

*Using Databases in Delphi***1. SET UP ADO TABLE**

- 1.1 Create database file in MS Access
- 1.2 In Delphi: choose ADOTable under the ADO menu on the component palette
- 1.3 Change settings in the Object Inspector: set the Connection String settings by clicking on the ellipse
- 1.4 Click on Build...
- 1.5 Choose: Microsoft Jet 4.0 OLE DB Provider
- 1.6 Click Next >>
- 1.7 Select database file (mdb file)
- 1.8 Erase user name ('Admin')
- 1.9 Make sure 'Blank Password' is selected
- 1.10 Click OK (on 'Data Link Properties' window)
- 1.11 Click OK (on 'ConnectionString' window)
- 1.12 Click on ADOTable component: choose Table name for ADOTable
- 1.13 Set ADOTable Active to TRUE
- 1.14 Add DataSource (link to table)
- 1.15 Add DBGrid and DBNavigator (found under 'Data Controls'; link to table)

**SET UP ADO CONNECTION****(To be used with Tables or Queries)**

- 12.1 Create database file in MS Access
- 12.2 In Delphi: choose ADOConnection under the ADO menu on the component palette
- 12.3 Change settings in the Object Inspector: set the Connection String settings by clicking on the ellipse
- 12.4 Click on Build...
- 12.5 Choose: Microsoft Jet 4.0 OLE DB Provider
- 12.6 Click Next >>
- 12.7 Select database file (mdb file)
- 12.8 Erase user name ('Admin')
- 12.9 Make sure 'Blank Password' is selected
- 12.10 Click OK (on 'Data Link Properties' window)
- 12.11 Click OK (on 'ConnectionString' window)
- 12.12 Click on the ADOConnection and go to the Object Inspector
- 12.13 Set Login Prompt to false
- 12.14 Set Connected to true
- 12.15 On ADOTable or ADOQuery choose this ADOConnection in the Connection property in the Object Inspector

**2. ADD RECORD TO TABLE USING CODING****2.1 A new record can be added using set data:**

```
procedure TForm1.Button1Click(Sender:
TObject);
begin
  ADOTable1.Append;
  ADOTable1['Name'] := 'John';
  ADOTable1['Surname'] := 'Doe';
  ADOTable1.Post;
end;
```

2.2 A new record can be added from an edit box:

```
procedure TForm1.Button1Click(Sender:
TObject);
begin
  ADOTable1.Append;
  ADOTable1['Name'] := Edit1.Text;
  ADOTable1['Surname'] := Edit2.Text;
  ADOTable1.Post;
  Edit1.Clear;
  Edit2.Clear;
end;
```

**2.3 Add a record from RadioGroup and SpinEdit:**

```
procedure TForm1.Button1Click(Sender:
TObject);
begin
  ADOTable1.Append;
  ADOTable1['Class'] :=
RadioGroup1.Items[RadioGroup1.ItemIndex];
  ADOTable1['Grade'] := SpinEdit1.Value;
  ADOTable1.Post;
end;
```

When using the RadioGroup, take note:

- Add the entries using the Items property
- The first item's index is 0
- RadioGroup1.ItemIndex refers to the selected item.

2.4 Add a record from a ComboBox

```
procedure TForm1.Button1Click(Sender:
TObject);
begin
  ADOTable1.Append;
  ADOTable1['Team'] :=
ComboBox1.Items[ComboBox1.ItemIndex];
  ADOTable1.Post;
end;
```

**OR (if the contents of the ComboBox should not be transferred as it is to the database)**

```
procedure TForm1.Button1Click(Sender:
TObject);
begin
  ADOTable1.Append;
  Case ComboBox1.ItemIndex of
    0 : ADOTable1['Grade'] := '10';
    1 : ADOTable1['Grade'] := '11';
    2 : ADOTable1['Grade'] := '12';
  end;
  ADOTable1.Post;
end;
```

2.5 Add a record from a DBLookupComboBox

- Set up DBLookupComboBox by choosing the DataSource to be used in ListSource
- Choose the field to be listed in ListField
- Choose a field as keyfield in KeyField
- When choosing a value the focus is placed on that particular record in the table used and any field can be used from that record, for example:

```
procedure TForm1.Button1Click(Sender:
TObject);
begin
  ADOTable1.Append;
  ADOTable1['Name'] := ADOTable2['Name'];
  ADOTable1.Post;
end;
```

**3. SEARCH FOR A RECORD IN A TABLE**

```
procedure TForm1.Button1Click(Sender:
TObject);
begin
  ADOTable1.First;
  while not ADOTable1.Eof do
  begin
    if ADOTable1['Name'] =
      Edit1.Text then
    begin
      ShowMessage('This record is
        present');
      Exit;
    end
    else
      ADOTable1.Next;
  end;
end;
```

4. SEARCH AND REPLACE FIELDS WITH SET DATA

```
procedure TForm1.Button2Click(Sender:
TObject);
begin
  ADOTable1.First;
  while not ADOTable1.Eof do
  begin
    if ADOTable1['Name'] = Edit1.Text then
    begin
      ADOTable1.Edit;
      ADOTable1['Name'] := Edit2.Text;
      end;
      ADOTable1.Next;
    end;
  end;
```

5. DO A CALCULATION IN A FIELD USING DATA FROM EXISTING FIELDS

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ADOTable1.First;
  While not ADOTable1.Eof do
  begin
    ADOTable1.Edit;
    ADOTable1['Total'] :=
      ADOTable1['Unit'] *
      ADOTable1['Amount'];
    ADOTable1.Next;
  end;
end;
```

6. DELETE RECORD MEETING SEARCH CRITERIA

```
procedure TForm1.Button1Click(Sender:
TObject);
begin
  ADOTable1.First;
  while not ADOTable1.Eof do
  begin
    if ADOTable1['Name'] = Edit1.Text then
    begin
      ADOTable1.Delete;
      Exit;
    end
    else
      ADOTable1.Next;
  end;
end;
```

7. DELETE RECORD CURRENTLY IN FOCUS**7.1 Click on record then click on Button1:**

```
procedure TForm1.Button3Click(Sender:
TObject);
begin
  ADOTable1.Delete;
end;
```

8. FILTER RESULTS**8.1 Filter according to exact search criteria (e.g. Name = 'John'):**

```
procedure TForm1.Button3Click(Sender:
TObject);
begin
  ADOTable1.Filter := 'Name = ' + ''' +
    Edit1.Text + ''';
  ADOTable1.Filtered := True;
end;
```

8.2 Filter according to similar search criteria (e.g. Name LIKE 'Jo%'):

```
procedure TForm1.Button3Click(Sender:
TObject);
begin
  ADOTable1.Filter := 'Name LIKE ' + ''' +
    Edit1.Text + '%''';
  ADOTable1.Filtered := True;
end;
```

8.3 Filter according to number value:

```
procedure TForm1.Button3Click(Sender:
TObject);
begin
  ADOTable1.Filter := 'Amount > ' +
    Edit1.Text;
  ADOTable1.Filtered := True;
end;
```

9. DELETE ALL RECORDS

```
procedure TForm1.Button3Click(Sender:
TObject);
begin
  ADOTable1.First;
  while not ADOTable1.Eof do
  begin
    if ADOTable1['Name'] <> '' then
      ADOTable1.Delete
    else
      ADOTable1.Next;
  end;
end;
```

10. COUNT RECORDS

```
procedure TForm1.Button3Click(Sender:
TObject);
var
  iCount : Integer;
begin
  iCount := 0;
  ADOTable1.First;
  while not ADOTable1.Eof do
  begin
    if ADOTable1['Name'] <> '' then
      iCount := iCount + 1;
    ADOTable1.Next;
  end;
  Label1.Caption := IntToStr(iCount);
end;
```

Wildcard: %

11. QUERIES



11.1 Set up query

- 11.1.1 Create database file in MS Access
- 11.1.2 In Delphi: choose ADOQuery under the ADO menu on the component palette
- 11.1.3 Change settings in Object Inspector: set Connection String settings by clicking on the ellipse
- 11.1.4 Click on Build...
- 11.1.5 Choose: Microsoft Jet 4.0 OLE DB Provider
- 11.1.6 Click Next >>
- 11.1.7 Select database file (mdb file)
- 11.1.8 Erase user name ('Admin')
- 11.1.9 Make sure 'Blank Password' is selected
- 11.1.10 Click OK (on 'Data Link Properties' window)
- 11.1.11 Click OK (on 'ConnectionString' window)
- 11.1.12 Set Query in Object Inspector
 - Click on ellipse next to (TStrings) for SQL property of the ADOQuery
 - Type in SQL code (see below)
 - Set Active property to True

11.2 Queries

11.2.1 Show all fields and all records

```
SELECT * FROM tblTableName;
```

11.2.2 Show certain field(s) and all records

```
SELECT Name, Surname FROM tblTableName;
```

11.2.3 Show all fields for records meeting certain exact criteria

```
SELECT * FROM tblTableName WHERE Name = "John";
```

11.2.4 Show all fields for records meeting similar criteria

```
SELECT * FROM tblTableName WHERE Name LIKE "Jo%";
```

11.2.5 Show all fields and all records sorted according to a field (ascending)

```
SELECT * FROM tblTableName ORDER BY Name;
```

11.2.6 Show all fields and all records sorted according to a field (descending)

```
SELECT * FROM tblTableName ORDER BY Name DESC;
```

11.2.7 Show all fields and all records within a set range (Unit is an Integer field)

```
SELECT * FROM tblTableName WHERE Unit BETWEEN 1 AND 6;
```

11.3 Set Query with coding

```
procedure TForm1.Button3Click(Sender:
TObject);
begin
    with ADOQuery1 do
    begin
        Active := false;
        SQL.Clear;
        SQL.Add('SELECT * FROM tblTableName;');
        Active := true;
    end;
end;
```

SQL coding added to 'SQL.Add' statement.

11.4 Change data (Amount field becomes 10) according to a condition (Unit equals to 4)

```
procedure TForm1.Button3Click(Sender:
TObject);
begin
    with ADOQuery1 do
    begin
        Active := false;
        SQL.Clear;
        SQL.Add('UPDATE tblTableName');
        SQL.Add('SET Amount=10 WHERE Unit=4');
        ExecSQL;
    end;
    ADOTable1.Refresh;
end;
```

11.5 Filter according to input from an edit box

```
procedure TForm1.Button3Click(Sender:
TObject);
begin
    with ADOQuery1 do
    begin
        SQL.Clear;
        SQL.Add('SELECT *');
        SQL.Add('FROM tblTableName');
        SQL.Add('WHERE NAME = ''' + Edit1.Text
        + ''');
        Open;
    end;
end;
```

11.6 Delete a record

```
procedure TForm1.Button3Click(Sender:
TObject);
begin
    with ADOQuery1 do
    begin
        Active := False;
        SQL.Clear;
        SQL.Add('DELETE FROM tblTableName
        WHERE Name = ''' + Edit1.Text + ''');
        ExecSQL;
    end;
end;
```

Remove 'WHERE' statement to delete all records.

11.7 Insert a record

```
procedure TForm1.Button3Click(Sender:
TObject);
begin
    with ADOQuery1 do
    begin
        Active := False;
        SQL.Clear;
        SQL.Add('INSERT INTO
        tblTableName(Name,Surname) VALUES (''
        + Edit1.Text + ''','' + Edit2.Text +
        ''')');
        ExecSQL;
    end;
end;
```

11.8 Create new field from calculation

Creates a new field 'AmountTax' created by adding 14% to the value in the Amount field

```
procedure TForm1.Button3Click(Sender:
TObject);
begin
    with ADOQuery1 do
    begin
        Active := false;
        SQL.Clear;
        SQL.Add('SELECT Name, Amount,
        [Amount]*1.14 AS AmountTax FROM
        tblTableName;');
        Active := true;
    end;
end;
```

11. 8 Aggregate functions

Count number of records

```
SELECT Count(*)
FROM tblResults
WHERE Num2 > 50;
```

Find minimum value

```
SELECT MIN(Num2) AS Lowest
FROM tblResults
```

Find maximum value

```
SELECT MAX(Num2) AS Highest
FROM tblResults
WHERE Num1 < 30
```

Get total of a particular field

```
SELECT SUM(Num2) AS Total
FROM tblResults
WHERE Date > #2009/05/01#;
```

Get average of a particular field

```
SELECT AVG(Num2) AS [Num2's Average]
FROM tblResults
WHERE UserID = 11
```

11.9 Date

Year returns year from a date field

Month returns year from a date field

Day returns year from a date field

For example:

```
Select Name, Day(DateBorn) AS DayBorn
FROM tblResults;
```

Date() returns current date

```
Select Name, Date() AS CurrentDate
FROM tblResults;
```

DateValue converts a String value to a date

```
Select * FROM tblResults
WHERE DateBorn > DateValue('' +
Edit1.Text + ''');
```

Returns records where the date of birth is later than the value typed into Edit1.

11.10 Linking tables

Show all (or selected) fields from two different tables with a common field

11.10.1 Using WHERE

```
procedure TForm1.Button3Click(Sender:
TObject);
begin
    with ADOQuery1 do
    begin
        Active := false;
        SQL.Clear;
        SQL.Add('SELECT *');
        SQL.Add('FROM tblTable1, tblTable2
        WHERE tblTable1.UserID =
        tblTable2.UserID;');

        Active := true;
    end;
end;
```

11.10.2 Using INNER JOIN

```
procedure TForm1.Button3Click(Sender:
TObject);
begin
    with ADOQuery1 do
    begin
        Active := false;
        SQL.Clear;
        SQL.Add('SELECT *');
        SQL.Add('FROM tblTable1 INNER JOIN
        tblTable2 ON tblTable1.UserID =
        tblTable2.UserID; ');

        Active := true;
    end;
end;
```

Object Oriented Programming

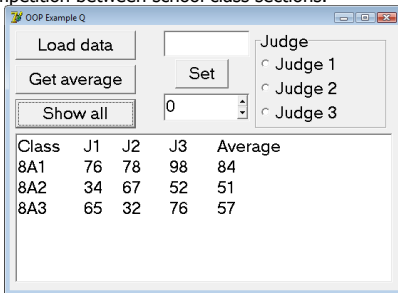
Object Oriented Programming uses objects (self-contained modules) to solve problems. A class can be created in a separate unit.

1. Typical course of action

- 1.1 Open Delphi; save main program
- 1.2 Create a new unit (class unit)
- 1.3 Save the unit (use this name as reference in main program)
- 1.4 Do the following in the class unit:
 - 1.4.1 Type in the word 'Interface'
 - 1.4.2 Add a uses statement – depends on what is needed, but to be safe use: uses SysUtils, Dialogs, Math;
 - 1.4.3 Add type section: type TClassName = class (TObject)
 - 1.4.4 Add properties (declare them in the same way as variables) – possibly under private
 - 1.4.5 Add a constructor (to create object) – possibly under public
 - 1.4.6 Add methods (functions and procedures) – possibly under public
 - 1.4.7 Add the word end;
 - 1.4.8 Standing between the type and end (mentioned above) press Ctrl+Shift+C
 - 1.4.9 Add coding for the methods
- 1.5 Add class unit name in uses section of main program
- 1.6 Add declaration of array of objects in the var section above implementation
- 1.7 Add a counter in the var section above implementation
- 1.8 Add button (or use FormActivate) to retrieve data from a text file and create the objects
- 1.9 Invoke methods to add/edit (set) objects
- 1.10 Invoke methods to get information and display it in a RichEdit for example

2. OOP Example

Create a program that uses OOP to simulate a competition between school class sections.



- Create a program that uses OOP techniques. Use the datafile (oopdata.dat) to populate an array of objects. The object must be constructed with a parameterized constructor. Add *SysUtils* to allow calculations and conversions in your program.

- Create the object unit that includes the following private fields:
 fClass : String;
 fJudge1 : Integer;
 fJudge2 : Integer;
 fJudge3 : Integer;
 fAverage : Integer;

- Add the following methods:
 constructor Create(sClass : String; iJudge1,iJudge2,iJudge3 : Integer);
Get values from text file
 procedure setJudge(iChoice, iMark : Integer);
Change value of certain class per Judge
 function getClass : String;
Return class name (e.g. 8A2)
 function getOutput : String;
Return the output as shown in the RichEdit above
 procedure calculateAverage;
Calculate the average (fAverage) from fJudge1, fJudge2, fJudge3 – remember they are all whole numbers.

- Design a user interface (main unit) that allows to load the data from the textfile (oopdata.dat) into an array of objects (arrClasses). Use the [Load data] button for this purpose.

- Use the [Get Average] button to invoke the 'calculateAverage' method for all the objects in the array.

- Use the [Show All] button and use the getOutput method to display all the objects in the RichEdit. Set adequate tabs.

- Use the [Set] button to change the score of the class specified in the Edit box for the Judge specified in the Radiogroup. Use the getClass method to find the class and invoke setJudge to change the value.

CLASS UNIT

```
unit SingU;
interface

uses SysUtils;
    {Add when using IntToStr for example}

type
  TSingClass = class(TObject)
  private
    fClass : String;
    fJudge1 : Integer;
    fJudge2 : Integer;
    fJudge3 : Integer;
    fAverage : Integer;
  public
    constructor Create(sClass : String;
      iJudge1,iJudge2,iJudge3 : Integer);
    procedure setJudge(iChoice, iMark :
      Integer);
    function getClass : String;
    function getOutput : String;
    procedure calculateAverage;
  end;

implementation

{ TSingClass }

procedure TSingClass.calculateAverage;
begin
  fAverage := (fJudge1+fJudge2+fJudge3) DIV 3;
    {Calculate integer average}
end;

constructor TSingClass.Create(sClass: String;
iJudge1, iJudge2, iJudge3: Integer);
begin
  fClass := sClass;
    {Assign values from text files}
  fJudge1 := iJudge1;
  fJudge2 := iJudge2;
  fJudge3 := iJudge3;
end;

function TSingClass.getClass: String;
begin
  getClass := fClass; {Return class name}
end;

function TSingClass.getOutput: String;
    {Output with tabs}
begin
  getOutput := fClass + #9 +
    IntToStr(fJudge1) + #9 + IntToStr(fJudge2)
    + #9 + IntToStr(fJudge3) + #9 +
    IntToStr(fAverage);
end;

procedure TSingClass.setJudge(iChoice, iMark:
Integer);
begin
  if iChoice = 1 then          {Change values}
    fJudge1 := iMark;
  if iChoice = 2 then
    fJudge2 := iMark;
  if iChoice = 3 then
    fJudge3 := iMark;
end;
end.
```

MAIN UNIT

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants,
  Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, SingU, Spin,
  ExtCtrls;          {Add object unit name}

type
  TForm1 = class(TForm)
    Button1: TButton;
    RichEdit1: TRichEdit;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    RadioGroup1: TRadioGroup;
    Edit1: TEdit;
    SpinEdit1: TSpinEdit;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  arrClasses : array[1..10] of TSingClass;
    {Array of objects}
  iCount : Integer; {Count number of objects}

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
  fFile : TextFile;
  sTemp, sClassI : String;
  iJudge1I, iJudge2I, iJudge3I : Integer;
begin
  AssignFile(fFile,'oopdata.dat');
    {Open text file}
  Reset(fFile);
  While not eof(fFile) do
  begin
    Readln(fFile,sTemp);
    inc(iCount);
    sClassI := copy(sTemp,1,pos(',',sTemp)-1);
      {Separate data fields; commas out}
    Delete(sTemp,1,pos(',',sTemp));
    iJudge1I := StrToInt
      (copy(sTemp,1,pos(',',sTemp)-1));
    Delete(sTemp,1,pos(',',sTemp));
    iJudge2I := StrToInt
      (copy(sTemp,1,pos(',',sTemp)-1));
    Delete(sTemp,1,pos(',',sTemp));
    iJudge3I := StrToInt(sTemp);
    arrClasses[iCount] := TSingClass.Create
      (sClassI,iJudge1I,iJudge2I,iJudge3I);
  end;
  CloseFile(fFile);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  iCount := 0;          {Number of objects to zero}
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  iForCount : Integer;
begin
  FOR iForCount := 1 to iCount do
    {Calculate averages for all objects}
  begin
    arrClasses[iForCount].calculateAverage;
  end;
end;

procedure TForm1.Button4Click(Sender: TObject);
var
  iForCount : Integer;
  sTemp : String;
begin
  RichEdit1.Clear;
  RichEdit1.Paragraph.TabCount := 4;
    {For TABS used}
  RichEdit1.Paragraph.Tab[0] := 70;
    {First tab in points from left margin}
  RichEdit1.Paragraph.Tab[1] := 110;
  RichEdit1.Paragraph.Tab[2] := 160;
  RichEdit1.Paragraph.Tab[3] := 210;
  RichEdit1.Lines.Add('Class' + #9 + 'J1' + #9 +
    'J2' + #9 + 'J3' + #9 + 'Average');
  FOR iForCount := 1 to iCount do
  begin
    sTemp := arrClasses[iForCount].getOutput;
    RichEdit1.Lines.Add(sTemp)
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
  iForCount,iJudge,iNewMark : Integer;
  sClassName : String;
begin
  sClassName := Edit1.Text;
    {Class name}
  iJudge := RadioGroup1.ItemIndex+1;
    {Add one because index start at 0}
  iNewMark := SpinEdit1.Value;
    {New mark allocated by judge}
  FOR iForCount := 1 to iCount do
  begin
    if arrClasses[iForCount].getClass =
      sClassName then
    arrClasses[iForCount].setJudge(iJudge,iNewMark);
  end;
end;

end.
```

*Arrays and StringGrid***1. Arrays (one dimensional)**

```
var
  frmArrays: TfrmArrays;
  arrNumbers : Array[1..100] of Integer;
  iCount : Integer;
```

1.1 Load numbers from a text file into an array

```
procedure TfrmArrays.btnLoadClick(Sender:
TObject);
var
  fTFile : TextFile;
  sTemp : String;
begin
  iCount := 0;
  AssignFile(fTFile, 'numbers.txt');
  Reset(fTFile);
  While not eof(fTFile) do
  begin
    iCount := iCount + 1;
    Readln(fTFile, sTemp);
    arrNumbers[iCount] := StrToInt(sTemp);
  end;
  CloseFile(fTFile);
end;
```

1.2 Remove numbers that repeat in an array

```
procedure TfrmArrays.btnRemoveRepeatsClick
(Sender: TObject);
var
  iOutBound, iInBound, iReplace : Integer;
begin
  FOR iOutBound := 1 to iCount-1 do
  begin
    FOR iInBound := iOutBound+1 to iCount do
    begin
      if arrNumbers[iOutBound] =
        arrNumbers[iInBound] then
      begin
        FOR iReplace := iOutBound to iCount do
        begin
          arrNumbers[iReplace] :=
            arrNumbers[iReplace+1];
        end;
        arrNumbers[iCount] := 0;
        iCount := iCount - 1;
      end;
    end;
  end;
end;
```

1.3 Show numbers from an array in a ListBox

```
procedure TfrmArrays.btnShowArrayClick(Sender:
TObject);
var
  iForCount : Integer;
begin
  ListBox1.Clear;
  FOR iForCount := 1 to iCount do
  begin
    ListBox1.Items.Add(IntToStr
      (arrNumbers[iForCount]))
  end;
end;
```

1.4 Find number (item) in an array

```
procedure TfrmArrays.btnFindClick(Sender:
TObject);
var
  iForCount : Integer;
  iSearch : Integer;
begin
  iSearch := StrToInt(edtFind.Text);
  FOR iForCount := 1 to iCount do
  begin
    if iSearch = arrNumbers[iForCount] then
    begin
      ShowMessage('Number found at: ' +
        IntToStr(iForCount));
      Exit;
    end;
  end;
end;
```

1.5 Binary search in an array

```
procedure TfrmArrays.btnBinarySearchClick
(Sender: TObject);
var
  iForCount : Integer;
  iSearch, iLowBound, iUpBound, iMiddle,
  iPosition : Integer;
  bFound : Boolean;
begin
  bFound := false;
  iLowBound := 1;
  iUpBound := iCount;
  iPosition := 0;
  iSearch := StrToInt(edtFind.Text);
  While (iLowBound <= iUpBound) AND
    (bFound = False) do
  begin
    iMiddle := (iLowBound + iUpBound) DIV 2;
    if iSearch = arrNumbers[iMiddle] then
    begin
      iPosition := iMiddle;
      bFound := True;
    end
    else
    if iSearch > arrNumbers[iMiddle] then
      iLowBound := iMiddle + 1
    else
      iUpBound := iMiddle - 1;
    end;
  end;
  if bFound = True then
  begin
    ShowMessage('Number found at: ' +
      IntToStr(iPosition));
  end
  else
  begin
    ShowMessage('Number not found');
  end;
end;
```

1.6 Sorting**1.6.1 Selection Sort**

```
Procedure TfrmArrays.btnSelectionSortClick
(Sender: TObject);
var
  iCountOut, iCountComp, iTemp : integer;
begin
  FOR iCountOut := 1 to iCount DO
  begin
    FOR iCountComp := iCountOut to iCount DO
    begin
      if arrNumbers[iCountOut] >
        arrNumbers[iCountComp] then
      begin
        iTemp := arrNumbers[iCountOut];
        arrNumbers[iCountOut] :=
          arrNumbers[iCountComp];
        arrNumbers[iCountComp] := iTemp;
      end;
    end;
  end;
end;
```

1.6.2 Bubble Sort

```
procedure TfrmArrays.btnBubbleSortClick(Sender:
TObject);
var
  iTemp, iCounter : Integer;
  bSwap : boolean;
begin
  Repeat
    bSwap := true;
    For iCounter := 1 to iCount - 1 do
    begin
      if arrNumbers[iCounter] >
        arrNumbers[iCounter+1] then
      begin
        iTemp := arrNumbers[iCounter];
        arrNumbers[iCounter] :=
          arrNumbers[iCounter+1];
        arrNumbers[iCounter+1] := iTemp;
        bSwap := false;
      end;
    end;
  until bSwap = true;
end;
```

2. Two dimensional arrays / StringGrid

```
var
  frmStringGrid: TfrmStringGrid;
  arrSeats : array[0..9,0..9] of String;
```

2.1 Load text file into 2D array

```
procedure TfrmStringGrid.btnLoadSeatsClick
(Sender: TObject);
var
  fFile : TextFile;
  sLine : String;
  iRow, iCol : Integer;
begin
  iRow := 0;
  AssignFile(fFile, 'data.txt');
  Reset(fFile);
  While NOT eof(fFile) DO
  begin
    Readln(fFile, sLine);
    For iCol := 0 to 9 do
    begin
      arrSeats[iCol, iRow] := sLine[iCol+1];
    end;
    iRow := iRow + 1;
  end;
  CloseFile(fFile);
end;
```

2.2 Load 2D array into StringGrid

```
procedure TfrmStringGrid.btnDisplayArrayClick
(Sender: TObject);
var
  iRow, iCol : Integer;
begin
  For iCol := 0 to 9 do
  begin
    for iRow := 0 to 9 do
    begin
      sgdNames.Cells[iCol, iRow] :=
        arrSeats[iCol, iRow];
    end;
  end;
end;
```

2.3 Load text file (comma delimited) into a StringGrid

```
procedure
TfrmStringGrid.btnLoadSalesClick(Sender:
TObject);
var
  fFile : TextFile;
  sLine, sTemp : String;
  iRow : Integer;
begin
  iRow := 0;
  AssignFile(fFile, 'comma.txt');
  Reset(fFile);
  While NOT eof(fFile) DO
  begin
    Readln(fFile, sLine);
    sTemp := copy(sLine, 1, Pos(',', sLine)-1);
    Delete(sLine, 1, Pos(',', sLine));
    sgdSales.Cells[0, iRow] := sTemp;

    sTemp := copy(sLine, 1, Pos(',', sLine)-1);
    Delete(sLine, 1, Pos(',', sLine));
    sgdSales.Cells[1, iRow] := sTemp;

    sTemp := copy(sLine, 1, Pos(',', sLine)-1);
    Delete(sLine, 1, Pos(',', sLine));
    sgdSales.Cells[2, iRow] := sTemp;

    sgdSales.Cells[3, iRow] := sLine;

    iRow := iRow + 1;
  end;
  CloseFile(fFile);
end;
```

2.4 Clear a StringGrid

```
procedure
TfrmStringGrid.btnClearGridClick(Sender:
TObject);
var
  iRow, iCol : Integer;
begin
  For iCol := 0 to 9 do
  begin
    for iRow := 0 to 9 do
    begin
      sgdNames.Cells[iCol, iRow] := '';
    end;
  end;
end;
```

*Arrays and StringGrid; Mathematical functions; String handling***2.5 Set all values in a StringGrid to 0**

```
procedure TfrmStringGrid.btnSetAllOClick
(Sender: TObject);
var
  iRow, iCol : Integer;
begin
  For iCol := 0 to 9 do
    begin
      for iRow := 0 to 9 do
        begin
          sgdNames.Cells[iCol,iRow] := '0';
        end;
      end;
    end;
end;
```

2.6 Save data from StringGrid in a text file

```
procedure TfrmStringGrid.btnSaveNamesClick
(Sender: TObject);
var
  fFile : TextFile;
  sLine : String;
  iRow, iCol : Integer;
begin
  iRow := 0;
  AssignFile(fFile, 'data.txt');
  Rewrite(fFile);
  For iRow := 0 to 9 do
    begin
      sLine := '';
      For iCol := 0 to 9 do
        begin
          sLine := sLine + sgdNames.Cells[iCol,
            iRow];
        end;
      Writeln(fFile, sLine);
    end;
  CloseFile(fFile);
end;
```

2.7 Count Xs and Os in a StringGrid

```
procedure TfrmStringGrid.btnCountClick(Sender:
TObject);
var
  iRow, iCol, iCountX, iCountO : Integer;
begin
  iCountX := 0;
  iCountO := 0;
  For iCol := 0 to 9 do
    begin
      for iRow := 0 to 9 do
        begin
          if sgdNames.Cells[iCol,iRow] = 'X' then
            iCountX := iCountX + 1
          else
            if sgdNames.Cells[iCol,iRow] = 'O' then
              iCountO := iCountO + 1
            end;
        end;
      lblTotalX.Caption := 'Total Xs: ' +
        IntToStr(iCountX);
      lblTotalO.Caption := 'Total Os: ' +
        IntToStr(iCountO);
    end;
end;
```

2.8 Change values of StringGrid according to coordinates from SpinEdits

```
procedure TfrmStringGrid.bntChangeClick(Sender:
TObject);
var
  iCol, iRow : Integer;
begin
  iCol := sedColumn.Value;
  iRow := sedRow.Value;
  if sgdNames.Cells[iCol, iRow] = 'X' then
    begin
      sgdNames.Cells[iCol, iRow] := 'O';
    end
  else
    sgdNames.Cells[iCol, iRow] := 'X';
  end;
```

2.9 Get totals of columns in a StringGrid

```
procedure TfrmStringGrid.btnGetTotalClick
(Sender: TObject);
var
  rTotal : Real;
  iRow, iCol : Integer;
begin
  sgdSales.Cells[0,5] := 'TOTAL';
  For iCol := 1 to 3 do
    begin
      rTotal := 0;
      for iRow := 1 to 4 do
        begin
          rTotal := rTotal +
            StrToFloat(sgdSales.Cells[iCol,iRow]);
        end;
      sgdSales.Cells[iCol,5] :=
        FloatToStr(rTotal);
    end;
end;
```

2.10 Look for value in StringGrid

```
procedure TfrmStringGrid.btnFindClick(Sender:
TObject);
var
  sFind : String;
  iCol, iRow : Integer;
begin
  sFind := edtFind.Text;
  For iCol := 1 to 3 do
    begin
      for iRow := 1 to 5 do
        begin
          if sgdSales.Cells[iCol,iRow] = sFind then
            begin
              Showmessage('Found at coordinates:
                Column ' + IntToStr(iCol) + ' ; Row ' +
                IntToStr(iRow));
            end;
          end;
        end;
      end;
    end;
```

2.11 Moving item around in a StringGrid**2.11.1 Reset**

```
procedure TfrmStringGrid.btnResetClick(Sender:
TObject);
begin
  For iPosCol := 0 to 9 do
    begin
      For iPosRow := 0 to 9 do
        begin
          sgdMoveArea.Cells[iPosCol,iPosRow] := '';
        end;
      end;
    end;
  iPosCol := 0;
  iPosRow := 0;
  sgdMoveArea.Cells[iPosCol,iPosRow] := 'X';
end;
```

2.11.2 Move up

```
procedure TfrmStringGrid.btnUpClick(Sender:
TObject);
begin
  sgdMoveArea.Cells[iPosCol,iPosRow] := '';
  iPosRow := iPosRow - 1;
  sgdMoveArea.Cells[iPosCol,iPosRow] := 'X';
end;
```

2.11.3 Move down

```
procedure TfrmStringGrid.btnDownClick(Sender:
TObject);
begin
  sgdMoveArea.Cells[iPosCol,iPosRow] := '';
  iPosRow := iPosRow + 1;
  sgdMoveArea.Cells[iPosCol,iPosRow] := 'X';
end;
```

2.11.4 Move left

```
procedure TfrmStringGrid.btnLeftClick(Sender:
TObject);
begin
  sgdMoveArea.Cells[iPosCol,iPosRow] := '';
  iPosCol := iPosCol - 1;
  sgdMoveArea.Cells[iPosCol,iPosRow] := 'X';
end;
```

2.11.5 Move right

```
procedure TfrmStringGrid.btnRightClick(Sender:
TObject);
begin
  sgdMoveArea.Cells[iPosCol,iPosRow] := '';
  iPosCol := iPosCol + 1;
  sgdMoveArea.Cells[iPosCol,iPosRow] := 'X';
end;
```

3. Mathematical functions

StrToInt
Converts string value to integer
iN := StrToInt(sNumber);

IntToStr
Converts integer value to string
sNumber := IntToStr(iN);

StrToFloat
Converts string value to real
rN := StrToFloat(sNumber);

FloatToStr
Converts real value to string
sNumber := FloatToStr(rN);

FloatToStrF
Converts real value to string with formatting (1 number after the decimal for example)
sN := FloatToStrF(rN, ffFixed, 15, 1);

Trunc
Truncates (cuts) the decimal point
Trunc(4.5);

Val
Converts string value to integer or real. Also checks for errors.
Val(sOriginal, iNum, iError);
Val(sOriginal, rNum, iError);

Round
Rounds real number to the nearest integer
Round(4.8);

Roundto
Rounds to a set power of 10
iH := Roundto(2745,3);

Inc
Increases value of variable
(You can also do this as:
iNumber := iNumber + 1;)
Inc(iNumber)

Dec
Decreases value of variable
(You can also do this as:
iNumber := iNumber - 1;)
Dec(iNumber);

Frac
Provides decimal part of real number
rK := Frac(3.54);

Sqr
Gives square of number typed in
rK := Sqr(16);

Sqrt
Gives square root of a number
rK := Sqrt(16);

Power
Raises first number to the power of the second number (134)
Power(13,4)

Pi
Provides the value of pi (n)
rW := rK * Pi;

Random
Provides a random number within a range of 0 and limit-1.
Randomize;
iX := Random(100);
(iX = any number between 0 and 99)

Remember to add the Math unit to the uses section of your program when using the RoundTo and Power functions.

4. String handling

sSourceText := 'The man walks'

- Determine the position of a piece of a text within a string
IntegerVariable := Pos(StrToBeFound, SourceText);
For example: iX := Pos ('m', sSourceText);
{The value of iX is now: 5}

- Display a certain character within a string using square brackets at the end of a variable.
StringVariable := SourceText[CharacterPosition];
For example: sNewText := sSourceText [2];
{The value of sNewText is now: 'h'}

- Display a certain section of text within a string.
For example: **StringVariable := Copy(SourceText, BeginPosition, Length);**
sNewText := Copy(sSourceText, 5, 3);
{The value of sNewText is now: 'man'}

- Insert a certain section of text within a string.
For example: **Insert(InsertText, SourceText, Position);**
Insert ('big ', sSourceText, 5);
{The value of sSourceText is now: 'The big man walks'}

- Remove a certain section of text within a string.
For example: **Delete(SourceText, Position, Length);**
Delete(sSourceText, 5, 4);
{The value of sSourceText is now: 'The walks'}

- Determine the length of a string.
For example: **IntegerVariable := Length(SourceText);**
iX := Length(sSourceText);
{The value of iX is now: 13}

- Change the whole string to lowercase.
For example: **LowerCase(SourceText);**
LowerCase(sSourceText);
{The value of sSourceText is now: 'the man walks'}

- Change the whole string to uppercase.
For example: **UpperCase(SourceText);**
UpperCase(sSourceText);
{The value of sSourceText is now: 'THE MAN WALKS'}
Take note: Use UpCase for Char type.