Last weeks program contains some first syntax apart from procedure and data. As part of the data structure, we used an *array* to hold the three components of the three dimensional coordinates here and there.
To propagate the segments of the line sequentially, we used the *loop* control statement. Hence on this hand-out we will introduce the one-dimensional array and the two control statements *For - Next Loop* and *If - Else*.

Arrays (one dimensional)

You might have asked yourself what kind of construct the

```
Dim here(2) As Double
Dim there(2) As Double
```

represents?

In Basic, just like in any other programming language, one can dimension any data type to be a set of sequentially indexed elements - an *array*. That means the elements occur in a hierarchical order from, say 0 to 2 in the points example of our line_walk code. *there(0)* is the first element that represent the X-axis component of the point, *there(1)* the Y-axis component and *there(2)* the Z-axis component.
Notice that all elements will always be of the data type the array was dimensioned to. If you want to send the complete array to another sub procedure or function you just call the the sub procedure or function with the array name as an argument:

```
random there,0, 25
```

The sub procedure or function that receives the array as an argument will have to declare it then with brackets again:

```
Sub random(endpt() as double, low As Double, top As double)
```

Thus, you will have full access to all elements of the array. Remember that the name of the argument can change when received!

You can also just send one element of an array as an argument to a sub procedure or function:

```
random there(2), 0, 25
```

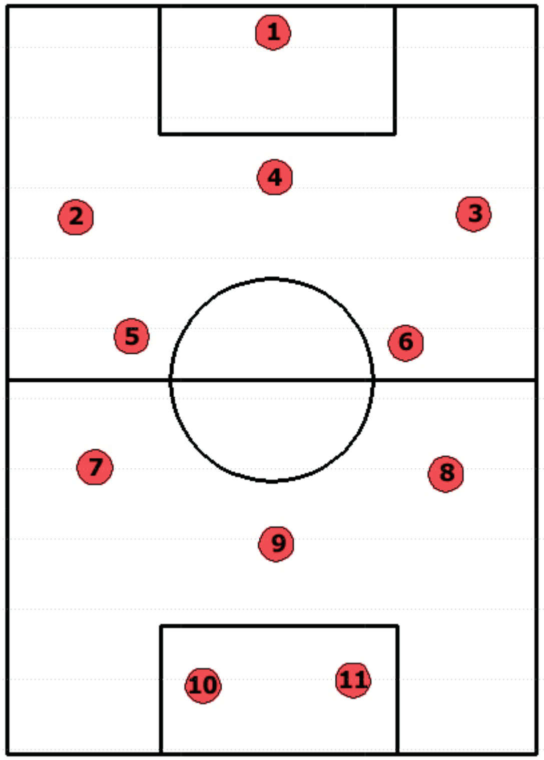
In that case the receiving sub procedure or function just declares a single variable with its data type laid out in the dimensioning of the array:

```
Sub random(height As Double, low As Double, top as double)
```

Again the name of the argument/ variable has been changed.
You can access the elements of the array in any index order you like. In other words it doesn't have to be sequentially or from the start. And an array can have as many elements as you like as long as there is enough memory. Just make sure that you never try to access an element index that doesn't exist, *like there(3)*. That would give you a Run-time error *(Out of range!)*.
It returns an error that is discovered by the compiler only when the program runs, because syntactically the code is correct. When a syntax error occurs, the compiler will tell you already when compiling. The error with *there(3)* occurs at run-time because the point *there* is dimensioned from 0 to 2 *(2)*, not *(3)*.

CONTROL STRUCTURES

As we have already mentioned on the last hand-out, a procedural program has a hierarchical structure with the main procedure, outsourced sub-procedures that handle tasks and functions that serve for specific calculations. Control statements help to determine the sequence of execution based on conditions.



A football team and its positions can be demonstrated to be an array:

```
Dim Team(1 to 11) as String

Team(1)  =  "Coates"
Team(2)  =  "Thum"
Team(3)  =  "Carranza"
Team(4)  =  "Derix"
Team(5)  =  ""
...
Team(11)  ="Rich"
```

where the positions are filled in by the players' names. Next week's game can see some changes in players for the same positions:

```
Team(2)  =  "Derix"
Team(4)  =  "Insull"
```

As in real arrays, indices won't change but their values can.

Control structures come in two flavours:

- *looping statements*
- *conditional statements*

Loops

Loops are a mechanism to repeat blocks of code as in the walk_line program:

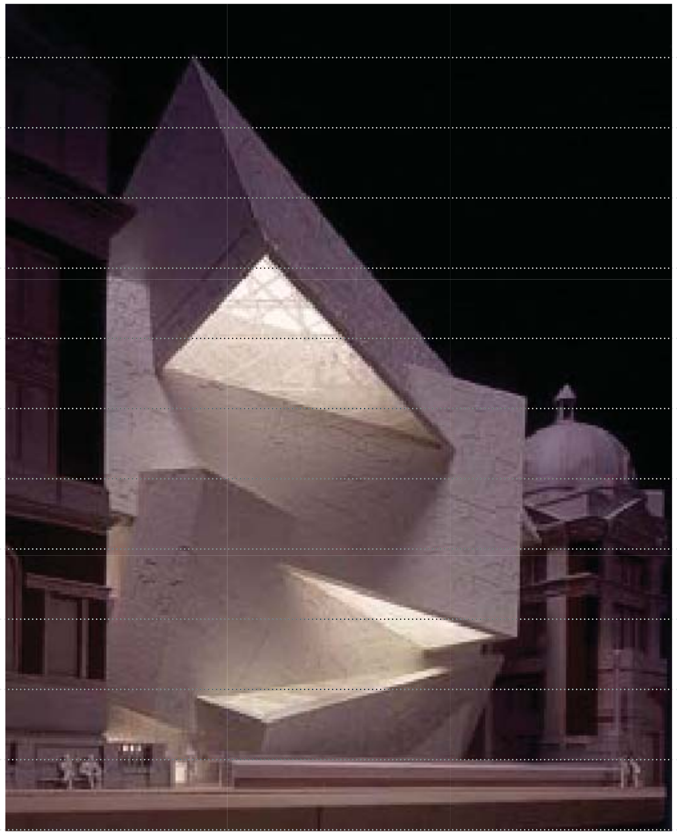
```
For i = 0 To 50
        here = there
        there = random()
        Set walker = ...AddLine(here, there)
Next i
```

There are different types of loops (*Do... Loop While(), While()...Wendt*) of which we will concentrate on the enumerative one, in other words the one like in our program where you repeat something a set number of times (here 50 times).
You always need a *counter* variable! Most often the *counter* variable is called *i*, standing for the *i(ndex)* of the loop. This variable is mostly of data type integer. This means that you should dimension the counter variable before you want to start a loop. The loop index or counter is increased (or decreased) between a lower and an upper limit - 0 and 50 in this code. The increase (or decrease) occurs linearly at the *Next i* statement. The upper limit indicates how many times to go around the loop (here 50 times).

The counter variable will be increased by 1 or decreased by 1 depending on how you set the limits for the loop. To make it very explicit one can also describe the increase/ decrease as

i = i +/- 1

(so if i = 0, then 0 = 0 + 1, making i = 1)



Victoria & Albert Museum Extension by Daniel Liebes-kind (simplified algorithm):

```
        For i = 0 to 3

                center.z = center.z + random(5,10)
                add.box center
                box.rotate3D

        Next i
```

Remember the Tofu...

which stands for the *Next i* statement.
The counter variable i will increase or de-crease by 1 from the lower limit you set the loop out to the upper limit. Once the counter variable has reached the upper limit then it will stop executing the loop and con-tinue below the loop with the rest of the code.
You can also jump the indices of the loop coun-ter by adding a *Step* keyword at the end of the upper limit definition:

```
For i = 0 to 10 Step 2
...
Next i
```

will count only 0, 2, 4, 6, 8, 10.

A loop can be exited before the counter vari-able reaches it upper limit with an *Exit For* statement, which is generally placed in a con-ditional statement.

Conditional Statements

We already saw conditional statements in Net-Logo. There they were simple called *If...[]* or *IfElse...[][]*.
In VBA, as many other language, the condition-al statement is more explicit and can have two different types of which we will look at the more basic one today:

```
If(expression) Then statement execution
```

One could say: 'If the weather is fine, I will go to the sea.' Written in VBA that would be:

```
If (weather = 1) Then gotosea
```

or

```
If (weather = 1) Then
        gotosea
End If
```

If...Then or *If...Then...End If* is the fixed syntax by VBA. (weather = 1) is the *expression* evaluated for executing the statement following the consequential keyword *then*. The '=' is the comparison operator. There can be any comparison operator depending on what you want to express. 'gotosea' is just the code or statement that will be executed if the *expression* evaluates to being TRUE. Conditional statements can only evaluate to TRUE of FALSE.
If the statement to be executed, like gotosea, is only one line long, the whole conditional statement can be written on one line as in the first example. The second example shows how it would be written are more statements to be executed.

For example, if one would say: 'If the weather is fine, I will go to the sea and eat and gamble.' then the code in VBA would look like that:

```
If (weather = 1) Then
        gotosea
        gamble
End if
```

If one would say: 'If the weather is fine, I will go to the sea and gamble otherwise I will watch tele.' then the code gets an 'alternative keyword' *Else* added:

```
If (weather = 1) Then
        gotosea
        gamble
Else
        watchTele
End If
```

One can also check a second *expression* in case the first didn't evaluate to be true. In that case an *ElseIf(condition)* statement is added.
Say:' If the weather is fine I will go to the sea and gamble. If the weather is ok I will go shopping. Otherwise I will just watch tele.'

```
If (weather = 1) Then
        gotosea
        gamble
ElseIf (weather = 2) Then
        shopping
Else
        watchTele
End If
```

Now the *expression* can evaluate to three different cases:
fine(1), ok(2) or anything else apart from 1 or 2.

One can also use the ***IIf*** function to assign a value to a variable evaluated through a condition. Instead of writing:

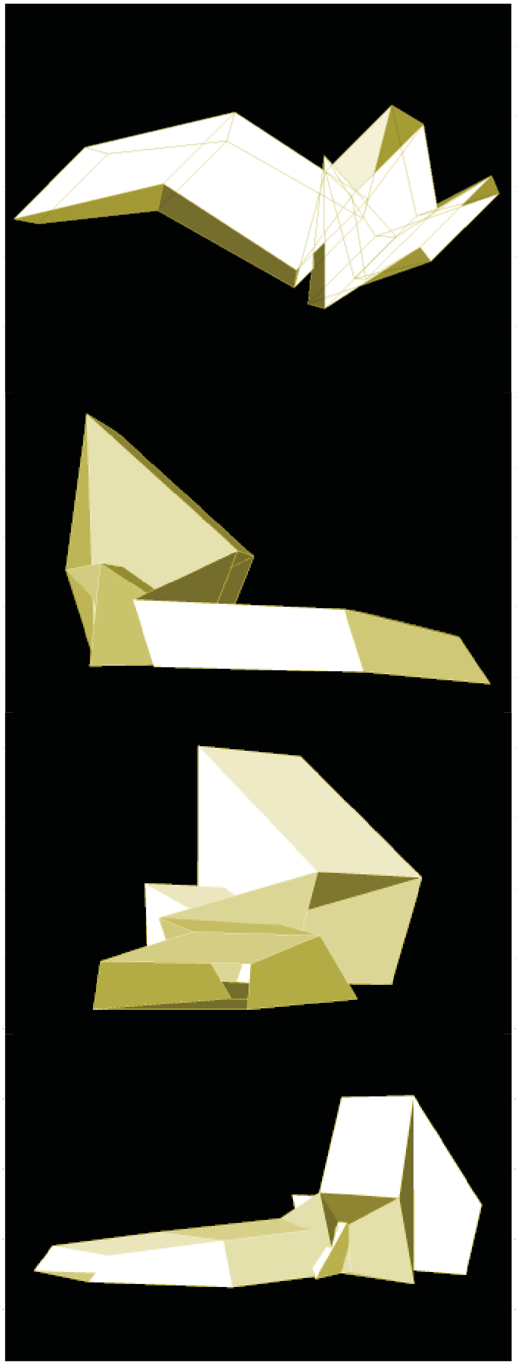
```
If (weather = 1) Then
        gotosea
Else
        watchTele
End If
```

```
what_to_do = IIf(weather = 1, gotosea, watchTele)
```

'what_to_do' is the *return variable* that gets either the *true-part* value or the *false-part* value, depending on the evaluation of the expression. The return variable needs to be dimensioned first.

```
variable = IIf(expression, true-part, false-part)
```

Later we will also take a look at multiple evaluation statements, called *Select Case*.



Computational Sketching

Today, we will experiment with 'live' writing algorithms that will change the quality of the line walk.

Intially, we will change the way the points of the line can be arranged.

Then, by projecting a clone of the first line outwards, a series of planes can be derived that start to look like architectural features. Thus, from a one-dimensional description a two dimensional representation can be arrived at.

Task

Everybody must experiment with other projections of the lines into a volumetric representation.

Keep it simple and try to use variations of the first lines only.

left: an experiment by a Viennese student to use lines and surfaces only to generate voluments composition