

Making a Complete Detail

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 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 Initget to Setup Keywords for the Getkeyword Function**
6. **Using Getkeyword to Obtain Text from the Keyboard**
7. **Using the If Function to Make Decisions in a Program**
8. **Using Getreal to Obtain a Real Number from the Keyboard**
9. **Using Getint to Obtain an Integer from the Keyboard**
10. **Using Initget, Getkeyword and the If Function to Set Variable Values**
11. **Creating Layers with the Visual AutoLISP Command Function**
12. **Doing the Math in Visual AutoLISP**
13. **Making Point Assignments in Visual AutoLISP**
14. **Drawing in Visual AutoLISP**
15. **Ending the Program**
16. **Saving the Program**
17. **Loading the Program**

Placing Text on Drawings as Easy as Lines and Circles

In this chapter, we will proceed to make a more difficult detail so that by the end of this textbook we will be making entire drawings with the Visual AutoLISP code. This exercise has been a favorite of the architectural design students when they construct custom wall sections with their own routine. In Figure 7.1, we see a partial wall section, but many students have added more detail over the years to develop a more usable tool. In the problem, we will draw a footer, concrete block, sill plate, joist, flooring and wall.

We identify every endpoint of the lines in the footer with a “P” and a number. We continue to label the vertices when the first block is shown on the detail, but we will not continue cataloging the rest of the blocks, since we will array the first concrete block for the number of courses in the foundation wall. Next, we add the sillplate, joist, floor and wall. There are dimensions shown on the sketch that will match the questions the user will be asked. To the right of the footer in Figure 7.1, we show what the footer width (fw) or the sillplate width (sillplate) will be, when the user selects an 8 inch or 12 inch block width. The next sketch shown in Figure 7.2 illustrates the x and y grid that will make up the coordinates for every point.

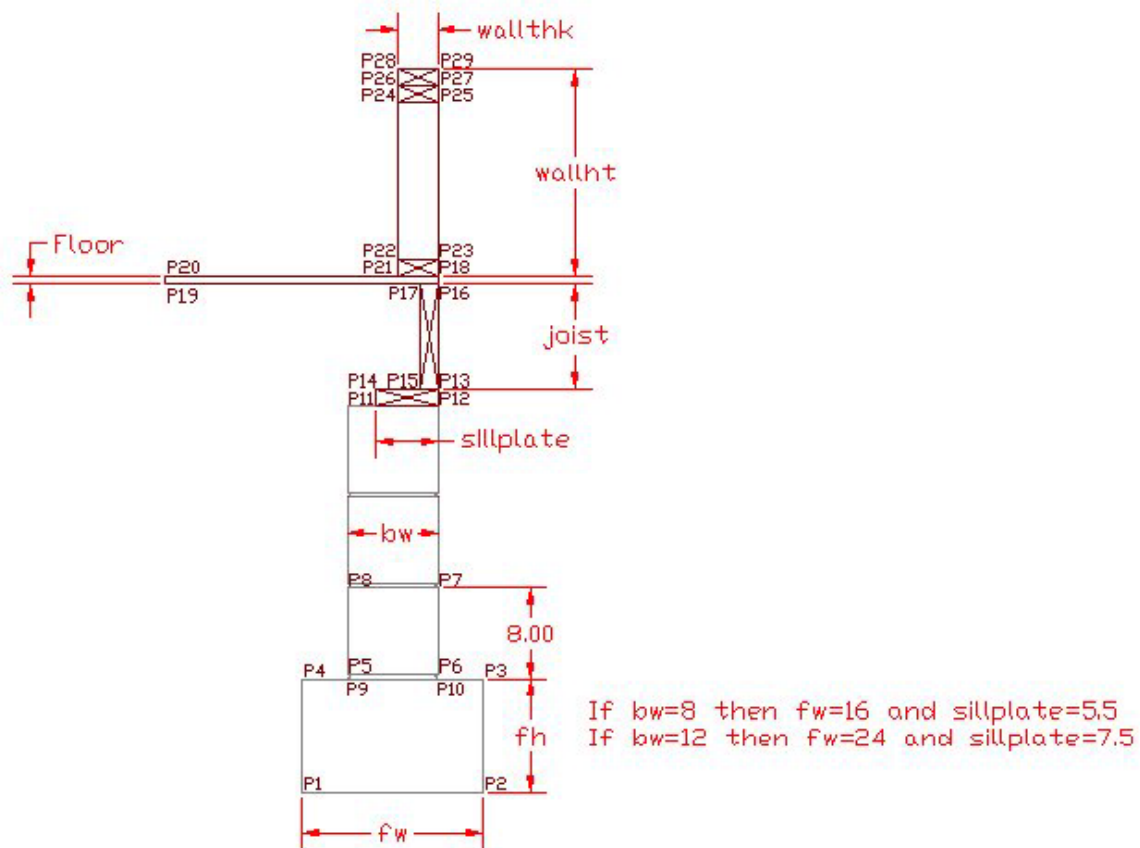


Figure 7.1 – Sketch of the Wallsection

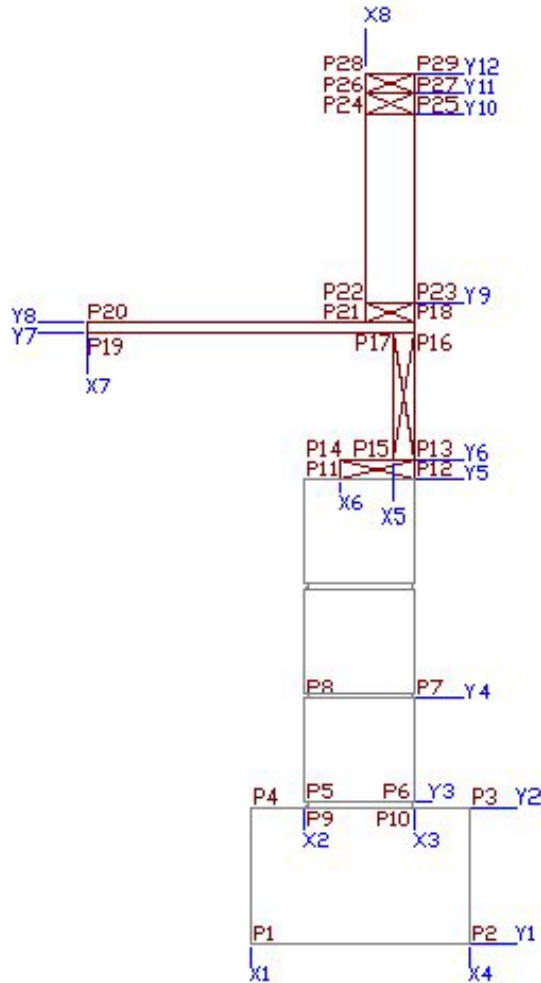


Figure 7.2 – X and Y Grid

This program will have every section of the Construction Code, including some new twists. We will create layers to place the different parts on the drawing. We will draw a concrete block and array the entities to create a foundation wall. We will draw arcs. The programming code will use every step in the Construction Code process. In the table below, we can see those steps listed for our benefit.

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

The math on this problem is very simple, so we will do all the procedures in order on this exercise. The first step we need to take is to launch the Visual LISP Editor in AutoCAD.

Starting the Code by Launching the Visual LISP Editor

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

```
;;; wallsection.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.

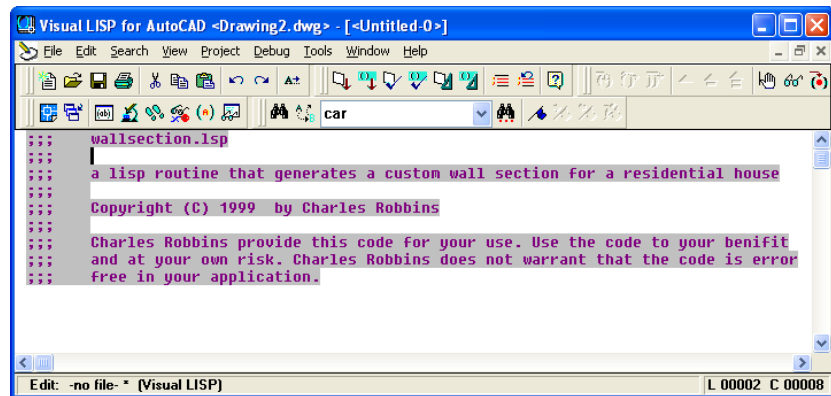


Figure 7.3 – Starting the Wallsection Program

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.

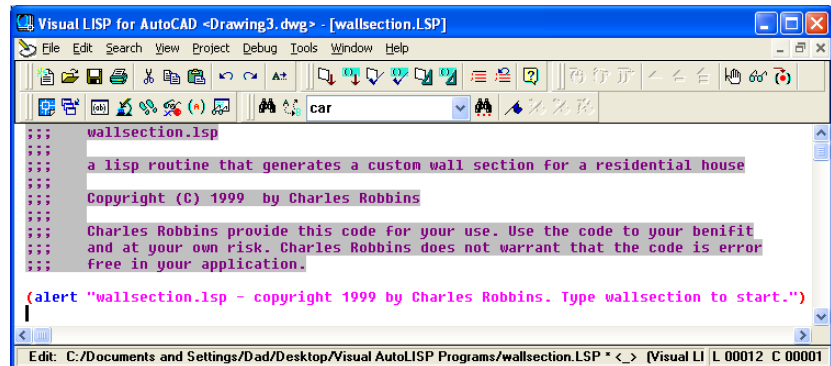


Figure 7.4 – Adding the Alert Expression

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.

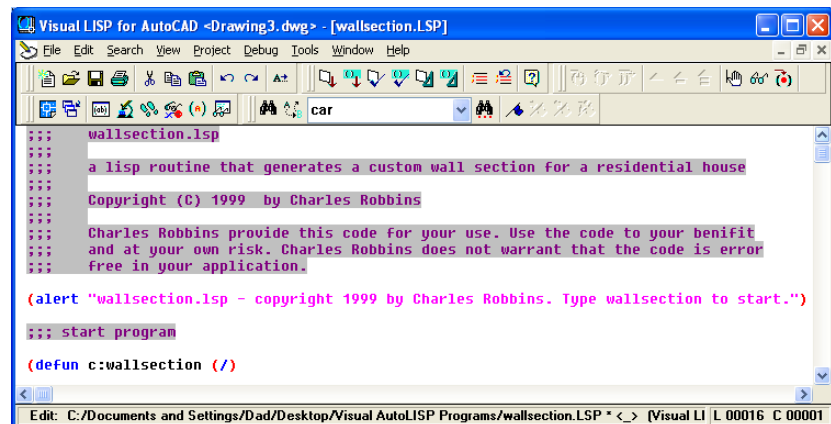


Figure 7.5 – The Defun Expression

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 "wallsection.lsp - copyright 1999 by charles robbins. type wallsection 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 **wallsection** which will be the execution symbol to start the program. Keep in mind the alert message that stated “type wallsection to start”. The alert message text and the **defun** symbol must match. The open and closed parenthesis “**()**” following the **wallsection** 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

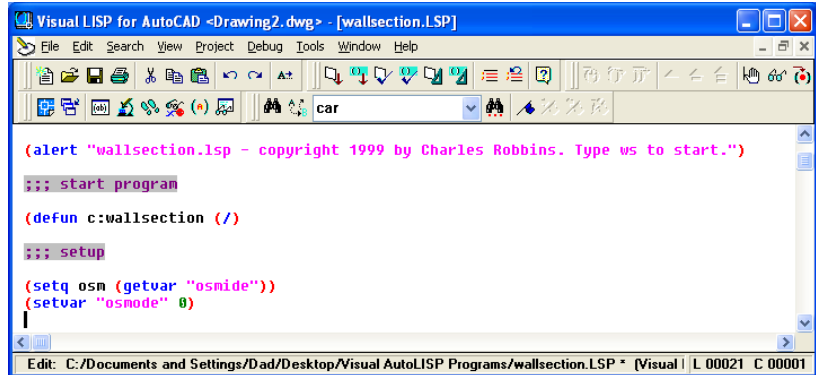


Figure 7.6 – Saving and Turning Off Object Snaps

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	<code>(setq osm (getvar "osmode"))</code>	Answer: 179
Get the AutoCAD version number	<code>(setq osm (getvar "acadver"))</code>	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	<code>(setvar "osmode" 0)</code>	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.

```

Visual LISP for AutoCAD <Drawing3.dwg> - [wallsection.LSP]
File Edit Search View Project Debug Tools Window Help
[Icons] [car]
;;; start program
(defun c:wallsection (/)
  ;;; setup
  (setq osm (getvar "osmode"))
  (setvar "osmode" 0)
  ;;; ask questions
  (setq sp (getpoint "\nPick starting point "))

```

Figure 7.7 – 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.

Periodically we will work at an organization that wants their details 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 detail at this company starts at the X and Y coordinates 14, 10. Then in this wallsection 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 txht the real number 0.125	(setq txht 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 Initget to Setup Keywords for the Getkeyword Function

Whenever we want to use the **getkeyword** 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.

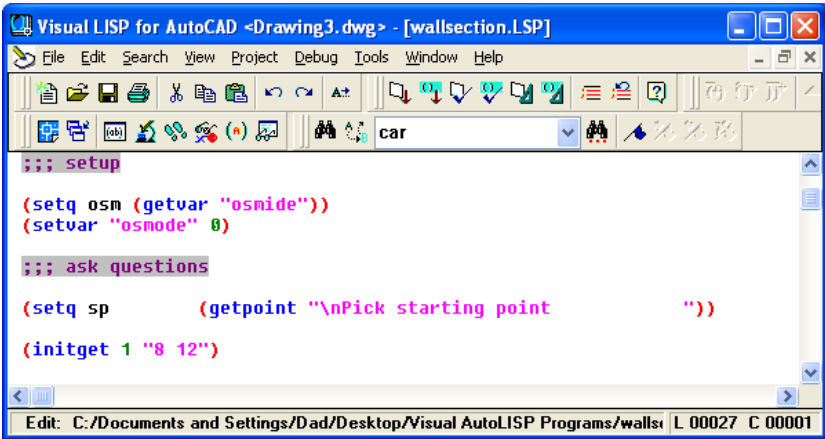


Figure 7.8 – Adding a Initget Expression

Construction Note:

At this stage of the program, we know that the user needs to choose whether the house will have 8 inch or 12 inch blocks in the foundation.

So type the following compound expression:

(initget 1 "8 12")

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 **8** or **12**, so the section of the code is written **"8 12"**.

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 getkeyword function
Example		
Allow the user to type yes (y) or no (n) at the keyboard	<code>(initget 1 "y n")</code>	Returns: nil
Allow the user to type 8 or 12 at the keyboard	<code>(initget 1 "8 12")</code>	Returns: nil

Using Getkeyword 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 **getkeyword** function. The user can type any entry at the keyboard, but the only one matching one of the keywords will be accepted. The **getkeyword** expression is set within the `(setq bw.....)` code.

```

Visual LISP for AutoCAD <Drawing3.dwg> - [wallsection.LSP]
File Edit Search View Project Debug Tools Window Help
car
;;; setup
(setq osm (getvar "osmode"))
(setqvar "osmode" 0)
;;; ask questions
(setq sp (getpoint "\nPick starting point "))
(initget 1 "8 12")
(setq bw (atoi (getkeyword "\nWhat is the block size? [8 12] ")))
Edit: C:/Documents and Settings/Dad/Desktop/Visual AutoLISP Programs/walls L 00028 C 00001

```

Figure 7.9 – Using the Getkeyword Function

So type the following compound expression:

`(setq bw (atoi (getkeyword "\nWhat is the block size? [8 12] ")))`

The information that the user types with the keyboard is stored in the variable name **bw**. 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 7.9 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 **8** or **12** will work.

The **atoi** function in front of the **getkeyword** expression will convert the text string “8” or “12” depending on the user response to an integer. Shown below is a table showing functions that will convert from real numbers, integers and text strings to another format. These are some of the easiest functions to use.

Convert from	To		
Real		Integer	String
Integer	float		itoa
String	atof	atoi	

Figure 7.10 – Conversion Functions

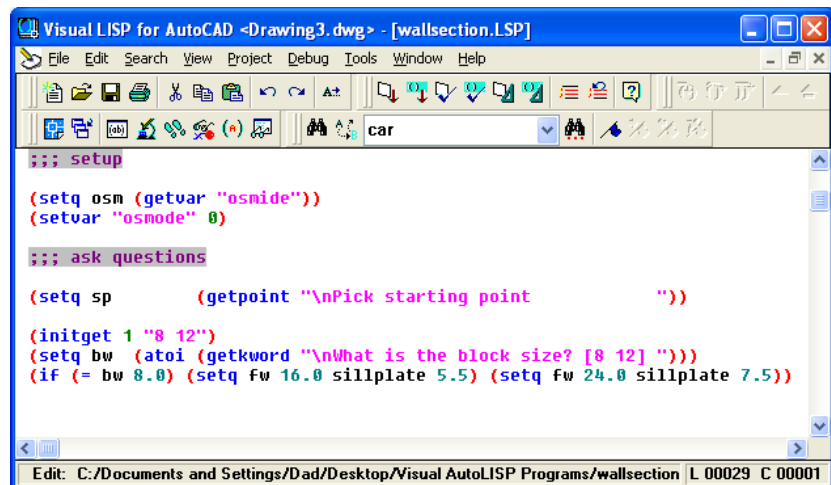
Practice typing the following examples of the **getkword** and **atoi** function at the command line of AutoCAD.

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 bw (getkword "\nWhat is the block size? [8 12]"))	Answer: Do you want tolerances? Then type: 8 “8”
Not responding the proper keyword	(setq fw (getkword "Footer width? [8 12] "))	Answer: Do you want tolerances? Then type: 1 Invalid option keyword and Repeats the initial question

Function	Name	Description
atoi	A Text String to an Integer	Will convert a text string to an integer
Examples		
Change “16” to 16	(atoi “16”)	Answer: 16
Change “24” to 24	(atoi “24”)	Answer: 24
Change the text string in variable fw to 16	(setq fw “16”) (atoi fw)	Answer: 16

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, or carry out the else section if the logical test is false.



```
Visual LISP for AutoCAD -Drawing3.dwg> - [wallsection.LSP]
File Edit Search View Project Debug Tools Window Help
car
;;; setup
(setq osm (getvar "osmode"))
(setvar "osmode" 0)
;;; ask questions
(setq sp (getpoint "\nPick starting point "))
(initget 1 "8 12")
(setq bw (atoi (getkword "\nWhat is the block size? [8 12] ")))
(if (= bw 8.0) (setq fw 16.0 sillplate 5.5) (setq fw 24.0 sillplate 7.5))
Edit: C:/Documents and Settings/Dad/Desktop/Visual AutoLISP Programs/wallsection L 00029 C 00001
```

Figure 7.11 – The If Function with Then and Else Statements

Construction Note:

We will use the **setq** function to obtain a footer width, which is twice the block size. If the block size is 8 inches then the footer height will be 2 x 8 inches or 16 inches. If the block size is 12 inches then the footer height will be 2 x 12 inches or 24 inches. In this exercise we can use the answer to one question as the solution to other inputs.

The **if** function is arranged to work in a more complex fashion than other AutoLISP tools. It 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 “What is the block size [8 12]”, then the 8 or 12 is stored in the variable **bw**. So in the logical test we are asking does the text string “8” equal “8”. 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 Wallsection routine:

```
(if (= bw 8.0) (setq fw 16.0 sillplate 5.5)(setq fw 24.0 sillplate 7.5))
```

The **setq** function can be written as shown in Figure 7.11 where there is a series of variables with a definition following each holder of changeable data. Therefore if the logical test of **(= bw 8.0)** is true, then **fw** will contain **16.0** and **sillplate** will hold **5.5**. This if expression contains an or else phrase for a logical test result of false, so if the logical test of **(= bw 12.0)** is false, then **fw** will contain **24.0** and **sillplate** will hold **7.5**.

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 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 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.

```

Visual LISP for AutoCAD <Drawing3.dwg> - [wallsection.LSP]
File Edit Search View Project Debug Tools Window Help
car
::: ask questions
(setq sp (getpoint "\nPick starting point "))
(initget 1 "8 12")
(setq bw (atoi (getkword "\nWhat is the block size? [8 12] ")))
(if (= bw 8.0) (setq fw 16.0 sillplate 5.5) (setq fw 24.0 sillplate 7.5))
(setq fh (getreal "\nWhat is the footer height? "))

```

Figure 7.12 – Using the Getreal Function

So type the following compound expression:

```
(setq fh (getreal "\nWhat is the footer height  "))
```

The information that the user types with the keyboard is stored in the variable name **fh**. 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 7.12, 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		
Ask for a number, user types a whole number and the reply is changed to a real number	<pre>(setq txtht (getreal "\nWhat is the text height?"))</pre>	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	<pre>(setq txtht (getreal "\nWhat is the text height?"))</pre>	Answer: What is the text height? Then type: 1/8 0.125

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 courses)** 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.

```
Visual LISP for AutoCAD <Drawing3.dwg> - [wallsection.LSP]
File Edit Search View Project Debug Tools Window Help
[Toolbar]
car
;; ask questions
(setq sp (getpoint "\nPick starting point  "))
(initget 1 "8 12")
(setq bw (atoi (getkword "\nWhat is the block size? [8 12] ")))
(if (= bw 8.0) (setq fw 16.0 sillplate 5.5) (setq fw 24.0 sillplate 7.5))
(setq fh (getreal "\nWhat is the footer height?  "))
(setq courses (getint "\nHow many courses of block?  "))
```

Figure 7.13 – Using the Getint Function

So type the following compound expression:

```
(setq courses (getint "\nHow many courses of block? "))
```

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	<pre>(setq coats (getint "\nHow many coats of enamel? "))</pre>	Answer: How many coats of enamel? Then type: 1 Returns: 1
Asking a question and inputting a real number	<pre>(setq coats (getint "\nHow many coats of enamel? "))</pre>	Answer: How many coats of enamel? Then type: 1.0 Returns: Requires an integer value and How many coats of enamel?

Using Initget, Getkword and the If Function to Set Variable Values

Construction Note:

We will use the **setq** function to obtain the floor joist size, which can be 2 x 10, 2 x 12, 2 x 16 or 2 x 20. When the joist size is selected, we will set the actual wood size for the 2 x 10 to be 1.5 x 9.25, for a 2 x 12 to be 1.5 x 11.25, for a 2 x 16 to be 1.5 x 15.25, and for a 2 x 20 to be 1.5 x 19.25.

Again we will use the **Initget** function to create a list of text choices for the next line of code using the **getkword** function. Type the following:

```
(initget 1 "10 12 16 20")
```

This will allow user to type the text strings 10, 12 16 or 20.

```

Visual LISP for AutoCAD -Drawing1.dwg> - [wallsection.LSP]
File Edit Search View Project Debug Tools Window Help
[Toolbar icons]
car
(setq fh (getreal "\nWhat is the footer heighth? "))
(setq courses (getint "\nHow many courses of block? "))

(initget 1 "10 12 16 20")
(setq joist (getkword "\nWhat is the size of joist, 2 x [ 10 12 16 20 ] "))
(if (= joist "10") (setq joist 9.25))
(if (= joist "12") (setq joist 11.25))
(if (= joist "16") (setq joist 15.25))
(if (= joist "20") (setq joist 19.25))
  
```

Figure 7.14 – Determining the Joist Size

Then we type the **getkword** code and set the answer to the variable **joist**. Type the following:

```
(setq joist (getkword "\nWhat is the size of joist, 2 x [ 10 12 16 20 ]  "))
```

Right after assigning the text value to the variable **joist**, then we will use the **if** function to reassign the more complex dimension of the true wood size to the variable **joist**. Although this technique uses more lines of code, the user does not have to remember exact joist dimensions but only their trade designations. Type the following lines:

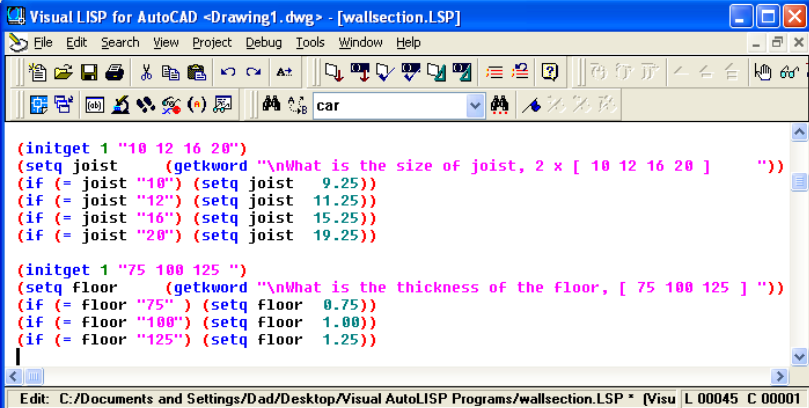
```
(if (= joist "10") (setq joist 9.25))  
(if (= joist "12") (setq joist 11.25))  
(if (= joist "16") (setq joist 15.25))  
(if (= joist "20") (setq joist 19.25))
```

Now that the joist height dimension is set, we will need to inquiry about the floor thickness.

Again we will use the **initget** function to create a list of text choices for the next line of code using the **getkword** function. Type the following:

```
(initget 1 "75 100 125 ")
```

This will allow user to type the text strings 75, 100 or 125.



The screenshot shows a Visual LISP for AutoCAD window titled "Visual LISP for AutoCAD <Drawing1.dwg> - [wallsection.LSP]". The window contains the following code:

```
(initget 1 "10 12 16 20")  
(setq joist (getkword "\nWhat is the size of joist, 2 x [ 10 12 16 20 ]  "))  
(if (= joist "10") (setq joist 9.25))  
(if (= joist "12") (setq joist 11.25))  
(if (= joist "16") (setq joist 15.25))  
(if (= joist "20") (setq joist 19.25))  
  
(initget 1 "75 100 125 ")  
(setq floor (getkword "\nWhat is the thickness of the floor, [ 75 100 125 ]  "))  
(if (= floor "75") (setq floor 0.75))  
(if (= floor "100") (setq floor 1.00))  
(if (= floor "125") (setq floor 1.25))
```

The status bar at the bottom of the window shows "Edit: C:/Documents and Settings/Dad/Desktop/Visual AutoLISP Programs/wallsection.LSP * [Visu L 00045 C 00001".

Figure 7.15 – Determining the Floor Size

Then we type the **getkword** code and set the answer to the variable **floor**. Type the following:

```
(setq floor (getkword "\nWhat is the thickness of the floor, [ 75 100 125 ]  "))
```

Right after assigning the text value to the variable **floor**, then we will use the **if** function to reassign the exact dimension of the true wood size to the variable **floor**. Although this technique uses more lines of code, the user does not have to type the decimal at the command line. Type the following lines:

```
(if (= floor "75") (setq floor 0.75))  
(if (= floor "100") (setq floor 1.00))  
(if (= floor "125") (setq floor 1.25))
```

Now that the floor thickness is set, we will need to inquiry about the wall thickness.

Again we will use the **Initget** function to create a list of text choices for the next line of code using the **getkeyword** function. Type the following:

```
(initget 1 "4 6 ")
```

This will allow user to type the text strings 4 or 6.

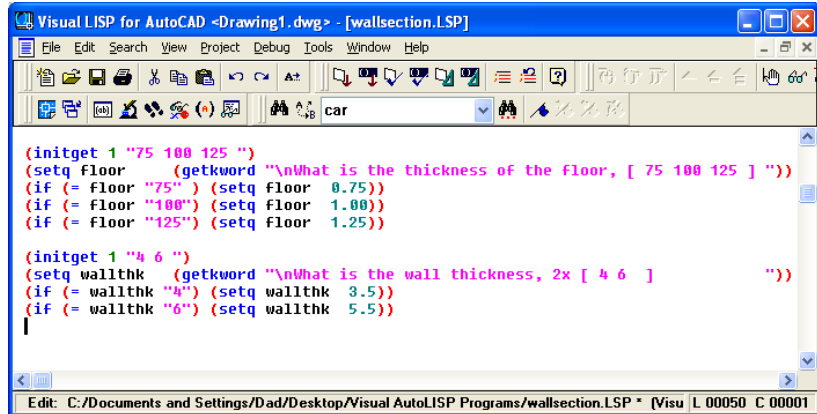


Figure 7.16 – Determining the Wall Thickness

Then we type the **getkeyword** code and set the answer to the variable **wallthk**. Type the following:

```
(setq wallthk (getkeyword "\nWhat is the wall thickness, 2x [ 4 6 ] "))
```

Right after assigning the text value to the variable **wallthk**, then we will use the **if** function to reassign the exact dimension of the true wood size to the variable **wallthk**. Again, this technique uses more lines of code, so the user does not have to type the decimal at the command line. Type the following lines:

```
(if (= wallthk "4") (setq wallthk 3.5))
(if (= wallthk "6") (setq wallthk 5.5))
```

Now that the wall thickness is set, we will need to inquiry about the wall height.

For the wall height the user can type any real number at the command line when prompted with the question, **"What is the wall height?"** Remember the user cannot type in a whole number with a feet, inches and a fraction, which many home builders like to use. The response needs to be typed in inches and decimals.

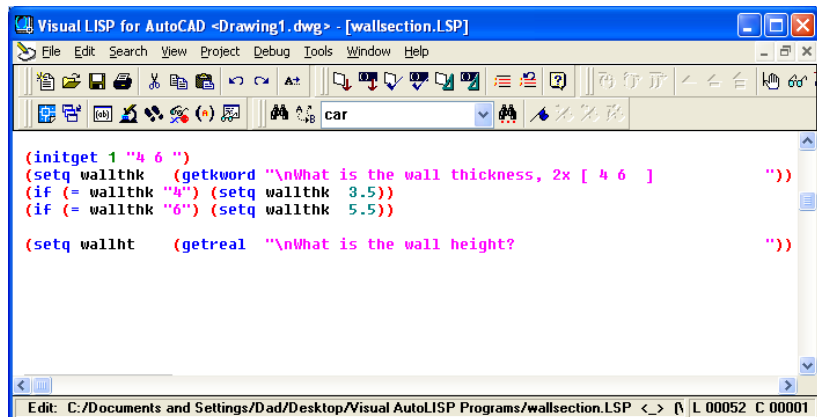


Figure 7.17 – Determining the Wall Height

In other programs, we will write a small routine allowing for feet, inches and fractions. For now, type the following:

```
(setq wallht (getreal "\nWhat is the wall height?"))
```

Now we will create the layers we need for this project.

Creating Layers with the Visual AutoLISP Command Function

In all of the other Visual AutoLISP programs, we run the code on the layer that we want to have the orthographic view or notes. In this detail, we will place the entities on unique layers such as footer, block, wood, dimension and text. Then we will lame addition layers for center, hidden and section lines.

```

(setq wallht (getreal "\nWhat is the wall height?"))
;;; setup layers
(command "layer" "n" "footer" "c" "8" "footer")
(command "layer" "n" "block" "c" "8" "block")
(command "layer" "n" "wood" "c" "14" "wood")
(command "layer" "n" "dimension" "c" "red" "dimension")
(command "layer" "n" "text" "c" "green" "text")
  
```

Figure 7.18 – Creating Layer in AutoLISP

An easy way to make a layer for a detail is to use the command function and to follow the creating a new layer process. This can be harder for individuals just newly training with AutoCAD, since many computer aided design tools are now in dialogue boxes and we cannot easily view all the options that are available with a command function. We will share the most common options in the table below.

Command Layer Function	Command Layer Function
<code>(command "layer" "n" "footer" "")</code>	Makes a new layer named “footer”
<code>(command "layer" "c" "8" "footer" "")</code>	Sets the layer color to “color 8” for layer named “footer”
<code>(command "layer" "lt" "center" "center" "")</code>	Sets the layer linetype to “center” for layer named “center”
<code>(command "layer" "s" "footer" "")</code>	Sets the current layer as “footer”
<code>(command "layer" "f" "footer" "")</code>	Freezes the layer named “footer”
<code>(command "layer" "t" "footer" "")</code>	Thaws the layer named “footer”
<code>(command "layer" "on" "footer" "")</code>	Turns the layer named “footer” on
<code>(command "layer" "off" "footer" "")</code>	Turns the layer named “footer” off
<code>(command "layer" "lo" "footer" "")</code>	Locks the layer named “footer”
<code>(command "layer" "u" "footer" "")</code>	Unlocks the layer named “footer”

We can combine layer options such as new and color and create a command line expression that will both create a new layer and set the color for that layer. When we use the color option “c”, the next item is the name or number of the color, followed by the name of the layer. To end the layer command expression, place an open and closed quote “” at the end of the code and then a closed parenthesis.

Type the following lines in the wallsection program.

```
;;; setup layers
```

```
(command "layer" "n" "footer" "c" "8" "footer" "")
(command "layer" "n" "block" "c" "8" "block" "")
(command "layer" "n" "wood" "c" "14" "wood" "")
(command "layer" "n" "dimension" "c" "red" "dimension" "")
(command "layer" "n" "text" "c" "green" "text" "")
```

Now we will make the center, hidden and section layers in the routine. We may be using different colors or layer names than what your organization uses, so feel free to make changes to the layer name or color that defines your group's standard layers. changes

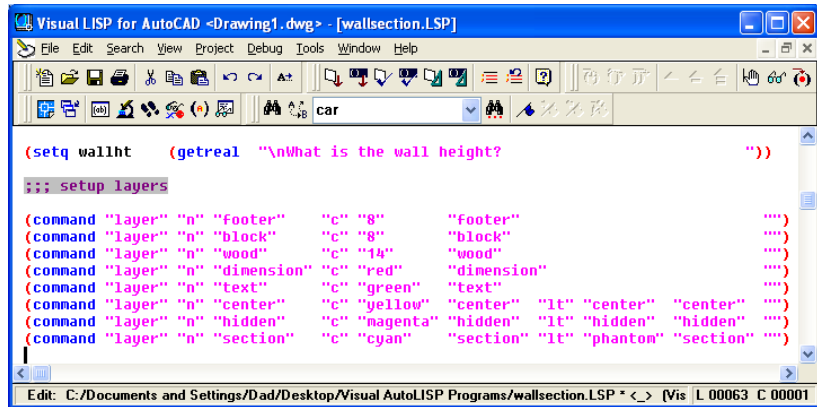


Figure 7.19 – Creating Layer with Special Linetypes

If the layer name, color and linetype already exists in the drawing, nothing will change when these lines of code execute.

Type the following lines in the wallsection program.

```
(command "layer" "n" "center" "c" "yellow" "center" "lt" "center" "center" "")
(command "layer" "n" "hidden" "c" "magenta" "hidden" "lt" "hidden" "hidden" "")
(command "layer" "n" "section" "c" "cyan" "section" "lt" "phantom" "section" "")
```

Doing the Math in Visual AutoLISP

Now, we will return back to the middle of the program and finish the math section of the code. Again the **setq** function is the choice for assigning values to the variables X1, X2, X3, X4, X5, X6, X7, X8, Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8, Y9, Y10, Y11 and Y12.

The **car** function is used with variable **sp** (the starting point) to extract the x-coordinate of the starting point list. If the starting point is (4, 3, 0) then **(car sp)** will return as 4 and be assigned to the variable X1. So the **car** function returns the first number in the list.

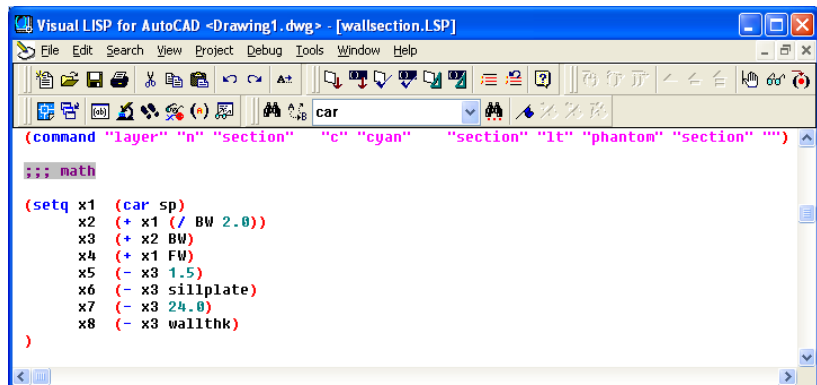


Figure 7.20 – Defining the X-Ordinates

That explains the use of **car** to find the coordinates **x1**, now we have to continue down the X grid to obtain value for **x2**. To obtain the **x2** coordinate, use the addition function **+** to add to the **x1** value. To get the number to add to the **x1**, we have to divide the block width **BW** by two using the divide function **/** like so. Notice that the function is written first, followed by the numerator **BW** and then the denominator **2.0**.

(/ BW 2.0)

And place that expression in the addition expression to build a compound expression.

(+ x1 (/ BW 2.0))

Now assign the value to **x2**

x2 (+ x1 (/ width 2.0))

Type the following lines in the wallsection program using the sketch in Figure 7.2 to find the X ordinate the dimension that defines the horizontal measurements.

;;; math

```
(setq x1 (car sp)
      x2 (+ x1 (/ BW 2.0))
      x3 (+ x2 BW)
      x4 (+ x1 FW)
      x5 (- x3 1.5)
      x6 (- x3 sillplate)
      x7 (- x3 24.0)
      x8 (- x3 wallthk)
)
```

Likewise, the **cadr** function is used with variable **sp** (the starting point) to extract the y-coordinate of the starting point. Again, if the starting point is (4, 3, 0) then (**cadr sp**) will return as 3 and be assigned to the variable **y1**. So the **cadr** function returns the second number in the list.

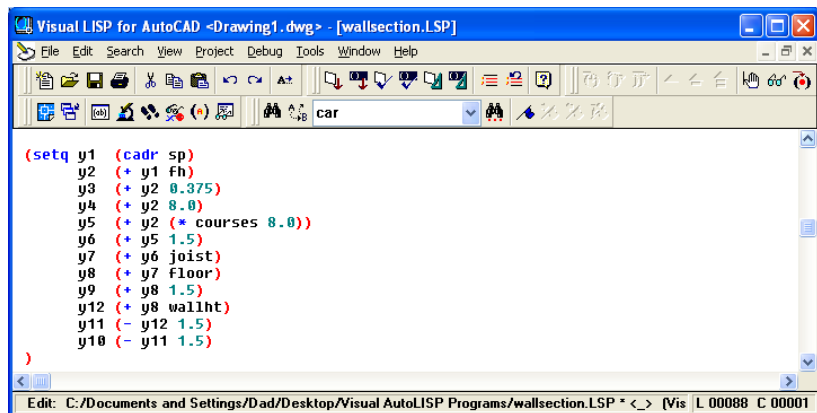


Figure 7.21 – Defining the Y-Ordinates

To find the value for the adaptable **y2**, we add the variable for the footer height **fh**. We use the adding LISP function **+** to make the expression

y2 (+ y1 fh)

Other than for the measurement **y5**, all of the vertical distances are compiled by using the adding or subtracting functions. Type the following lines in the wallsection program using the sketch in Figure 7.2 to find the Y ordinate the dimension that defines the vertical measurements.

```
(setq y1 (cadr sp)
      y2 (+ y1 fh)
      y3 (+ y2 0.375)
      y4 (+ y2 8.0)
      y5 (+ y2 (* courses 8.0))
      y6 (+ y5 1.5)
      y7 (+ y6 joist)
      y8 (+ y7 floor)
      y9 (+ y8 1.5)
      y12 (+ y8 wallht)
      y11 (- y12 1.5)
      y10 (- y11 1.5)
)
```

Making Point Assignments in Visual AutoLISP

One of the easiest sections of code for a new or experienced programmer to accomplish is the point assignments, where one assigns X and Y coordinates to the point vertexes. Basically, we did the work when we made the Wallsection sketch. When we define the points for the footer, remember when we read that coordinate P1 is (X1, Y1). P2 is (X4, Y1). P3 is (X4, Y2). P4 is (X1, Y2). Now we write a **setq** expression setting these grids coordinates to the points **p1**, **p2**, **p3**, **p4** through **p29**.

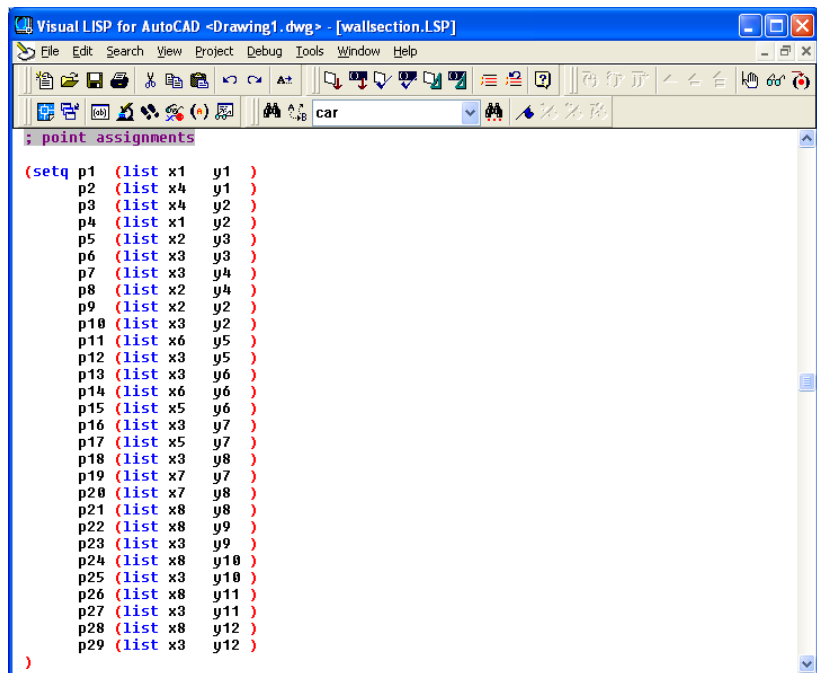


Figure 7.22 – Defining the Point Assignments

The **list** function can create an X, Y and Z coordinate by typing the appropriate X and Y values after the function name. We do not need to add the Z coordinate if the value is going to be zero. (See Figure 7.22)

Type the following lines in the wallsection program using the sketch in Figure 7.2 to find the X and Y coordinate for each point.

; point assignments

```
(setq p1 (list x1 y1 )
      p2 (list x4 y1 )
      p3 (list x4 y2 )
      p4 (list x1 y2 )
      p5 (list x2 y3 )
      p6 (list x3 y3 )
      p7 (list x3 y4 )
      p8 (list x2 y4 )
      p9 (list x2 y2 )
      p10 (list x3 y2 )
      p11 (list x6 y5 )
      p12 (list x3 y5 )
      p13 (list x3 y6 )
      p14 (list x6 y6 )
      p15 (list x5 y6 )
      p16 (list x3 y7 )
      p17 (list x5 y7 )
      p18 (list x3 y8 )
      p19 (list x7 y7 )
      p20 (list x7 y8 )
      p21 (list x8 y8 )
      p22 (list x8 y9 )
      p23 (list x3 y9 )
      p24 (list x8 y10 )
      p25 (list x3 y10 )
      p26 (list x8 y11 )
      p27 (list x3 y11 )
      p28 (list x8 y12 )
      p29 (list x3 y12 )
)
```

Drawing in Visual AutoLISP

Before drawing the first line in the Wallsection detail, we will set the current layer as “footer”. Before we draw an entity using the command functions of line, pline and arc tools, we will continually place the article on the precise drawing layer. Type the following code to set the current layer to “footer”.

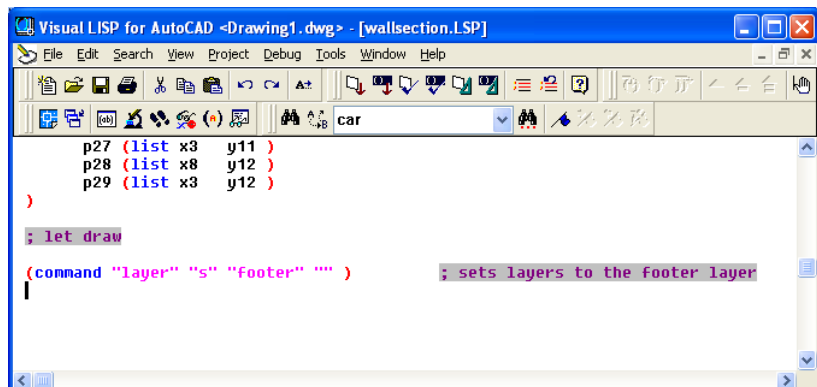


Figure 7.23 – Setting a Footer Layer as Current

(command "layer" "s" "footer" "")

Now that the layer is set to footer, we will proceed to draw the footer which has four points. When automatically drawing any entity in AutoCAD, the programmer uses the **command** function which evokes any AutoCAD standard command. We have to state this rule, since ARX commands typed at the command line like Render or Rotate3D need to be executed differently, which we did in Chapter 6 with the **saveimg** function. After the **command** function is typed, the command **"line"** follows in quotes, then by the point vertexes **p1 p2 p3 p4** of the box and finally **"c"** to close the polygon.

Type the following code:

;; lets draw

(command "line" p1 p2 p3 p4 "c")

And the LISP routine draws a rectangle representing the footer on the AutoCAD graphical display. Next, we set the current layer to block, so the concrete block in the foundation wall will be on a unique layer. Some students will make a layer called foundation and place both the footer and the blocks on one layer. Type this next code.

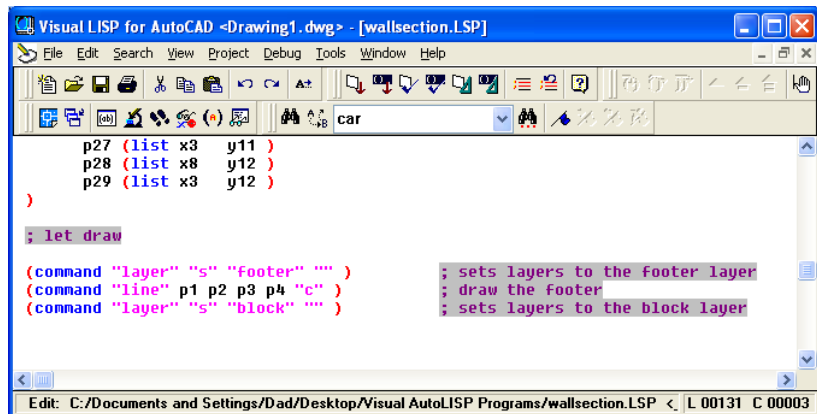


Figure 7.24 – Setting a Block Layer as Current

(command "layer" "s" "block" "") ; sets layers to the block layer

Now we want to draw the first concrete block using the pline drawing tool, so there will be a single entity instead of four lines. The LISP expression for **"pline"** is very much the same as **"line"** when typing in the syntax. Type the following text in your Wallsection program.

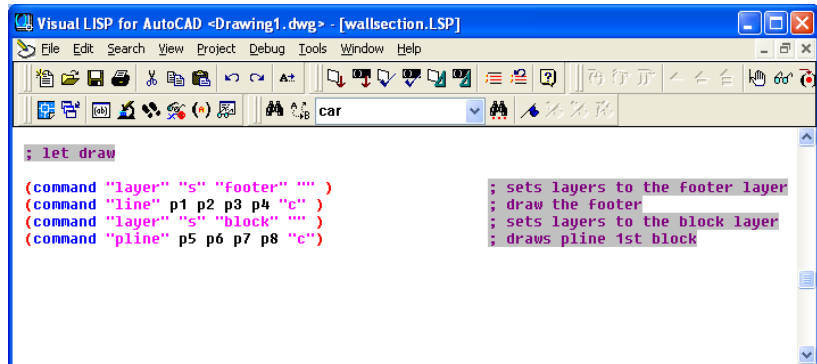


Figure 7.25 – Draw the Concrete Block with Pline

(command "pline" p5 p6 p7 p8 "c") ; draws pline 1st block

Now, we do not want to stop a running program to select an entity to move, copy or array, so we are going to obtain the last polyline by using the **ssget "L"** function. We will place the information regarding the last unit into a variable called **ss1**. When we want to modify this article, we just refer to **ss1**.

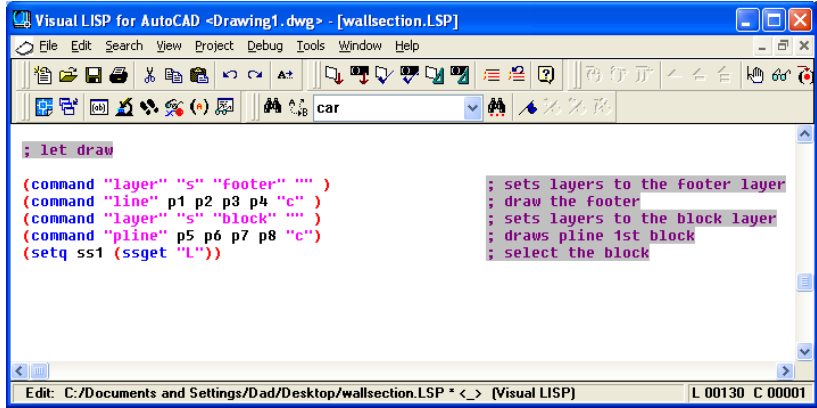


Figure 7.26 – Select the Last Entity Drawn

To assign the last item drawn, type the following expression.

(setq ss1 (ssget "L")) ; select the block

Practice typing the following examples of the **ssget "L"** function at the command line of AutoCAD.

Function	Name	Description
ssget	Obtain a Selection Set	Allows the user to create a selection set by picking entities on the graphical display
Examples		
For picking with a mouse	(setq ss1 (ssget))	Selection Set: 7
Last entity drawn	(setq ss1 (ssget "L"))	Selection Set: 9
Select the last entity selected	(setq ss1 (ssget "P"))	Selection Set: 10
Selects all the entities	(setq ss1 (ssget "A"))	Selection Set: 3

We will then draw the two arcs that represent the mortar joint between the footer and the block. To draw an arc, we will want to place the points in a counterclockwise manner in the AutoCAD drawing system. We will use the Start – End – Radius method for forming the arc.

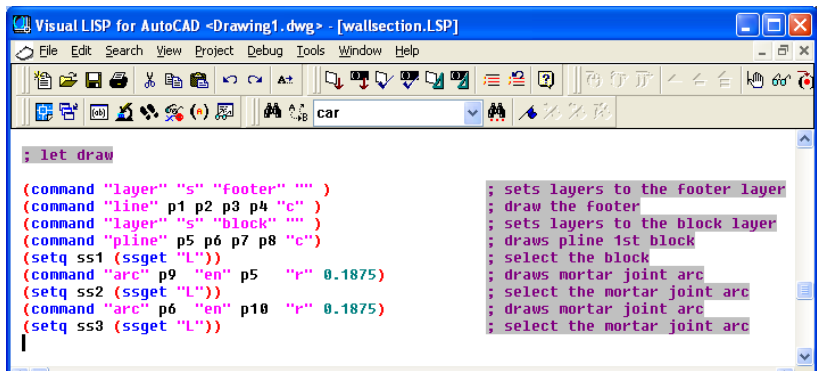


Figure 7.27 – Draws the Mortar Joint

We will opt for **p9** as the arc starting point, and **p5** for the end point. The mortar joint is 0.375 so the radius will be **0.1875**. After typing the **command "arc"** expression, store information each about the arc using the **ssget "L"** function.

Type the following expressions and comments.

```
(command "arc" p9 "en" p5 "r" 0.1875) ; draws mortar joint arc
(setq ss2 (ssget "L")) ; select the mortar joint arc
(command "arc" p6 "en" p10 "r" 0.1875) ; draws mortar joint arc
(setq ss3 (ssget "L")) ; select the mortar joint arc
```

Now we have the concrete block and mortar joint symbolized by the two arcs drawn on the graphical display and stored in memory, so we can easily array the entities in the detail. The array tool is a two part command, the first section we select objects and the second part we execute the function.

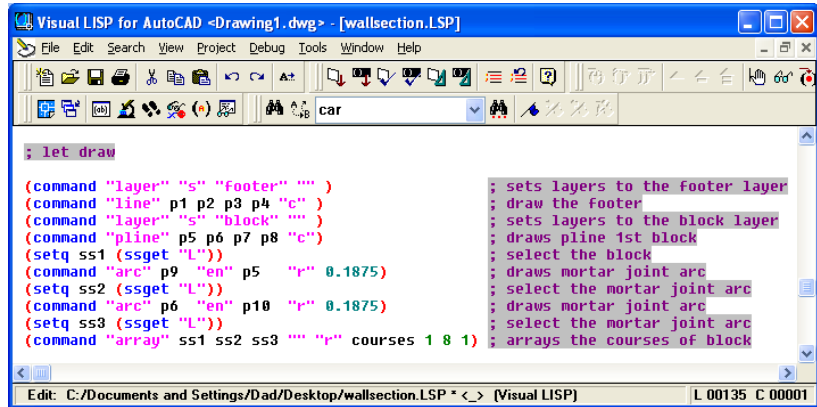


Figure 7.28 – Array the Blocks and Mortar Joint

Type the following expression and comments.

```
(command "array" ss1 ss2 ss3 "" "r" courses 1 8 1) ; arrays the courses of block
```

The objects we select are **ss1 ss2** and **ss3**. Use the double quotes **""** as an Enter to proceed the next element of the array command. The **"r"** is for the rectangle array. The number of rows is the variable **courses** and the number of columns is **1**. The distance between rows is **8** and the space between columns is **1**. The rectangular array is a very straight forward command.

Finally to finish the drawing section of the Construction Code, we will set the current layer to **"wood"** and then draw the sillplate, joist, floor, sole plate, stud, and both top plates. The outside of the wood is made with a polyline and the crossed lines inside the polyline represent cutting line when we inspect the detail. Again, Figure 7.2 is an excellent reference for checking points for the **pline** and **line** functions.

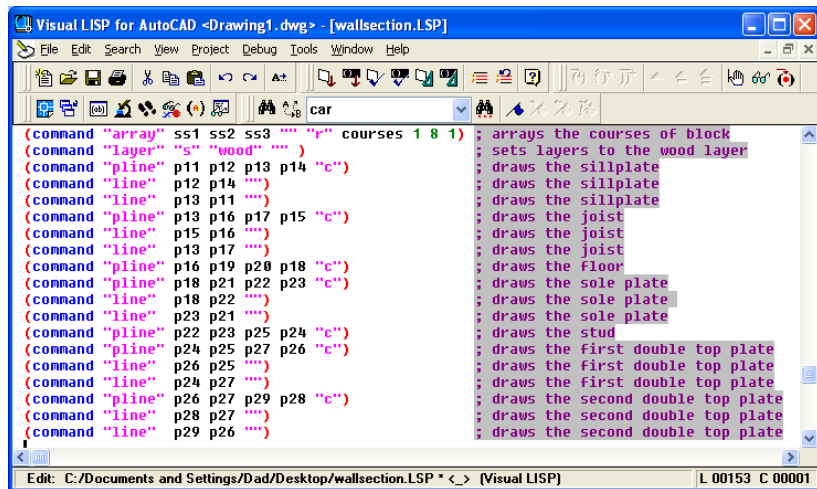


Figure 7.29 – Draws the Wood Components

Type the following expressions and comments.

<code>(command "layer" "s" "wood" "")</code>	<code>; sets layers to the wood layer</code>
<code>(command "pline" p11 p12 p13 p14 "c")</code>	<code>; draws the sillplate</code>
<code>(command "line" p12 p14 "")</code>	<code>; draws the sillplate</code>
<code>(command "line" p13 p11 "")</code>	<code>; draws the sillplate</code>
<code>(command "pline" p13 p16 p17 p15 "c")</code>	<code>; draws the joist</code>
<code>(command "line" p15 p16 "")</code>	<code>; draws the joist</code>
<code>(command "line" p13 p17 "")</code>	<code>; draws the joist</code>
<code>(command "pline" p16 p19 p20 p18 "c")</code>	<code>; draws the floor</code>
<code>(command "pline" p18 p21 p22 p23 "c")</code>	<code>; draws the sole plate</code>
<code>(command "line" p18 p22 "")</code>	<code>; draws the sole plate</code>
<code>(command "line" p23 p21 "")</code>	<code>; draws the sole plate</code>
<code>(command "pline" p22 p23 p25 p24 "c")</code>	<code>; draws the stud</code>
<code>(command "pline" p24 p25 p27 p26 "c")</code>	<code>; draws the first double top plate</code>
<code>(command "line" p26 p25 "")</code>	<code>; draws the first double top plate</code>
<code>(command "line" p24 p27 "")</code>	<code>; draws the first double top plate</code>
<code>(command "pline" p26 p27 p29 p28 "c")</code>	<code>; draws the second double top plate</code>
<code>(command "line" p28 p27 "")</code>	<code>; draws the second double top plate</code>
<code>(command "line" p29 p26 "")</code>	<code>; draws the second double top plate</code>

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)`

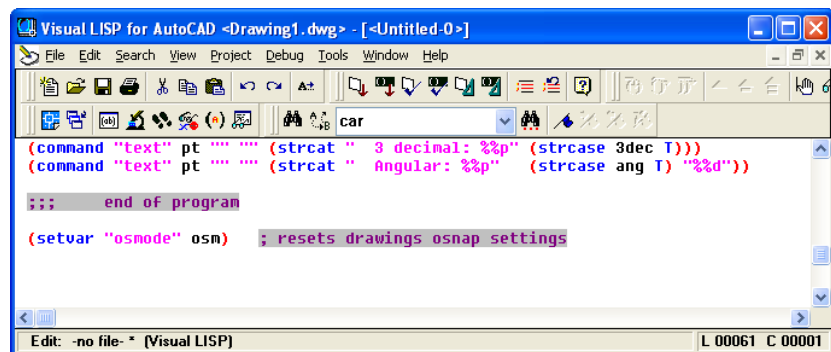


Figure 7.30 – End of Program

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)`

)

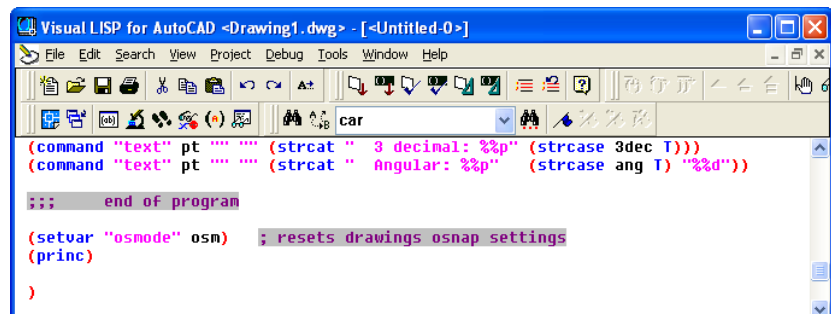


Figure 7.31 – Using the Princ Function

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 "wallsection" 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 7.32, "wallsection.LSP successfully loaded"

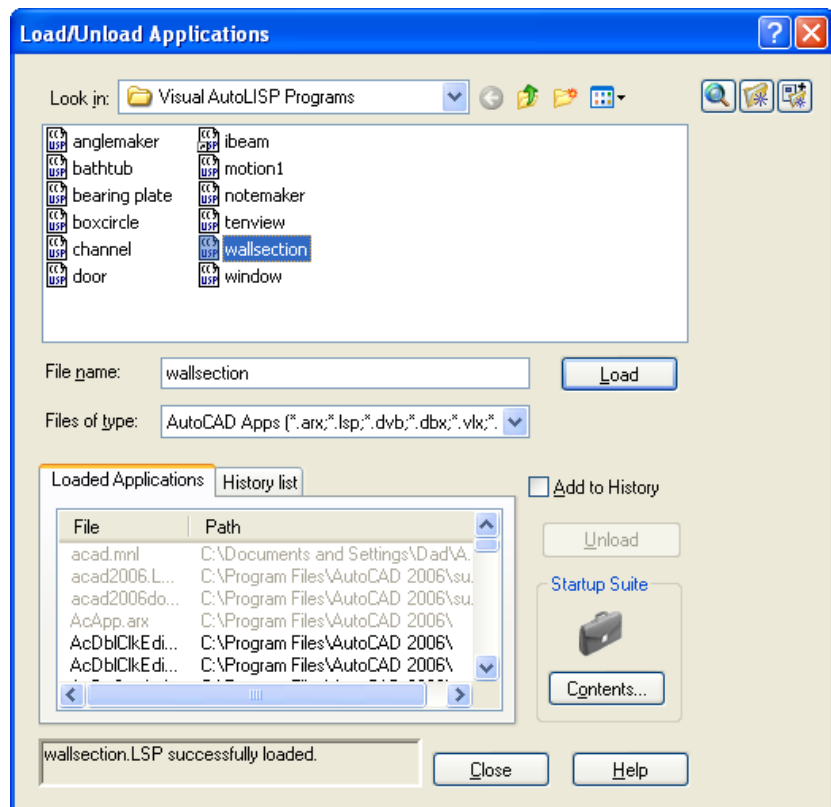


Figure 7.32 – Loading the Wallsection Program

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.

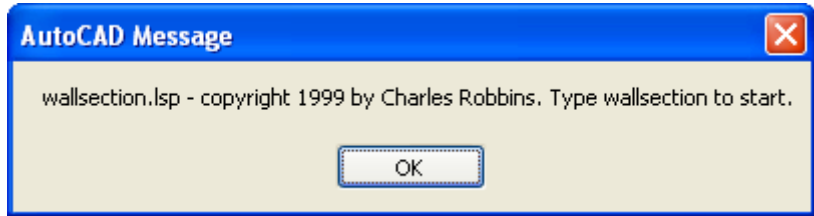


Figure 7.33 – The Alert Message

Running the Program

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

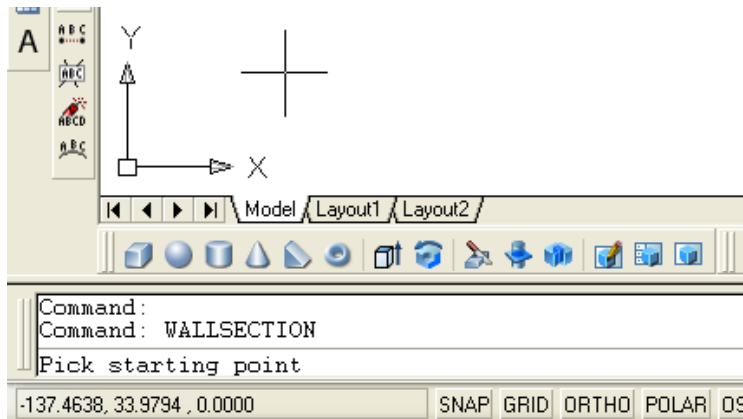


Figure 7.34 – Starting the Program

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 wallsection.LSP code for your benefit.

```

;;; wallsection.lsp
;;;
;;; a lisp routine that generates a custom wall section for a residential house
;;;
;;; Copyright (C) 1999 by Charles Robbins
;;;
;;; Charles Robbins provide this code for your use. Use the code to your benefit
;;; and at your own risk. Charles Robbins does not warrant that the code is error
;;; free in your application.

(alert "wallsection.lsp - copyright 1999 by Charles Robbins. Type wallsection to start.")

;;; start program

(defun c:wallsection (/)

```

```

;;; setup

(setq osm (getvar "osmode"))
(setvar "osmode" 0)

;;; ask questions

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

(initget 1 "8 12")
(setq bw (atoi (getkeyword "\nWhat is the block size? [8 12] ")))
(if (= bw 8.0) (setq fw 16.0 sillplate 5.5) (setq fw 24.0 sillplate 7.5))

(setq fh (getreal "\nWhat is the footer height? "))
(setq courses (getint "\nHow many courses of block? "))

(initget 1 "10 12 16 20")
(setq joist (getkeyword "\nWhat is the size of joist, 2 x [ 10 12 16 20 ] "))
(if (= joist "10") (setq joist 9.25))
(if (= joist "12") (setq joist 11.25))
(if (= joist "16") (setq joist 15.25))
(if (= joist "20") (setq joist 19.25))

(initget 1 "75 100 125 ")
(setq floor (getkeyword "\nWhat is the thickness of the floor, [ 75 100 125 ] "))
(if (= floor "75") (setq floor 0.75))
(if (= floor "100") (setq floor 1.00))
(if (= floor "125") (setq floor 1.25))

(initget 1 "4 6 ")
(setq wallthk (getkeyword "\nWhat is the wall thickness, 2x [ 4 6 ] "))
(if (= wallthk "4") (setq wallthk 3.5))
(if (= wallthk "6") (setq wallthk 5.5))

(setq wallht (getreal "\nWhat is the wall height? "))

;;; setup layers

(command "layer" "n" "footer" "c" "8" "footer" "")
(command "layer" "n" "block" "c" "8" "block" "")
(command "layer" "n" "wood" "c" "14" "wood" "")
(command "layer" "n" "dimension" "c" "red" "dimension" "")
(command "layer" "n" "text" "c" "green" "text" "")
(command "layer" "n" "center" "c" "yellow" "center" "lt" "center" "center" "")
(command "layer" "n" "hidden" "c" "magenta" "hidden" "lt" "hidden" "hidden" "")
(command "layer" "n" "section" "c" "cyan" "section" "lt" "phantom" "section" "")

;;; math

(setq x1 (car sp)
      x2 (+ x1 (/ BW 2.0))
      x3 (+ x2 BW)
      x4 (+ x1 FW)
      x5 (- x3 1.5)
      x6 (- x3 sillplate)
      x7 (- x3 24.0)
      x8 (- x3 wallthk)
)

(setq y1 (cadr sp)
      y2 (+ y1 fh)
      y3 (+ y2 0.375)
      y4 (+ y2 8.0)
      y5 (+ y2 (* courses 8.0))
      y6 (+ y5 1.5)
      y7 (+ y6 joist)
      y8 (+ y7 floor)
)

```

```

    y9 (+ y8 1.5)
    y12 (+ y8 wallht)
    y11 (- y12 1.5)
    y10 (- y11 1.5)
)

; point assignments

(setq p1 (list x1 y1 )
      p2 (list x4 y1 )
      p3 (list x4 y2 )
      p4 (list x1 y2 )
      p5 (list x2 y3 )
      p6 (list x3 y3 )
      p7 (list x3 y4 )
      p8 (list x2 y4 )
      p9 (list x2 y2 )
      p10 (list x3 y2 )
      p11 (list x6 y5 )
      p12 (list x3 y5 )
      p13 (list x3 y6 )
      p14 (list x6 y6 )
      p15 (list x5 y6 )
      p16 (list x3 y7 )
      p17 (list x5 y7 )
      p18 (list x3 y8 )
      p19 (list x7 y7 )
      p20 (list x7 y8 )
      p21 (list x8 y8 )
      p22 (list x8 y9 )
      p23 (list x3 y9 )
      p24 (list x8 y10 )
      p25 (list x3 y10 )
      p26 (list x8 y11 )
      p27 (list x3 y11 )
      p28 (list x8 y12 )
      p29 (list x3 y12 )
)

; let draw

(command "layer" "s" "footer" "" ) ; sets layers to the footer layer
(command "line" p1 p2 p3 p4 "c" ) ; draw the footer
(command "layer" "s" "block" "" ) ; sets layers to the block layer
(command "pline" p5 p6 p7 p8 "c") ; draws pline 1st block
(setq ss1 (ssget "L")) ; select the block
(command "arc" p9 "en" p5 "r" 0.1875) ; draws mortar joint arc
(setq ss2 (ssget "L")) ; select the mortar joint arc
(command "arc" p6 "en" p10 "r" 0.1875) ; draws mortar joint arc
(setq ss3 (ssget "L")) ; select the mortar joint arc
(command "array" ss1 ss2 ss3 "" "r" courses 1 8 1) ; arrays the courses of block
(command "layer" "s" "wood" "" ) ; sets layers to the wood layer
(command "pline" p11 p12 p13 p14 "c") ; draws the sillplate
(command "line" p12 p14 "" ) ; draws the sillplate
(command "line" p13 p11 "" ) ; draws the sillplate
(command "pline" p13 p16 p17 p15 "c") ; draws the joist
(command "line" p15 p16 "" ) ; draws the joist
(command "line" p13 p17 "" ) ; draws the joist
(command "pline" p16 p19 p20 p18 "c") ; draws the floor
(command "pline" p18 p21 p22 p23 "c") ; draws the sole plate
(command "line" p18 p22 "" ) ; draws the sole plate
(command "line" p23 p21 "" ) ; draws the sole plate
(command "pline" p22 p23 p25 p24 "c") ; draws the stud
(command "pline" p24 p25 p27 p26 "c") ; draws the first double top plate
(command "line" p26 p25 "" ) ; draws the first double top plate
(command "line" p24 p27 "" ) ; draws the first double top plate
(command "pline" p26 p27 p29 p28 "c") ; draws the second double top plate

```

```
(command "line" p28 p27 "") ; draws the second double top plate
(command "line" p29 p26 "") ; draws the second double top plate

;;; end of program

(command "layer" "s" "0" "")
(command "zoom" "e")
(setvar "osmode" osm)
(gc)
(princ)
)
```