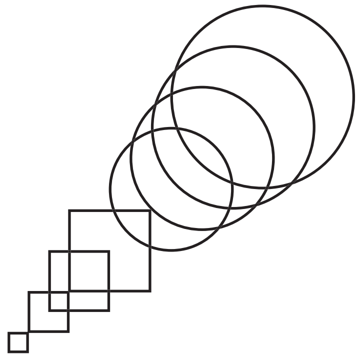
Conditional Statements

The IF Statement

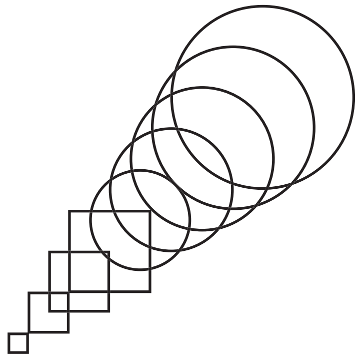
• IF ... THEN ...

```
PROCEDURE theIf;
VAR
        i : INTEGER;
BEGIN
        FOR i:=0 TO 9 DO BEGIN
                IF (i<5) THEN Rect(i,i,i*2,i*2);
                IF (i>5) THEN Oval(i,i,i*2,i*2);
        END;
END;
RUN(theIf);
```



• IF ... THEN ... ELSE..

```
PROCEDURE theIf;
VAR
        i : INTEGER;
BEGIN
        FOR i:=0 TO 9 DO BEGIN
                IF (i<5) THEN Rect(i,i,i*2,i*2)
                ELSE Oval(i,i,i*2,i*2);
        END;
END;
RUN(theIf);
```

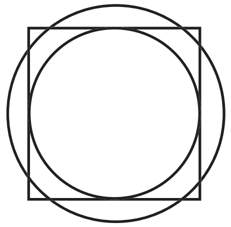


Conditional Statements

The CASE Statement

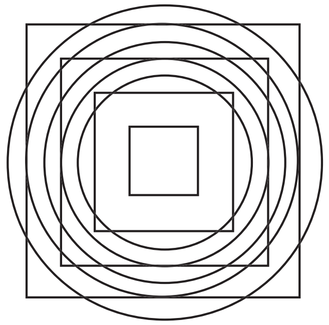
- **CASE ... OF ... END;**

```
PROCEDURE theIf;
VAR
        i : INTEGER;
BEGIN
        FOR i:=0 TO 9 DO BEGIN
                CASE i OF
                        2: Rect(-i,-i,i,i);
                        4,5: Oval(-i/2,-i/2,i/2,i/2);
                END;
        END;
END;
RUN(theIf);
```



- **CASE ... OF ... [OTHERWISE] ... END;**

```
PROCEDURE theIf;
VAR
        i : INTEGER;
BEGIN
        FOR i:=0 TO 9 DO BEGIN
                CASE i OF
                        0..4: Rect(-i,-i,i,i);
                        OTHERWISE Oval(-i/2,-i/2,i/2,i/2);
                END;
        END;
END;
RUN(theIf);
```



Arrays in VectorScript

• An **array** in VectorScript is a collection of data values referenced by a **single identifier**. Arrays allow **large amounts of data** to be **stored and manipulated** during script execution.

• VectorScript arrays are **indexed.**

• VectorScript provides support for two types of arrays:
static arrays (ARRAY), and **dynamic arrays** (DYNARRAY).

  • Static Array
  • Dynamic Array

Arrays in VectorScript

Static Array

• Static arrays (ARRAY) are declared using the **same method as** used for **variables**

• Static arrays come in **one-** and **two-dimensional** varieties.
The general syntax for one-dimensional static arrays is:

```
<identifier> : ARRAY [ m..n ] OF <data type>;
e.g. myArray : Array[0..23] OF INTEGER;
```

• To retrieve a value from an element of a one-dimensional array, the bracket notation has to be used, e.g.

```
j := values[3];
values[23] := 15.5;
total := price[i] + tax;
```

Arrays in VectorScript

Static Array

• example Script:

```
PROCEDURE ExampleArray;
VAR
        s:STRING;
        i:INTEGER;
        words:ARRAY[1..10] OF STRING;
BEGIN
        words[1]:='VectorScript ';
        words[2]:='is ';
        words[3]:='a ';
        words[4]:='fine ';
        words[5]:='language.';

        FOR i:=1 TO 5 DO s:=Concat(s,words[i]);
                Message(s);
        END;
END;
Run(ExampleArray);
```

Arrays in VectorScript

Static Array

• Two-dimensional static arrays **extend the syntax of a one-dimensional** array by adding an additional array index to the declaration:

```
<identifier> : ARRAY [ m..n,r..s ] OF <data type>;
```

• In the declaration for the two-dimensional array, **the first index** value defines the **number of "rows"** in the array, while **the second index** defines the **number of "columns."**

• Accessing an element in a two-dimensional array is not very different from a one-dimensional array:

```
j := values[3,5];
values[23,1] := 15.5;
total := price[i,j] + tax;
```

• If we think of the two-dimensional array in terms of **rows** and **columns**, we would use **two index values to indicate the row and column position** of the array element to be indexed.

Arrays in VectorScript

Dynamic Array

• Dynamic arrays (DYNARRAY) in VectorScript are **similar to static arrays**, with the notable exception of how they are dimensioned, or sized.

• While static arrays are explicitly sized when they are declared in the VAR block of your script, **the size of a dynamic array is declared during the actual execution of a script.**

• Dynamic arrays **can also be resized** at any point **during script execution** to suit your data storage requirements.

• Dynamic arrays can also be specified as **one- or two-dimensional**. The general syntax for dynamic arrays are:

- one-dimensional:
  <identifier> : DYNARRAY [] OF <data type>;

- two-dimensional:
  <identifier> : DYNARRAY [,] OF <data type>;

Arrays in VectorScript

Dynamic Array

• **To dimension** a dynamic array, VectorScript uses the **ALLOCATE** keyword (along with a reference to the array):

```
ALLOCATE int_values[1..5];
```

• Extended String Support with CHAR Arrays

  • VectorScript also supports a specialized set of functionality when using arrays of the CHAR data type.

  • Arrays of type CHAR can be used in place of the STRING data type in certain operations within VectorScript.

• for more Details to manipulating STRINGS and CHAR data type you can check the manual

Arrays in VectorScript

Dynamic Array

• example Script:

```
PROCEDURE Example_DynArray;
VAR
        i,j,numtxt : INTEGER;
        h : HANDLE;
        textStore: DYNARRAY[] OF STRING;
BEGIN
        numtxt:=Count(((T=Text) & (SEL=TRUE)));
        j:=1;
        ALLOCATE textStore[1..numtxt];
        h:=FSActLayer;
        WHILE (h <> NIL) DO BEGIN
                IF (GetType(h) = 10) THEN BEGIN
                        textStore[j]:=GetText(h);
                        j:=j+1;
                END;
                h:=NextSObj(h);
        END;
        ALLOCATE textStore[1..numtxt+2];
        TextOrigin(2,2);
        CreateText(‘New text 1’);
        numtxt:=numtxt+1;
        textStore[numtxt]:=GetText(LNewObj);
        TextOrigin(2,4);
        CreateText(‘New text 2’);
        numtxt:=numtxt+1;
        textStore[numtxt]:=GetText(LNewObj);
        FOR i:=1 TO numtxt DO BEGIN
                Message(‘Array element ‘,i,’ contains ‘, textStore[i]);
                Wait(1);
        END;
END;
Run(Example_DynArray);
```

Structures

• A structure in VectorScript is a **collection of one or more variables** which are grouped together under a single identifier for convenient handling.

• Structures help to **organize complex data into groupings** that may be treated as a single “unit” instead of separate entities.

• The general syntax for a structure declaration is:

```
<structure name> = STRUCTURE
<identifier>[,<identifier>,…] : <data type>;
<identifier>[,<identifier>,…] : <data type>;
```

• Members within a structure may be referred to directly using the **. (structure member) operator**.

```
<structure name>.<member name>
```

• This format, also known as “**dot notation**,” gives you direct access to the value within the specified member.

Structures

• Example:

```
PROCEDURE Example_structure;

TYPE
        HANSPETER = STRUCTURE
                vorname, nachname : STRING;
        END;

        ADRESSE = STRUCTURE
                strasse : STRING;
                hausnummer: INTEGER;
                stadt : STRING;
        END;

VAR
        person1: HANSPETER;
BEGIN
        person1.vorname:= 'Uschi';
        person1.nachname:='Biedermann';

        Message(person1.vorname, ' ' ,person1.nachname);
END;
Run(Example_structur);
```