

## ***Controlling Data within the System***

We have looked so far at how the interface may be used to display and capture user data. We have also looked at how the RAM may be used to store data temporarily so that it may be processed. We have also seen how the assignment operator may be used to copy data from one part of the system to another.

In this lecture, we will look at some more programming concepts and also introduce the use of the debugger.

- Assignment
- Sequence
- Selection
- Repetition
- Proceduralisation
- Parameters
- Return Values

### ***Assignment***

Assignment is carried out by the assignment operator

=

The assignment operator copies data from right to left.

←  
Destination = Source

Wouldn't it be nice though if you could see this process taking place?

Well you can.

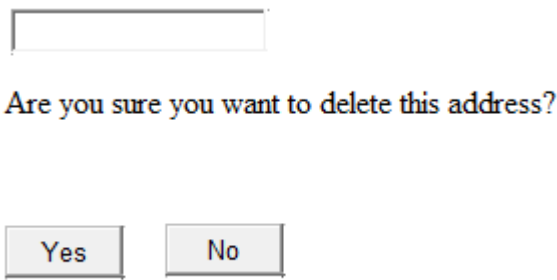
We are going to use a tool available in all programming languages called the debugger.

### ***Using the Debugger – Adding Breakpoints***

The debugger allows us to add what are called breakpoints to the code.

In this example, we shall use the delete code you have been working on in the lab.

Open the web form Delete.aspx in design view.

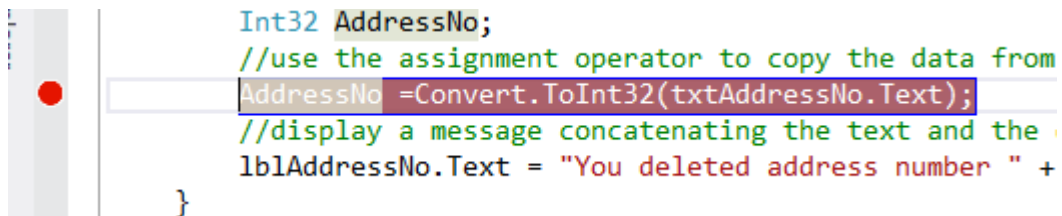


The next step is to double click the Yes button so that you can see the event handler code.

```
protected void btnYes_Click(object sender, EventArgs e)
{
    //this line of code re-directs back to the main page
    //Response.Redirect("Default.aspx");

    //declare a variable in ram to store the data from the interface
    Int32 AddressNo;
    //use the assignment operator to copy the data from the interface to the ram converting the data type
    AddressNo =Convert.ToInt32(txtAddressNo.Text);
    //display a message concatenating the text and the data in the variable
    lblAddressNo.Text = "You deleted address number " + AddressNo;
}
```

To add a break point, place the cursor on a line of code and press F9. The line should be highlighted in red.



```
Int32 AddressNo;
//use the assignment operator to copy the data from
AddressNo =Convert.ToInt32(txtAddressNo.Text);
//display a message concatenating the text and the
lblAddressNo.Text = "You deleted address number " +
}
```

To turn the break-point off again just press F9 a second time.

The next step is to see this working.

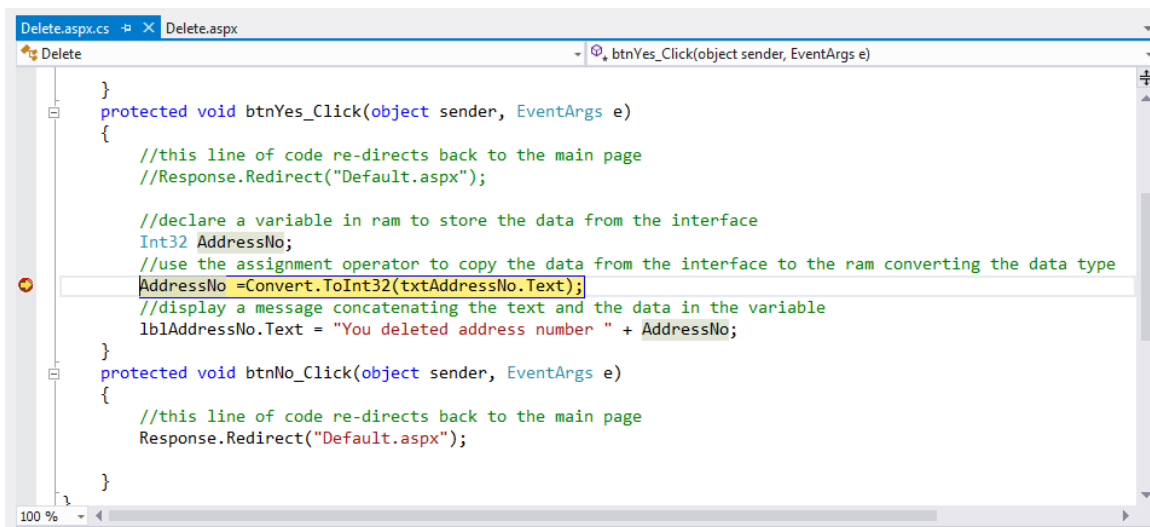
Run the program F5 and enter data as normal.

Are you sure you want to delete this address?

Yes

No

When you press the Yes button, the click event is triggered but this time the program “breaks” or pauses at the break point...



```
protected void btnYes_Click(object sender, EventArgs e)
{
    //this line of code re-directs back to the main page
    //Response.Redirect("Default.aspx");

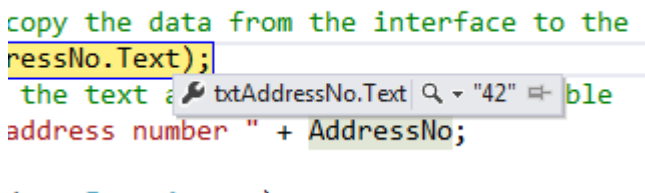
    //declare a variable in ram to store the data from the interface
    Int32 AddressNo;
    //use the assignment operator to copy the data from the interface to the ram converting the data type
    AddressNo = Convert.ToInt32(txtAddressNo.Text);
    //display a message concatenating the text and the data in the variable
    lblAddressNo.Text = "You deleted address number " + AddressNo;
}

protected void btnNo_Click(object sender, EventArgs e)
{
    //this line of code re-directs back to the main page
    Response.Redirect("Default.aspx");
}
```

This allows us to do a few things.

Firstly, we may inspect the data in different parts of the system.

If we hold the mouse over the Text property of the text box txtAddressNo we will see what data it contains...



```
copy the data from the interface to the
AddressNo.Text);
the text of txtAddressNo.Text is "42"
address number " + AddressNo;
```

Notice that the variable contains zero...

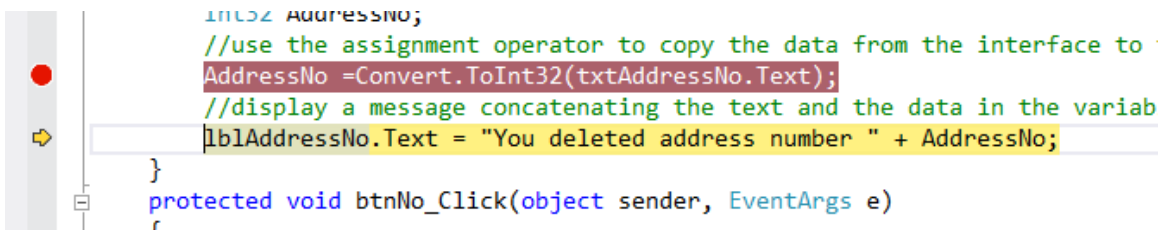
```

//declare a variable in
Int32 AddressNo;
//use the assignment op
AddressNo =Convert.ToInt
//disp AddressNo 0
lblAddressNo.Text = "You
}

```

The next thing we can do is step over each line of code a line at a time.

Press F10 and the code will move onto the next line...



```

Int32 AddressNo;
//use the assignment operator to copy the data from the interface to
AddressNo =Convert.ToInt32(txtAddressNo.Text);
//display a message concatenating the text and the data in the variab
lblAddressNo.Text = "You deleted address number " + AddressNo;
}
protected void btnNo_Click(object sender, EventArgs e)
{
}

```

Notice now that the data in the variable AddressNo has changed to the data from the text box...

```

Int32 AddressNo;
//use the assignment op
AddressNo =Convert.ToInt
//disp AddressNo 42
lblAddressNo.Text = "You

```

What we have seen here is the assignment operator doing its job. It has copied the data stored in txtAddressNo.Text to the RAM at location called AddressNo.

To set the program running out of break mode press F5.

## Sequence

Sequence is the normal order the programming language processes lines of code. It processes line one then moves onto line two and so on. Comments are ignored and lines are not missed until we break the rules of sequence.

Notice the lines of code in this function...

```

1  protected void btnYes_Click(object sender, EventArgs e)
    {
        //this line of code re-directs back to the main page
        //Response.Redirect("Default.aspx");

        //declare a variable in ram to store the data from the interface
        Int32 AddressNo;
        //use the assignment operator to copy the data from the interface to the ram converting the data type
        AddressNo = Convert.ToInt32(txtAddressNo.Text);
        //display a message concatenating the text and the data in the variable
4  lblAddressNo.Text = "You deleted address number " + AddressNo;
    }

```

The code runs from the first line to the last in sequence from the first line, the function definition to the last line.

The code doesn't go backward or skip lines unless we deliberately break the rules of sequence.

## ***Selection***

One way that we can break the rules of sequence is through selection. Selection is achieved by use of If statements and they involve making a choice based on some condition.

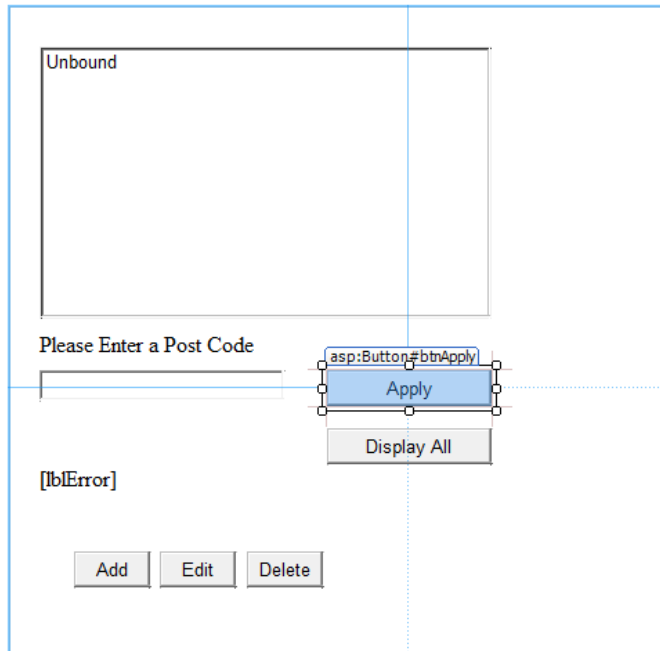
If (certain condition applies)

```

{
    //do this
}
Else
{
    //do this
}

```

An example of selection may be seen in the main page for the finished address book Default.aspx.



Access the event handler for the Delete button and insert a breakpoint like so...

```
//event handler for the delete button
protected void btnDelete_Click(object sender, EventArgs e)
{
    //var to store the primary key value of the record to be deleted
    Int32 AddressNo;
    //if a record has been selected from the list
    if (lstAddresses.SelectedIndex != -1)
    {
        //get the primary key value of the record to delete
        AddressNo = Convert.ToInt32(lstAddresses.SelectedValue);
        //store the data in the session object
        Session["AddressNo"] = AddressNo;
        //redirect to the delete page
        Response.Redirect("Delete.aspx");
    }
    else //if no record has been selected
    {
        //display an error
        lblError.Text = "Please select a record to delete from the list";
    }
}
```

Next run the program and press Delete without first selecting an item off the list box...

1 Some Street LE1 1BE  
22 The Road N19 6EF  
33 High Street LE1 6FG  
22 The Road N19 6EF

Do not click here first!

Please Enter a Post Code

Apply

Display All

4 records in the database

Add

Edit

Delete

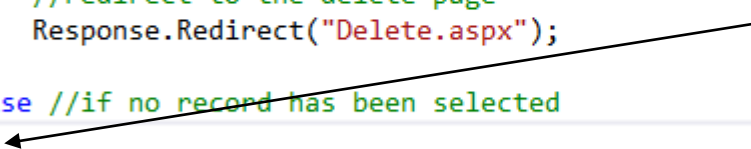
When you press delete the first thing that happens is that the program enters break mode...

```
//if a record has been selected from the list
if (lstAddresses.SelectedIndex != -1)
{
    //get the primary key value of the record to delete
    AddressNo = Convert.ToInt32(lstAddresses.SelectedValue);
    //store the data in the session object
    Session["AddressNo"] = AddressNo;
    //redirect to the delete page
    Response.Redirect("Delete.aspx");
}
else //if no record has been selected
{
    //display an error
    lblError.Text = "Please select a record to delete from the list";
}
```

What this If statement is checking is if the user has made a selection on the list box.

In this case, they haven't so watch what happens when you press F10...

```
//if a record has been selected from the list
if (lstAddresses.SelectedIndex != -1)
{
    //get the primary key value of the record to delete
    AddressNo = Convert.ToInt32(lstAddresses.SelectedValue);
    //store the data in the session object
    Session["AddressNo"] = AddressNo;
    //redirect to the delete page
    Response.Redirect("Delete.aspx");
}
else //if no record has been selected
{
    //display an error
    lblError.Text = "Please select a record to delete from the list";
}
```



The rules of sequence are broken and the program jumps over this code...

```
//get the primary key value of the record to delete
AddressNo = Convert.ToInt32(lstAddresses.SelectedValue);
//store the data in the session object
Session["AddressNo"] = AddressNo;
//redirect to the delete page
Response.Redirect("Delete.aspx");
```

The if statement only executes this code...

```
//display an error
lblError.Text = "Please select a record to delete from the list";
}
```

Press F5 to see the result.

This time make sure that you have selected an item from the list box and now press delete.



1 Some Street LE1 1BE
22 The Road N19 6EF
33 High Street LE1 6FG
22 The Road N19 6EF

What happens?

This time because the list box HAS been selected it runs the top section of code ignoring the bottom one...

```
//if a record has been selected from the list
if (lstAddresses.SelectedIndex != -1)
{
    //get the primary key value of the record to delete
    AddressNo = Convert.ToInt32(lstAddresses.SelectedValue);
    //store the data in the session object
    Session["AddressNo"] = AddressNo;
    //redirect to the delete page
    Response.Redirect("Delete.aspx");
}
else //if no record has been selected
{
    //display an error
    lblError.Text = "Please select a record to delete from the list";
}
```

## ***Repetition***

The next way that the rules of sequence may be broken is by repetition. Repetition uses loops to repeat sections of code based on a set of conditions.

```
While (This there is still stuff to process)
{
    //keep repeating the code here
}
```

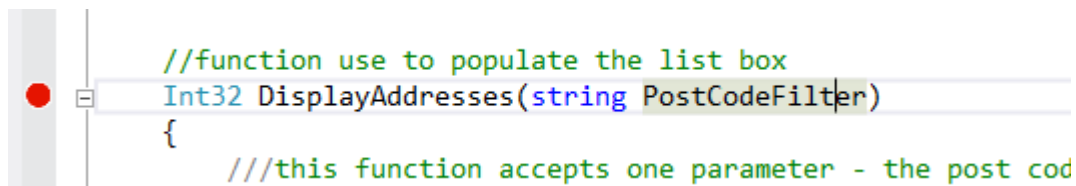
An example of a loop is found in the web form Default.aspx.

Locate the function called DisplayAddresses

```
//function use to populate the list box
Int32 DisplayAddresses(string PostCodeFilter)
{
    ///this function accepts one parameter - the post code to filter the list on
    ///it populates the list box with data from the middle layer class
    ///it returns a single value, the number of records found

    //create a new instance of the clsAddress
    clsAddress MyAddresses = new clsAddress();
    //var to store the count of records
    Int32 RecordCount;
    //var to store the house no
    string HouseNo;
    //var to store the street name
    string Street;
    //var to store the post code
    string PostCode;
    //var to store the primary key value
    string AddressNo;
    //var to store the index
    Int32 Index = 0;
    //clear the list of any existing items
    lstAddresses.Items.Clear();
    //call the filter by post code method
    MyAddresses.FilterByPostCode(PostCodeFilter);
    //get the count of records found
    RecordCount = MyAddresses.Count;
}
```

Place a breakpoint on the first line of the function and run the application.



The function is set up so that it runs every time the page loads.

It should break like so...

```

//function use to populate the list box
Int32 DisplayAddresses(string PostCodeFilter)
{
    ///this function accepts one parameter - the post code to filter the list on
    ///it populates the list box with data from the middle layer class
    ///it returns a single value, the number of records found

    //create a new instance of the clsAddress
    clsAddress MyAddresses = new clsAddress();
    //var to store the count of records
    Int32 RecordCount;
    //var to store the house no
    string HouseNo;
    //var to store the street name
    string Street;

```

Press F10 until you get to this line of code...

```

MyAddresses.FilterByPostCode(PostCodeFilter);
//get the count of records found
RecordCount = MyAddresses.Count;
//loop through each record found using the index
while (Index < RecordCount)

```

What do you suppose this line does?

MyAddress.Count contains the count of address in the table, in this case 4. The assignment operator will copy this number four to the variable RecordCount...

```

RecordCount = MyAddresses.Count;
//loop through each record found using the index
while (Index < RecordCount)

```

Notice the following section of code...

```

//loop through each record found using the index to point to each record in the data table
while (Index < RecordCount)
{
    //get the house no from the query results
    HouseNo = Convert.ToString(MyAddresses.QueryResults.Rows[Index]["HouseNo"]);
    //get the street from the query results
    Street = Convert.ToString(MyAddresses.QueryResults.Rows[Index]["Street"]);
    //get the post code from the query results
    PostCode = Convert.ToString(MyAddresses.QueryResults.Rows[Index]["PostCode"]);
    //get the address no from the query results
    AddressNo = Convert.ToString(MyAddresses.QueryResults.Rows[Index]["AddressNo"]);
    //set up a new object of class list item
    ListItem NewItem = new ListItem(HouseNo + " " + Street + " " + PostCode, AddressNo);
    //add the new item to the list
    lstAddresses.Items.Add(NewItem);
    //increment the index
    Index++;
}

```

Watch how it behaves as you keep pressing F10.

It gets to the line after `Index++` and then it jumps back to the top. It will do this four times...

```
//loop through each record found using the index to point to each record in the data table
while (Index < RecordCount)
{
    //get the house no from the query results
    HouseNo =Convert.ToString(MyAddresses.QueryResults.Rows[Index]["HouseNo"]);
    //get the street from the query results
    Street =Convert.ToString(MyAddresses.QueryResults.Rows[Index]["Street"]);
    //get the post code from the query results
    PostCode = Convert.ToString(MyAddresses.QueryResults.Rows[Index]["PostCode"]);
    //get the address no from the query results
    AddressNo =Convert.ToString(MyAddresses.QueryResults.Rows[Index]["AddressNo"]);
    //set up a new object of class list item
    ListItem NewItem = new ListItem(HouseNo + " " + Street + " " + PostCode, AddressNo);
    //add the new item to the list
    lstAddresses.Items.Add(NewItem);
    //increment the index
    Index++;
}
```

What the loop is doing is it is processing the data for the four records in the database.

## ***Proceduralisation***

Proceduralisation is just a long word to describe making functions.

As we have said computer programs are made of functions and a function is a set of instructions that performs a single operation on the data well.

To see a function in action we shall add a breakpoint to the click event of the Apply button on Default.aspx.

```
//event handler for the apply button
protected void btnApply_Click(object sender, EventArgs e)
{
    //declare var to store the record count
    Int32 RecordCount;
    //assign the results of the DisplayAddresses function to the record count var
    RecordCount = DisplayAddresses(txtPostCode.Text);
    //display the number of records found
    lblError.Text = RecordCount + " records found";
}
```

Run the program and press Apply.

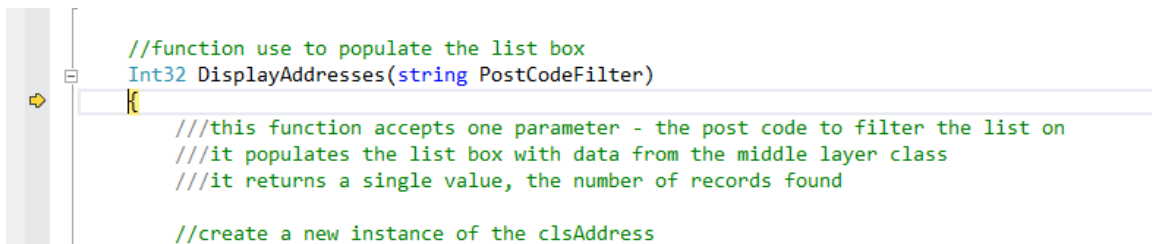
Do this first using F10 the code should just run in sequence from first line to last.

Now try it again but when the following line of code is selected press F11...

```
RecordCount = DisplayAddresses(txtPostCode.Text);
```

Note what happens.

The execution of the code jumps to the function.

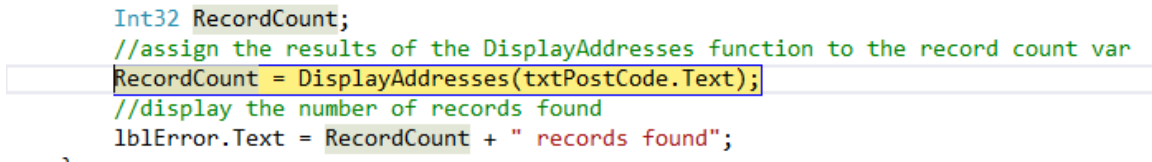


```
//function use to populate the list box
Int32 DisplayAddresses(string PostCodeFilter)
{
    ///this function accepts one parameter - the post code to filter the list on
    ///it populates the list box with data from the middle layer class
    ///it returns a single value, the number of records found

    //create a new instance of the clsAddress
```

If you keep pressing F10 until the function has finished note what happens.

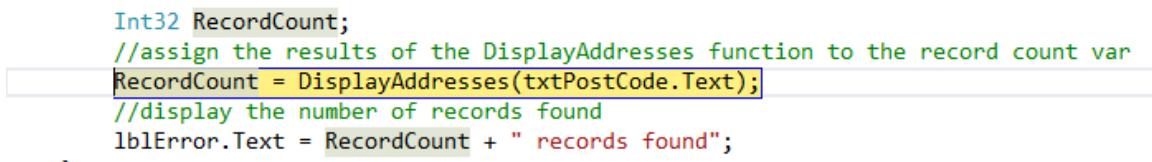
The code jumps back to the where the function call was made...



```
Int32 RecordCount;
//assign the results of the DisplayAddresses function to the record count var
RecordCount = DisplayAddresses(txtPostCode.Text);
//display the number of records found
lblError.Text = RecordCount + " records found";
```

## **Parameters**

Notice also in this code the section in brackets...



```
Int32 RecordCount;
//assign the results of the DisplayAddresses function to the record count var
RecordCount = DisplayAddresses(txtPostCode.Text);
//display the number of records found
lblError.Text = RecordCount + " records found";
```

The brackets contain the parameters for the function.

The presence of brackets is the way we can tell if something is a function. It may not have parameters but it will have brackets.

This time enter some text into the filter text box and press Apply.

Please Enter a Post Code

LE

Apply

Display All

4 records in the database

When the code breaks inspect the value passed as a parameter...

```
DisplayAddresses function to the record co  
ses(txtPostCode.Text);  
ords found  
+ " records found".
```

txtPostCode.Text Q "LE"

Now press F11 to step into the function and inspect the parameter in the function definition...

```
//function use to populate the list box  
Int32 DisplayAddresses(string PostCodeFilter)  
{  
    ///this function accepts one parameter - the post code to fi  
    ///it populates the list box with data from the middle laver
```

PostCodeFilter Q "LE"

Parameters give us a mechanism for copying data from one function to another.

## Return Values

The next thing is to press F10 on the function until we get to the last line of code...

```
        Index++;  
    }  
    ///return the number of records found  
    return RecordCount;  
}
```

RecordCount

Inspect the data in the variable RecordCount and make a note of it. (In this example, it is 2 the number of records found by the function.)

Press F10 until you have returned to the code that called the function.

```
Int32 RecordCount;  
//assign the results of the DisplayAddresses funct  
RecordCount = DisplayAddresses(txtPostCode.Text);  
//display RecordCount 2 records found  
lblError.Text = RecordCount + " records found";  
}
```

Notice that the value 2 has been passed back as the return value of the function and assigned to the variable RecordCount.

Return values are a mechanism allowing functions to “report back” with some data value.

Over the rest of the module we shall explore each of these concepts in greater depth.