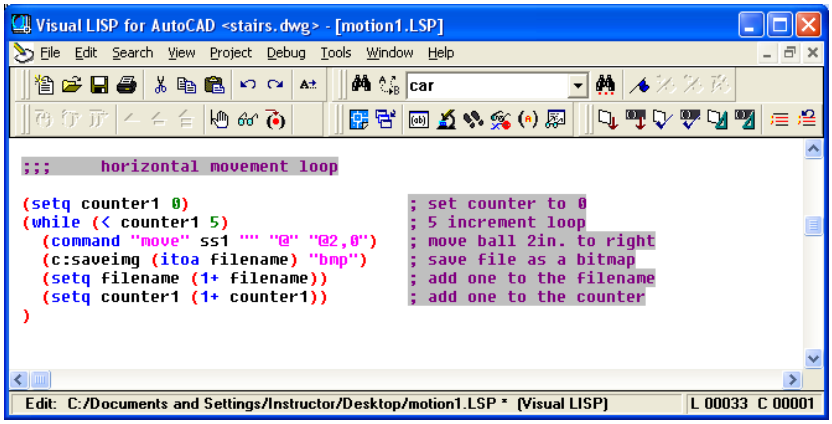

Figure 10.18 – Add One to the Loop Counter

Now we can add the final parenthesis to the end of the while loop under the last expression. The entire while loop is shown below.

```
;;;  horizontal movement loop

(setq counter1 0)                          ; set counter to 0
(while (< counter1 5)                      ; 5 increment loop
  (command "move" ss1 "" "@" "@2,0")       ; move ball 2in. to right
  (c:saveimg (itoa filename) "bmp")        ; save file as a bitmap
  (setq filename (1+ filename))            ; add one to the filename
  (setq counter1 (1+ counter1))            ; add one to the counter
)
```

Copying Code and Making Changes

Now you already know that we can make syntax errors in AutoLISP, so if we check the horizontal movement while loop and know that the expressions are correct, we can highlight the nine lines, which include comments, then right click in the highlighted area to read the menu choices. Select Copy from the menu as shown in Figure 10.19.

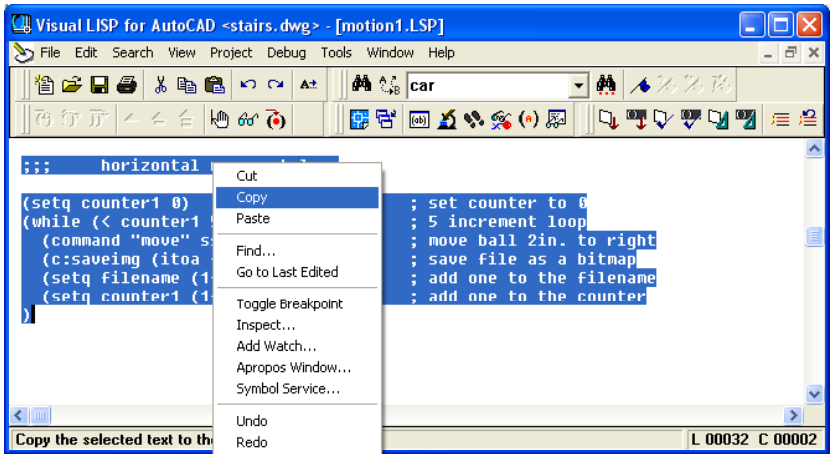

Figure 10.19 – Copy the While Loop to Create Another

Now place the cursor two spaces below the last expression and right click with the mouse. Select Paste and a copy of the first while loop will now be placed in the routine.

In the comment, change the word "horizontal" to "vertical" as in Figure 10.20. In this while loop the circle needs to go down an eight inch step. We are moving the circle 2 inches down in every displacement and with 4 loops in this **while** function, means 2 times 4 equals 8. Other than the condition statement in the while loop and the direction of the circle moving, both of the while loops are identical.

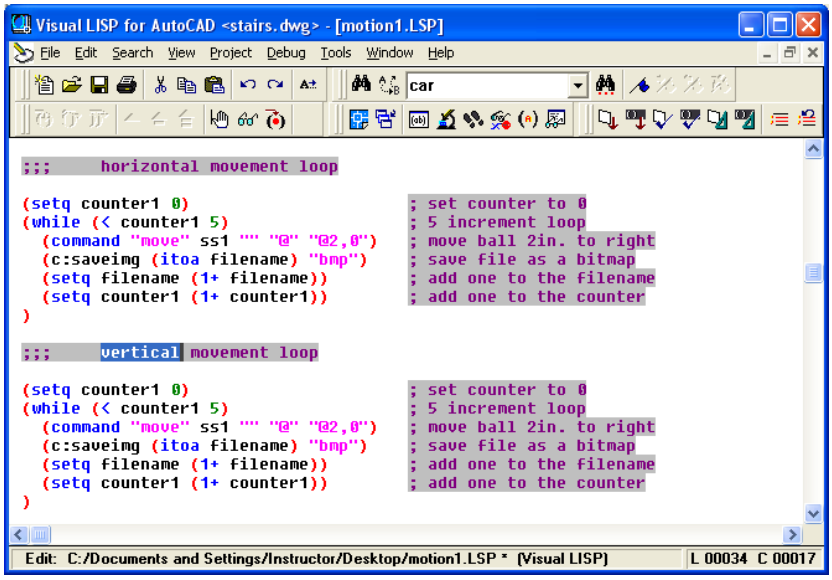

Figure 10.20 – Changing a Comment in the Vertical Loop

In the vertical movement loop, change the condition statement from a "**5**" to a "**4**" as shown below.

(while (< counter1 4)

Change the comment field to 4 increment loop.

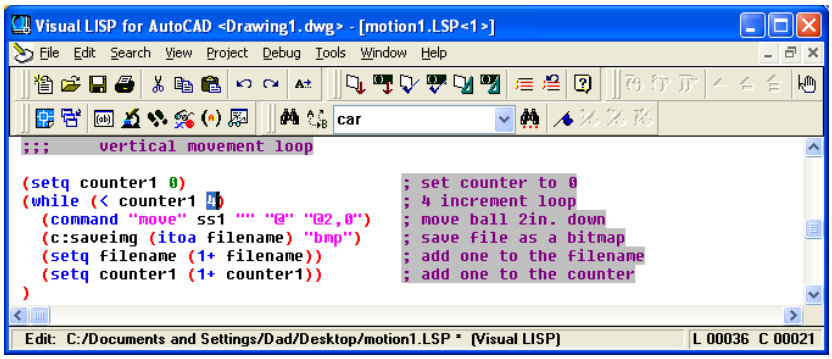

Figure 10.21 – Changing the Condition Statement to 4

Now we want to move the circle 2 inches down. To review, the **ss1** is the selection set containing the circle, the double quotes **""** means **Enter** so we are now in the next part of the Move command. The **"@"** symbol will pick a point on the graphical display. The **"@0,-2"** will move the circle 2 inches downward. Change **"@2,0"** to **"@0,-2"**. Do not forget to change the comment field.

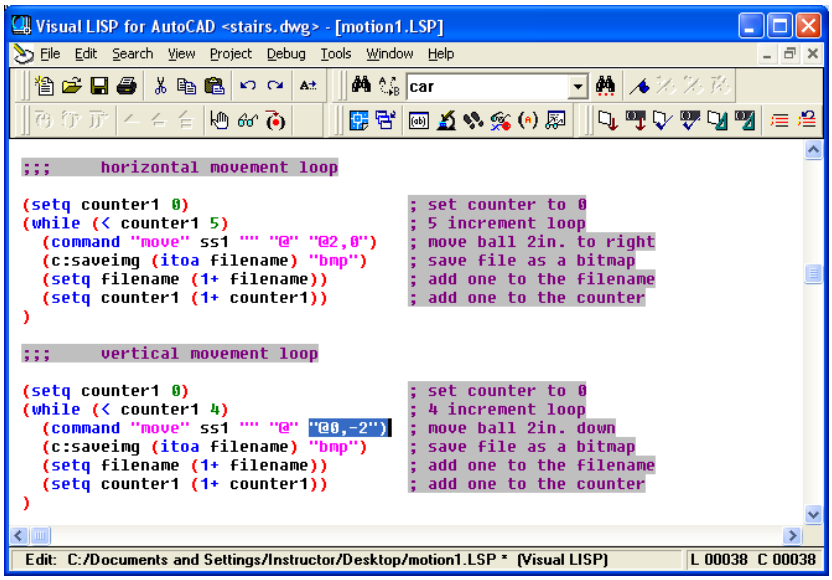

Figure 10.22 – Move the Ball Down Two Inches

To end the program, we will need to place a parenthesis at the end of the code to close the defun **c:m1** function. Type the following code.

;;; end the program

)

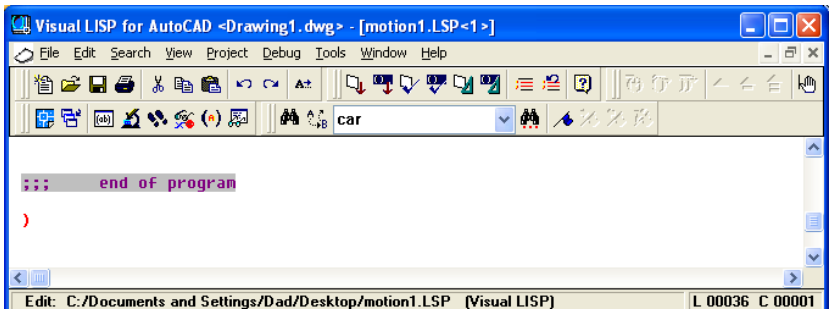

Figure 10.23 – Move the Ball Horizontally to the Right

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 "motion1" 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 10.24,

"motion1.LSP successfully loaded"

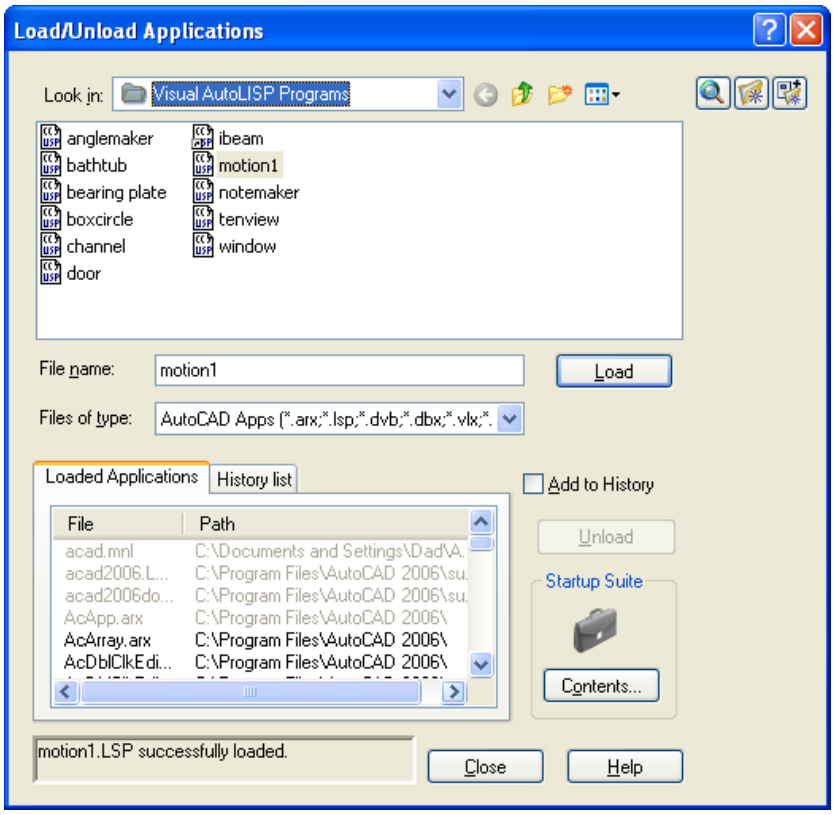

Figure 10.24 – Move the Ball Horizontally to the Right

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 stating: "motion1.lsp – copyright 1997 by charles robbins. Type m1 to start"

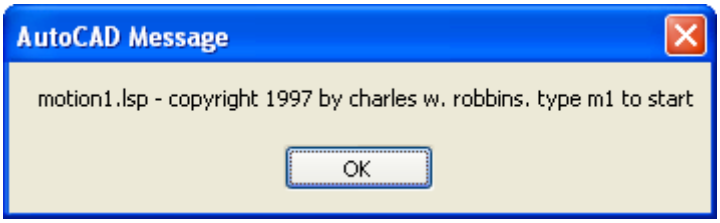

Figure 10.25 – Move the Ball Horizontally to the Right

Press the OK button if you agree with the message and follow your own instructions by typing **m1** at the command line. The circle will move 10 inches to the right and 8 inches down to the second step and stop. Our graphical display should appear as shown in Figure 10.26. If the ball moved as shown, continue with writing the third while loop. If not we must troubleshoot the code to find the error against the entire code shown at the end of the chapter.

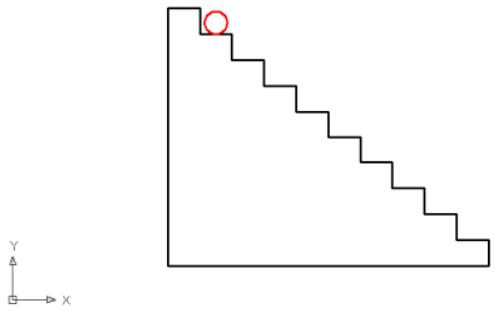

Figure 10.26 – Move the Ball Down One Step

Adding a While Loop and Nesting Loops

The comment for starting the stair while loop is added right below the **(setq filename 1000)** and before we start any while loop, we need to set a condition to control the loop and in this case we will set a counter before entering the while loop. Using the setq function, we type the expression below to start the loop at zero.

(setq staircounter 0)

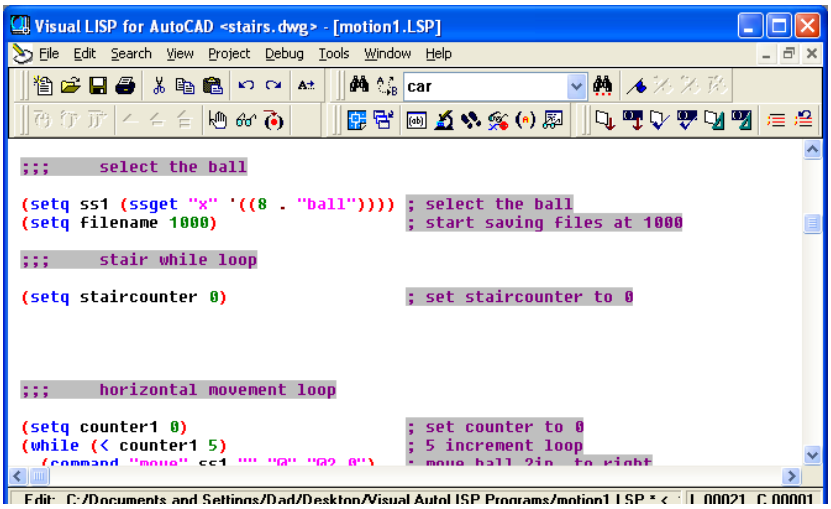

Figure 10.27 – Add Another Counter Expression

The next expression is the code is the entrance to the while loop.

(while (< staircounter 10)

A condition statement trails the **while** function. Since there are ten steps in the stairs, this while loop will contain a condition statement that allows for ten loops in the while function.

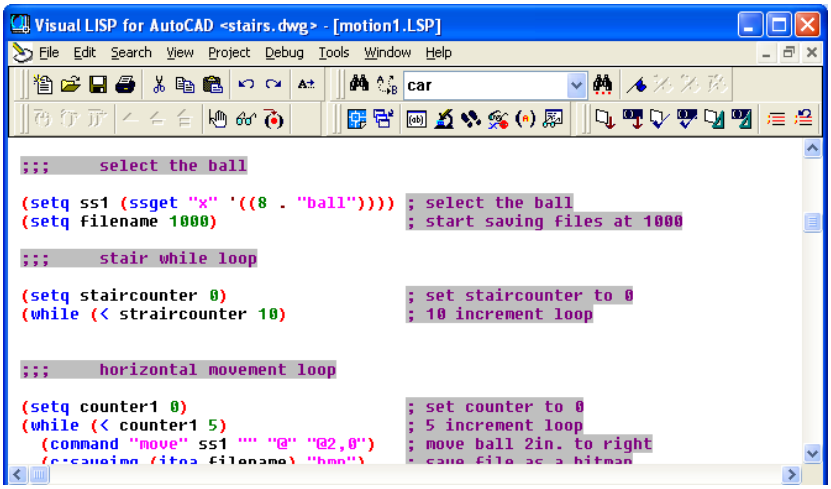

Figure 10.28 – Add the While Loop for 10 Stairs

The next expression we will place in the code will add one to the variable **staircounter**. The expression will be written after the second while loop as shown in Figure 10.29. We will use the 1+ function to add 1 to the 0, so the variable **staircounter** will now be 1.

(setq staircounter (1+ staircounter))

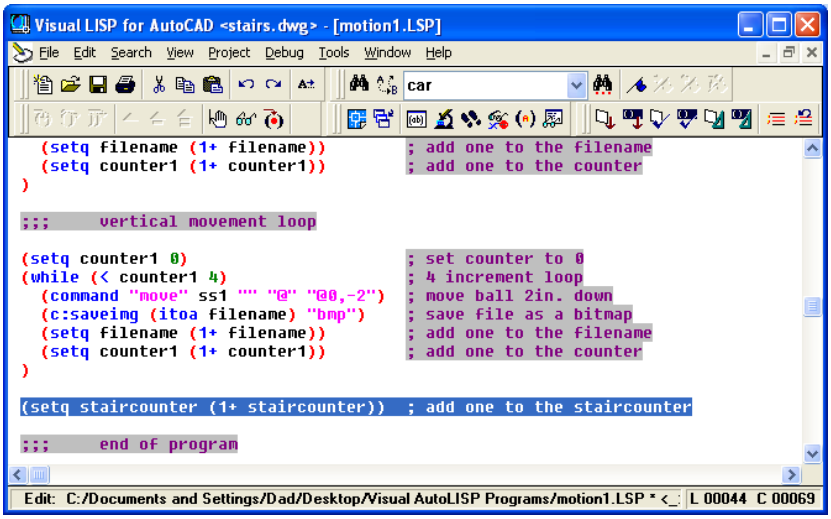

Figure 10.29 – Add One to the Staircounter

Now we can add the final parenthesis to the end of the third while loop under the last expression. The entire while loop is shown below. Now the first two while loops are nested inside the third while loop. In a single incremental loop in the third while loop, the circle will go ten inches to the right and then eight inches down.

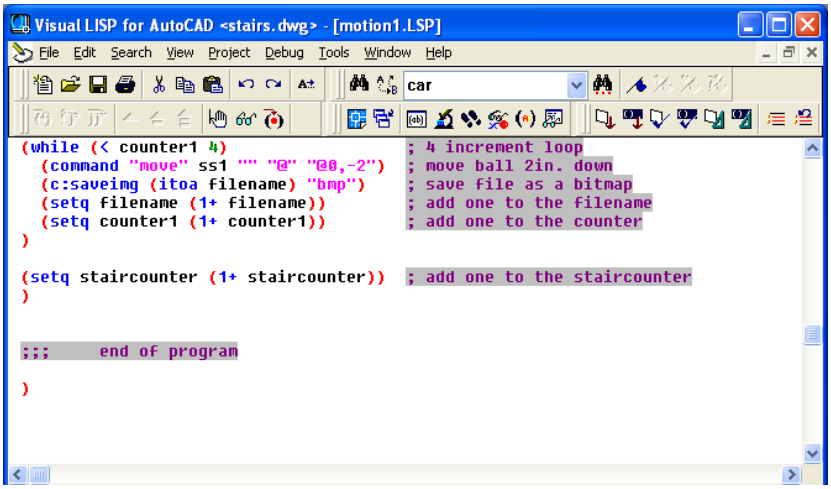

Figure 10.30 – Add a Closing Parenthesis to the Third Loop

Now, the first time around the loop the condition is (< 1 10) which is true so all of the expression inside the while loop will be read in the program. This will continue into the tenth time into the loop the condition is (< 9 10) which is also true, because 9 is less than 10. Now on eleventh time into the loop the condition is (< 10 10) which is false and so the while loop will not execute and the next expression in the code will be read, which is the end of the program. Now save the program, load the **motion1.LSP** application and run the code by typing **m1** at the command line.

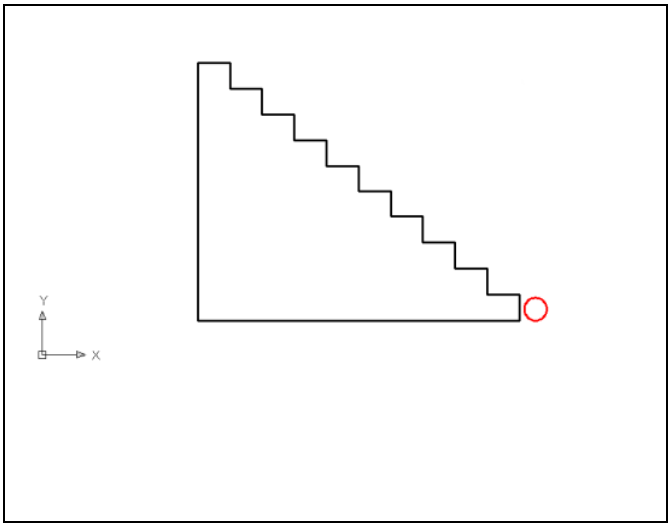

Figure 10.31– Move the Ball Down the Stairs

Written below is the entire motion1.LSP code for your benefit.

```
;;;   motion1.lsp
;;;   a program that moves a ball down 10 stairs
;;;   copyright 1997 by charles w. robbins

(alert "motion1.lsp - copyright 1997 by charles w. robbins. type m1 to start")

;;;   start the program

(defun c:m1 (/)

;;;   select the ball

(setq ss1 (ssget "x" '((8 . "ball"))))          ; select the ball
(setq filename 1000)                            ; start saving files at 1000

;;;    stair while loop

(setq staircounter 0)                           ; set staircounter to 0
(while (< staircounter 10)                      ; 10 increment loop

;;;   horizontal movement loop

(setq counter1 0)                               ; set counter to 0
(while (< counter1 5)                           ; 5 increment loop
  (command "move" ss1 "" "@" "@2,0")            ; move ball 2in. to right
  (c:saveimg (itoa filename) "bmp")             ; save file as a bitmap
  (setq filename (1+ filename))                 ; add one to the filename
  (setq counter1 (1+ counter1))                 ; add one to the counter
)

;;;   vertical movement loop

(setq counter1 0)                               ; set counter to 0
(while (< counter1 4)                           ; 4 increment loop
  (command "move" ss1 "" "@" "@0,-2")           ; move ball 2in. down
  (c:saveimg (itoa filename) "bmp")             ; save file as a bitmap
  (setq filename (1+ filename))                 ; add one to the filename
  (setq counter1 (1+ counter1))                 ; add one to the counter
)

(setq staircounter (1+ staircounter))           ; add one to the staircounter
)

;;;   end of program

)
```