

**NEAR EAST UNIVERSITY**

**Faculty of Engineering**

**Department of Computer Engineering**

**Specialized School Information System**

**Graduation Project**

**COM- 400**

**Student: Pelin BEZGIN SOY(20010486)**

**Supervisor: Asst. Prof. Dr. Firudin Muradov**

**Nicosia – 2006**

## **ACKNOWLEDGEMENTS**

First of all, I wish to thank my supervisor, Assist.Prof.Dr. Firudin Muradov, for intellectual support, encouragement, and enthusiasm, which made this project possible, and his patience for correcting both my stylistic and scientific errors.

I would like to thank my family who gave their lasting encouragement in my studies and enduring these all expenses and supporting me in all events, so that I could be successful in my life time. I specially thank to my mother and my father who help me in joining this prestigious university and helped me to make my future brighter.

All people who have contributed in the preparation of my project to complete it successfully.

I would like to thank my sister Elif Bengi Bezginsoy, who supported and helped me all the time.

I am also very much grateful to all my friends who gave their precious time to help me and all valuable information which I really need to complete my project.

## TABLE OF CONTENTS

<b>ACKNOWLEDGEMENT</b>	I
<b>TABLE OF CONTENTS</b>	II
<b>ABSTRACT</b>	V
<b>INTRODUCTION</b>	1
<b>1. .BASIC CONCEPT OF DELPHI</b>	<b>2</b>
1.1 What is DELPHI?	2
1.2 Starting to Delphi Programing	3
1.2.1 The Form	3
1.2.2 The Code Editor	4
1.2.3 The Speedbar	5
1.2.4 The Component Palette	5
1.2.5 The Object Inspector	6
1.2.6 Events	7
1.2.7 Exit Program	7
1.2.8 Boxes	8
1.2.9 Inputting Data	8
1.2.10 Additional Component Palette	10
1.2.11 System Palette	13
1.3 Standart I/O Component	14
1.3.1 Panel Component	14
1.3.2 Group Box Component	14
1.3.3 Check Boxes	15
1.3.4 Group boxes, Radio buttons and Radio Group Boxes	15
1.3.5 Spin Dials	16
1.3.6 Tab Order	17
1.3.7 Dialog Boxes	17
1.3.7.1 ShowMesage Procedure	17
1.3.7.2 The MessageDlg Function	17
1.4 Input Forms	18
1.4.1 Input Box	18
1.4.2 CustomBoxes	19

1.4.3 Main Menu	19
1.5 Forms	20
1.5.1 Form Templates	20
1.5.2 Multiple Documents	20
1.5.3 Form Style Property	21
1.5.4 Project Manager	21
1.5.5 Units	21
1.5.6 Declarations	22
1.5.7 Comments	23
1.6 Loops	23
1.6.1 If...Then	23
1.6.2 If...Then...Else	24
1.6.3 The Case Statement	24
1.6.4 For...To...Do...Statement	25
1.6.5 While...Do...Statement	26
1.6.6 Repeat...Until	26
1.7 Procedures and Functions	27
1.7.1 Parameter Passing	27
1.7.2 Functions	28
1.7.3 Recursions	29
1.7.4 Variable Declarations	30
1.8 Debugging	30
1.8.1 Program Errors	30
1.8.2 Using The Debugger	30
1.8.3 Running The Program To A Breakpoint	31
1.8.4 Setting Breakpoints	31
1.8.5 Breajpoints Options	32
1.8.5.1 Viewing Breakpoints	32
1.8.5.2 Conditional Breakpoints	32
1.9 File I/O	33
1.9.1 File Attributes	34
1.9.2 File Operation	34
1.9.3 Typed Files	35

1.9.3.1 Typed File Commands	35
1.9.4 Untyped Files	35
1.9.5 File Handling Components	36
<b>2. DATABASE CONCEPT OF DELPHI 7</b>	<b>38</b>
2.1 About Dbase And Paradox	38
2.1.1 dBASE IV Table Specification	38
2.1.2 dBase V Table Specifications	38
2.1.3 dBASE Field Types	39
2.2 Paradox Standard Table Specifications	40
2.2.1 Paradox 5 Table Specifications	42
2.2.2 Paradox 7 and Above Table Specifications	42
2.2.2.1 Paradox Field Types	43
<b>3. DATABASE DESIGN OF THE PROGRAM</b>	<b>46</b>
3.1 Database Design of The Program	46
<b>CONCLUSION</b>	<b>66</b>
<b>REFERENCES</b>	<b>67</b>
<b>APPENDIX A: Program Codes</b>	<b>68</b>
<b>APPENDIX B: Database Tables</b>	<b>88</b>

## **ABSTRACT**

Data, gathered around us as a collection of facts, is of no use unless it is organized and represented in some meaningful form. Data represented in some meaningful form like, tables, charts, or graphs become information, which can be easily processed.

The collection of data, usually refereed to as the database, contains information about one particular enterprise. These days' databases are used by a variety of users and organizations, which are important tools in processing DBMS, which are designed to manage large amount of data.

This project has as its goal to develop software, processing information about activities of a specialized schoool information system. Software developed in this project contains both student and teacher information. The ideas of the project can be improved by adding items for processing all activities of the specialized schoool information system.



## INTRODUCTION

Nowadays the technology is developed a lot and started to use by anyone in the world no matter who is he/she. Because of the technology is available at every platform of our life human needs to combine both software and hardware. Without human direction, hardware is a useless amalgam of metal and plastic. The computer science both hardware and software is being developed over the previous years , programming is always providing the science by a systematic development.

A DBMS is a computerized record-keeping system that stores, maintains and provides access to information. A database system involves four major components data, hardware, software, users. The objective of this project was to design software for a specialized school information system, which include teacher and student information, so fully qualified software has been made. The Software is fully capable to reach any information about teacher and student with student or teacher name.

The project consists of introduction, three chapters, conclusion and references.

Chapter one describes the general information about delphi programming that how to use components, forms, procedures and functions, debugging to solve errors of program.

Chapter two describes database concept of delphi 7 which contains dBase and paradox table specifications and fields types.

Chapter three defines database design of the program, fields of tables ,relationships between tables and some forms of programs with their explanations that how to use forms .

# CHAPTER 1

## 1. BASIC CONCEPT OF DELPHI

### 1.1 What is DELPHI ?

Borland Delphi 2006 is the latest Integrated Development Environment product release from Borland Software Corporation. This is now the tenth version of Delphi, Borland's flag ship Rapid Application Development (RAD) environment, and this paper provides an introductory look at what makes this release compelling.

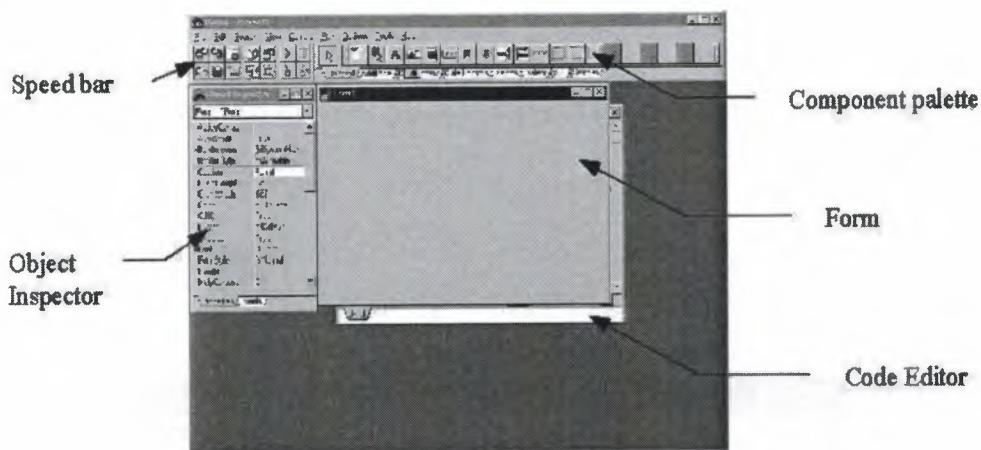
What has always set Borland apart from other vendors has been its pragmatic approach to providing developer tools that are right for the challenges that commercial developers face today, while leading them into the emerging technologies of tomorrow with the confidence that their development investments remain relevant, adaptable and extendable in the future.

Delphi 2006 continues the tradition and has extended capabilities in significant areas with considerable implications for software developers at every level. This document focuses on the Delphi capabilities within this release and is not intended to cover what's new for the C/C++ language specific capabilities.



## 1.2 Starting to Delphi Programming

Delphi is a programming language that easily lets you write Windows based programs. The Integrated Development Environment IDE for Delphi enables you to create, run and debug your program from a universal 'front-end' (user interface). The screen that greets the programmer is similar to the following:

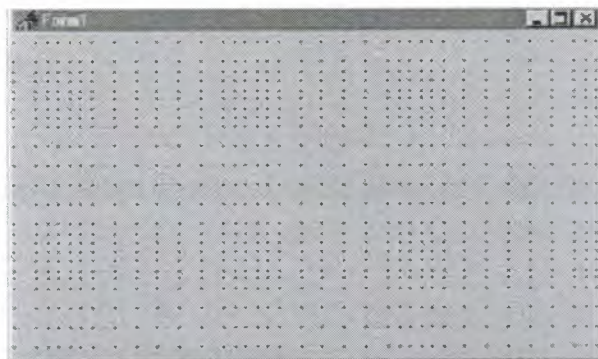


**Figure 1.1** Delphi screen

It has the usual Windows File, Edit, View menu controls, and consists of five parts (clockwise from top left): The Speedbar, Component Palette(s), Form, Code Editor (behind the form) and the Object Inspector.

### 1.2.1 The Form

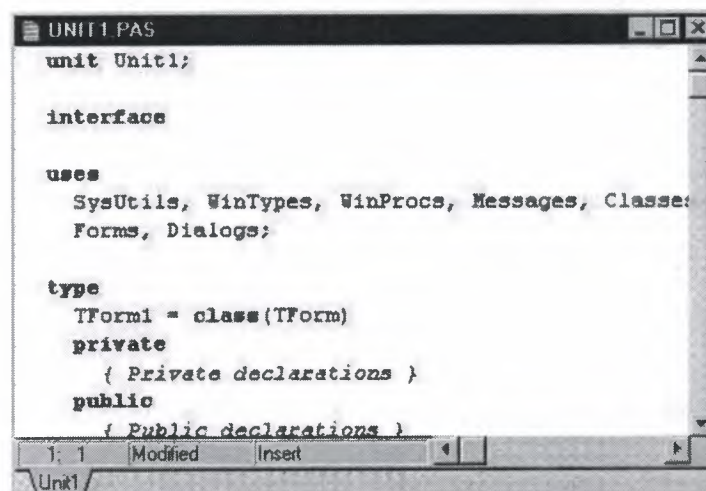
This is the most important part of the development and final application. In development, it is where the 'components' such as display boxes and buttons are placed, and in the application it is what the user finally sees.



**Figure 1.2 Form**

The grid points are to help the user place components on the form, and do not appear at runtime.

### 1.2.2 The Code Editor



**Figure 1.3 Code Editor**

Behind the form is the code editor. This can be brought to the front by clicking it, or it will automatically appear. When an item placed on the form is 'double-clicked', thus enabling the code to be written for that component. You will notice that some code is already inserted for you, but you will still have to write the important parts!

### 1.2.3 The Speedbar

The Speedbar contains a shortcut way of using some of Delphi's more commonly used features.

The default speedbar contains three groups for

- Project Management
- File and Unit management
- Debugging

It can be configured to include any of the items in the menu.



Figure 1.4 Speedbar

### 1.2.4 The Component Palette

The Component Palette(s) contain all the expected features of the user interface, in particular buttons, text boxes, list boxes, etc. In order to place a component on the form, simply click on the component palette, then click on the form.



Figure 1.5 Component Palette



As you write Delphi applications, you will notice the layout as use of the components in other applications. Clicking the tabs (e.g. 'Additional', 'Data Access' etc.) will bring up further components.

## 1.2.5 The Object Inspector

This provides information about the 'objects' on the form and the 'events' that can happen to an object (hence the two pictures below).

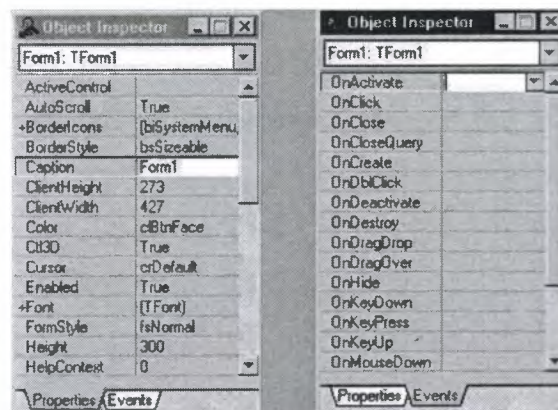


Figure 1.6 Object inspector

For instance an object such as a 'Label' will have a position on the form, a certain height and width, a particular font and size, a background colour, borders etc - these are known as its **properties**.

Events happen to an object such as it being clicked or double-clicked, or when the mouse moves over it etc.

The Object Inspector here shows information about the Form itself, i.e. it has the default caption 'Form1' and an event 'OnActivate' will run some code when the form is activated

Clicking on an object on the form will bring up the relevant information on the object inspector.

Help on properties can be obtained by clicking the property and pressing the F1 function key.

For the Help function to work, you must copy the delphi.hdx file to their directory

### **1.2.6 Events**

Delphi is an example of an Event driven program. The program responds to the **user** or events that happen, rather than just proceeding through a sequence of instructions as programs you have written in the past. However this can make programs difficult to debug.

some examples

- OnClick
- OnKeyDown
- OnKey Press
- OnEnter
- OnExit
- OnMouseDown
- OnMouseU
- OnMouseMove

### **1.2.7 Exit program**

Create a button labelled 'Close' and add the code :

Close; or Application.Terminate;



### 1.2.8 Boxes

**Edit Box** - one line of text can be displayed or entered.

**Memo Box** -display or enter manylines of text.

**List Box** -Scrollable list of items that can be selected, but not altered by the user

More Edit Box Properties:

- Name Default (Edit1)
- Text
- BorderStyle
- Color

Position :

- Height
- Width
- Top
- Bottom
- Hint
- ShowHint

A List Box doesn't have a 'text' property - as it has a number of lines, it has 'Items (0, 1, 2etc.)'

Text is added using **ListBox1.Items.add(FloatToStr (Result));**

### 1.2.9 Inputting data

All data in boxes is (are) treated as a string.To convert data to numbers, then conversion routines must be used.

The basic ones are:

**StrToInt** and **StrToFloat** , which convert the string to an integer or floating point number and**FloatToStr** and **IntToStr**, which do the opposite.

## components on the standard palette



**Main Menu** - this is used to design a main menu (e.g. File Edit View .. Help) for application.



**Pop-up Menu** - similar to Main Menus, but do not have a menu bar - they appear when the right mouse button is clicked.



**Label** - used to display text that the user cannot change (but can be changed by the program).



**Edit box** - used to enter or display a single line of text.



**Memo** - used to display or edit many lines of text.




**Button** - used to initiate an action, they can only have one line of text. Main events are OnClick and OnDbleClick, and the text on the button is held in the Caption property.





**Check button** - used to select items (true or false - true if checked). The checked property indicates if checked or not, and it can be made unavailable to the user with the enabled property





**Radio button** - similar to the Check button, but usually grouped in a radio group, scroll or panel box, where only one item in that group is selected (e.g. AM/FM/SW radio buttons)


 **List box** - used to display a list from which the user can select (but not modify). The list can be modified by the program only, and will automatically scroll if the box is not large enough.

 **Combo box** - similar to a list box, but allows the user to select from a list or type in their own text. The Items property contains the text in a string list

 **Scroll** - used to control the screen display (for example if text is too big for the screen), or used to select values from a range. (In this case max and min values are defined properties)

 **Group** - Used to group items, so that they are treated together as a single group.

 **Radio Group** - provides an easy way of creating a set of radio buttons, in one action, rather than creating the buttons, and then grouping using Group.

 **Panel** - Used to combine previous components onto their own area, e.g.. used for creating tool bars or palettes. The panel appearance can be changed with bevel and border properties.

### 1.2.10 Additional Component Palette

The Additional component palette looks like:



**Figure 1.7** Additional component palette

The components of the Additional palette perform the following functions:



The **BitBtn** component acts like an ordinary button but has an icon, also called a glyph, on the button face. The Kind property provides some common glyphs, or a custom one may be defined. There is a Caption property and if this is blank, then the glyph only will be displayed.



The **SpeedButton** are button components without captions, and used to create tool bars or tool palettes. Speed buttons can have up to four images associated with them depending on the state of the button: Up: unselected; Down: Selected; Disabled: the button is not available and the image is greyed out; StayDown: the button is permanently selected.



The **TabSet** component presents horizontal tabs users can click to initiate actions, like the palette set themselves. Tab set controls are commonly used with TNotebook controls to display pages within the same dialog box.



The **Notebook** component is a component that can display multiple pages, each with its own set of controls. Notebook components are frequently used with tab set controls to let the user select pages in the notebook by clicking a tab.



The **TabbedNotebook** component contains multiple pages, each with its own set of controls. The user selects a page by clicking the page's tab.



The **MaskEdit** component is used to format data entry and check for proper data input. The EditMask property defines common inputs such as: telephone numbers '(415)555-1212', dates '06/27/94' and time '09:05:15PM'. Custom input formats and number ranges can be defined.





The **Outline** component is used to display multilevel outlines of data in a hierarchical tree.



The **StringGrid** component is a subset of the DrawGrid component and provides a table for displaying string information



The **DrawGrid** component displays text or graphic information in a cell. . The cell selected is found in the Selection property; cells can be filled at run-time using the OnDraw Cell event and the MouseToCell event determines which cell the mouse is currently over.



The **Image** component displays a graphical image on a form. The image can contain an icon, metafile, or bitmap graphic. The stretch property will resize the image to fit the component, and the Autosize property resizes the image control to fit the image.



The **Shape** component is used to draw simple shapes on the form.



The **Bevel** component puts beveled lines, shapes and boxes on the form by using its Shape property. The Style property determines if the bevel is raised or lowered.



The **Header** component is a sectioned visual control that displays text and allows each section to be resized with the mouse.



The **ScrollBox** component is used to contain a larger area which can be scrolled horizontally or vertically.



### 1.2.11 System Palette

More sophisticated Delphi programs use a dialog box to select and open the desired files. This is easily done in Delphi using components from the System Palette.



**Figure 1.8** System palette

From left to right the components are:

**Timer.** This component causes an **OnTimer** event after a specific period of time depending on the **Interval** property.

**Paint Box.** This provides its own 'canvas' to paint on, rather than painting on the form.

**FileListBox.** This provides a list box which displays files in the currently selected directory.

**DirectoryListBox.** This provides a list box which displays directories in the currently selected drive.

**DriveComboBox.** Provides a list box that shows the current drive.

**FilterComboBox.** Provides a list of filters (e.g. \*.doc) which can be selected to limit the display of files.

**MediaPlayer.** This can be used to control devices which have a Media Control Interface (MCI)

**OLE.** The Object Link and Exchange component provides the ability to link and embed objects.

### 1.3 Standard I/O Components.

- Panel Component.
- Group Box Component.
- Radio Buttons.
- Check Boxes.
- Spin dials.
- An extension, Gauges.

#### 1.3.1. Panel Component

The Panel component is used to contain components or other containers such as Group boxes. The appearance of the panel component can be changed by altering the Bevel inner, outer and width parameters, and Border Style and Width items.

The Panel Component is :



#### 1.3.2 Group Box Component

Delphi components can be grouped together on the form to make the user interface more obvious and easy to use. The Group Box border cannot be altered like the Panel Component, but has a Caption.

The Group Box Component is :



### 1.3.3. Check Boxes

Check boxes are used to select items, e.g. items from a list:

does your computer have a) a hard disk, b) a CD ROM, c) a sound card, d) TV card etc

The Check Box icon is:



Check Boxes can be checked, unchecked or disabled and 'greyed' out by changing their properties:

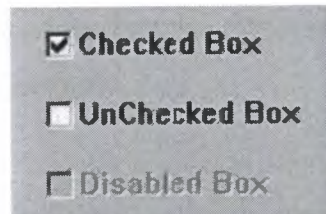


Figure 1.9 Check boxes

### 1.3.4 Group boxes, Radio buttons and Radio Group Boxes

Group boxes can be used to group items together, but as radio buttons are often grouped together, a Radio Group Box is a special component designed for this purpose. Radio buttons in Delphi, are just like those on your radio. Only one of a group can be selected, and pushing one, makes the others 'pop up', or be deselected. They are used for selecting only one item of a group e.g. male/female, 1<sup>st</sup> year/2<sup>nd</sup> year/ 3<sup>rd</sup> year, Age groups, Country of origin etc.

An application may have more than one group of radio buttons however, so they can be grouped using a Radio Group Box'.

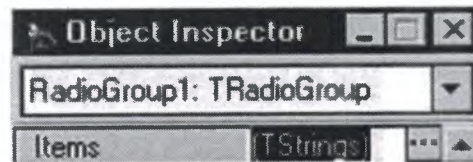
The radio button icon is:



and the radio group box icon is:



'Items' property from the Object inspector, and click the box with the dots in it :



**Figure 1.10** Items

This displays a dialog box where you simply type the names of the boxes.

Type in the names :FM, AM, LW, SW, on individual lines. Change a few of the Radio group box properties, in particular the align and see what happens to the radiobuttons. Captions are placed to the right of the buttons. With individual radio buttons, captions can be on the left or right of the buttons.

### 1.3.5 Spin Dials

A frequency has to be chosen for the radio station. This could be done using an Edit Box, but sometimes it is more useful to use 'Up' and 'Down' arrows to select from a range. Spin Edit boxes are used to do this.

The spin edit boxes have some useful properties including minimum and maximum values. For example, the FMrange can be limited between 88 and 108 MHz.

The Spin edit box is found in the SAMPLE component Tab. The SAMPLE component tab consists of: gauges, colour grid, spin button, spin edit, directory outline and calendar



**Figure 1.11** Spin dials

and Spinedit box is fourth item and looks like:





### 1.3.6 Tab Order

The tab order tells the program how to move between the boxes when the tab key is pressed when the program is running, (If Tab stop is TRUE).

### 1.3.7 Dialog Boxes

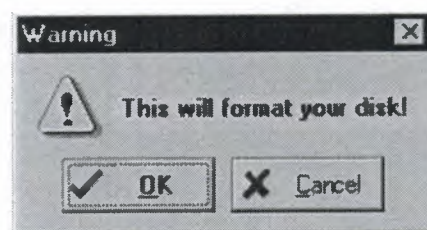
These are not components, but appear when the program runs. Dialog boxes are used to display information and messages, and accept limited input. You can use standard dialog boxes or create your own.

#### 1.3.7.1 ShowMessage Procedure

The simplest way to display a message is to use the ShowMessage procedure  
`ShowMessage('This message is displayed with an OK button');`

#### 1.3.7.2 The Message Dlg Function

If you want to accept input from a selection of buttons (e.g. Yes, No, Cancel), the MessageDlg Function is used. The MessageDlg function returns a value indicating which button has been clicked. An example Warning Message Dialog box is shown:



**Figure 1.12** Warning Message Dialog Box



## 1.4 Input Forms

Dialog boxes can also be used to prompt for input, using the two function InputBox and InputQuery.

The InputBox displays an edit box prompting for input, along with OK and Cancel buttons. It returns the value in the box if the OK button is pressed, or the default string, if cancel button or escape key pressed. The function is called with : InputBox (ACaption,, APrompt, ADefault : string) :string;

The function returns a string and the parameters (strings) passed to the function are:

Table 1 function Parameters

Parameter	Description	Example
ACaption	the caption of the box	'Input Box Title'
APrompt	the prompt at the side of the box	'Enter text here'
ADefault	the initial text in the box	'Default text'

### 1.4.1 InputBox

('Input Box Title', 'Prompt', 'Default string');

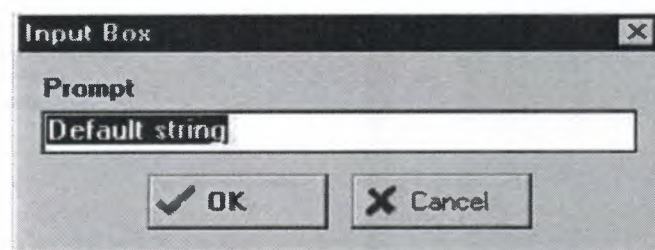


Figure 1.13 Input box

### 1.4.2 Custom Boxes

The dialog boxes provided by Delphi cover a wide range of requirements, but programmers may like to create their own boxes. This is accomplished through the use of forms.

### 1.4.3 Main Menu

Delphi provides Dialog boxes to Open, Save and Print files. These can be displayed using a button, or more professionally by a Main Menu. Place a Main Menu on a form, click it to start a menu list, then right-click the list. The following appears:

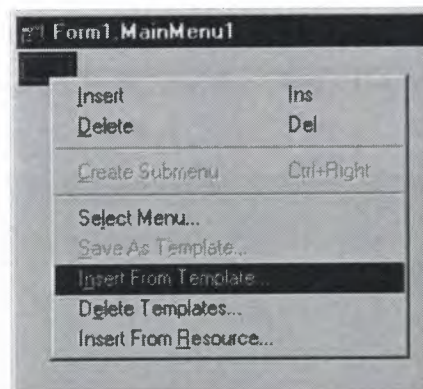


Figure 1.14 View Main Menu

Selecting 'Insert from Template' displays a number of common templates, including File and Edit. Selecting the File template adds the following File Menu.

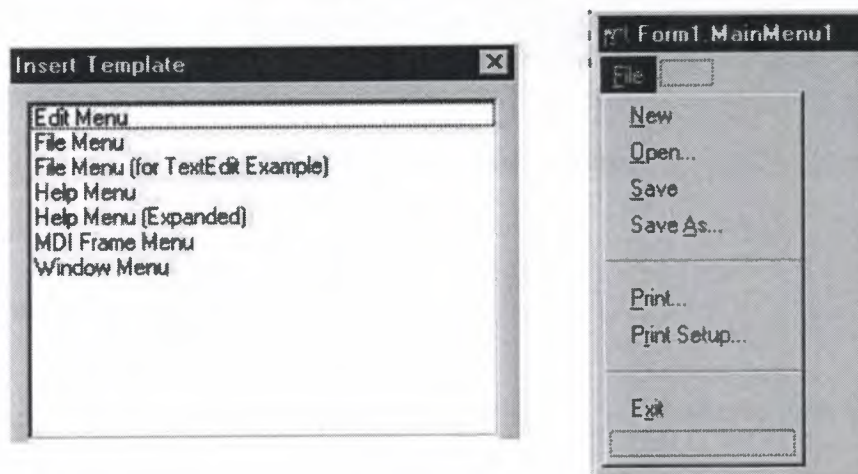


Figure 1.15 Insert from Template

## 1.5 Forms

Most application will use more than one form. There are several different types of forms, and some default forms are provided by Delphi. Forms can be added to applications by using File | New form or File | New | New Items | Forms (Delphi 2.0 +)

### 1.5.1 Form Templates

Some of the available forms are:

Standard dialog box - with OK and Cancel buttons, or its alternative with the buttons on the right.

About Box - dialog box with help

Password - dialog box with a password edit box

Tabbed Notebook - a form with Tabs, like the Object Inspector or Project | Options

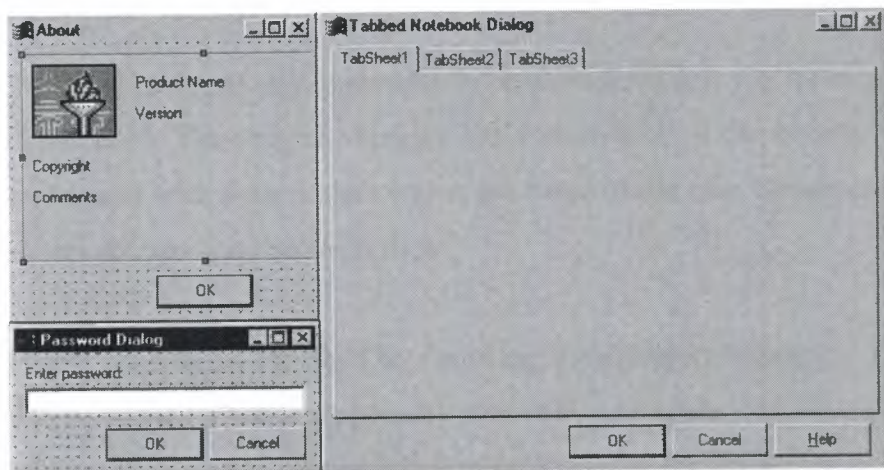


Figure 1.14 Example of forms .

### 1.5.2 Multiple Documents

In some applications you want forms or windows to be opened within other forms, rather than in their own right.



This is known as multi document interface, or MDI. In this case the forms are opened as 'child' forms of the 'parent' form. There can only be one parent form, but as many child forms as required. Single document interface, or SDI, forms have only a single parent window.

### **1.5.3 Form Style Property**

The `FormStyle` property has four possible values:

`fsNormal` - The form is a Single Document interface (SDI)

`fsMDIform` - The form is an MDI parent form

`fsMDIChild` - The form is an MDI child form

`fsStayOnTop` - The form is SDI and stays on top of other forms

### **1.5.4 Project Manager**

You may have found that if you inadvertently create a form that you no longer require, you cannot delete it. The Project Manager lets you do this. It displays the forms and unit files associated with your project which are listed in the uses clause of your `.DPR` file. It also lets you navigate between files..

You can access the Project Manager by choosing `View|Project Manager`. The Project Manager can be opened when you open any project by setting the desktop settings.

If you have a lot of files, possibly in different projects, use the Project Manager. It will show you the location of each file in the project. This will help you when you create backups that include all files in the project.

### **1.5.5 Units**

You will have noticed by now that you save your project as a unit, and the Delphi program uses other code which in fact are also units such as

SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls, Forms

Units enable the programmers to write and compile their code and save them as an individual, stand-alone library items. The unit can be distributed and because it is compiled, other users are unable to see the code.

### 1.5.6 Declarations

All variables and constant names used and their *type* have to be declared. This is done at the beginning of the code, before the procedure declaration. Constants are declared using the word `const`, and variables are declared using the word `var`.

Variables have the following basic data *types*:

Table2 Variable data types

Data type	Number of bytes used	Description
Byte	1	0 to 255 unsigned integers
Word	2	0 to 65535 unsigned integer
ShortInt	1	-128 to +127
Integer	2	-32768 to +32767
Longint	4	-2147483648 to +2147483647
Single	4	floating point numbers
Double	8	
Extended	10	
Real	6	for compatibility - do not use
Boolean	1	TRUE or FALSE
Char	1	letters of the alphabet : 'A' or '**'
String	0 to 255	
Pointer	2	
PChar	2	pointer to a character
Variant	-	



### 1.5.7 Comments

{this is a comment- it is contained within curly brackets}

## 1.6 Loops

### 1.6.1 If .. Then

The simplest decision making construct is the if .. then command.

It takes the form of:

If <condition> then <statement>;

The <condition> is a logical comparison which produces a true or false (boolean) answer. The logical comparisons take the following forms:

For example:

If salary > 1000000 then Edit1.text := 'I'm a millionaire'; {note the use of quotes}

Table 3 Example of operator

Operator	English	Example
	is equal to	(2+2) = 4
	is not equal to	(1+2) <> 4
	is less than	X < 2 Y
	is less than or equal to	<= 3 X >
= <> <	greater than is	Y (X+Y)
> >=	greater than or equal to	>= Z

These are also known as relational operators. Note that the colon-equals sign (:=) is for assignment in expressions, and equals sign alone (=) is used for comparison. The <condition> test does not have to be simple.

### 1.6.2 If .. Then .. Else

A more useful version is the if .. then .. else , which takes the form of:

If <condition> then <statement1> else <statement2>

If expenditure > income then Edit1.text := 'Miserable' else Edit1.text := 'Happy';

If .. then statements can be nested (i.e. included within other if .. then statements) and so can else statements, but this can lead to very confusing programs.

### 1.6.3. The Case Statement

There will come a time when there will be multiple if.. then statements, e.g.

If (age = 12) then edit1.text := 'you are nearly a teenager';

If (age = 13) then edit1.text := 'you are a teenager';

If (age = 18) then edit1.text := 'you can vote';

The case statement makes programming this easier. The format is

Case <variable> of list else <statement> The above code would become:

Case age of

12: edit1.text := 'you are nearly a teenager';

13: edit1.text := 'you are a teenager';

18: edit1.text := 'you can vote';

else

edit1.text := 'You are not 12,13 or 18';

end;

The case variable can only be of type an integer, char or an enumerated type. An enumerated type lists all the possible values that a variable of that type can have, e.g. a type 'day' could be defined (consisting of Sunday to Saturday).

#### 1.6.4 For .. To .. Do Statement

It is a common requirement in programming to perform some instructions a set number of times, i.e. to loop. Delphi provides a for .. to .. do loop construct\*for this, it takes the form of:

For <variable> := <start value> to <final value> do statement

An example will make this clearer.

Suppose the sum of the first 10 numbers needs to be found, the loop can be used to simply do it.

For count := 1 to 10 do

Points to note

- The loop variable can only be an integer.
- The loop variable does not have to be initialized before entering the loop, the compiler will do that for you.
- The loop variable does not need to start with '1' , it can start with any integer number.
- The loop variable is automatically incremented. There is no need for a line that says : count := count +1.
- DO NOT alter the loop variable within the loop. This is a common error.
- The terminal count does not have to be a number, it can be a variable, say N, as long as it is defined before entering the loop!
- The loop variable may be decremented by one if downto is used instead of to in the for loop.

### 1.6.5 While .. Do..Statement

When the loop variable has to be changed by an amount other than one, or a test occurs within the loop, it is better to use the while .. do or repeat .. until statements.

As seen the for .. to .. do loop will execute a loop a set number of times. However, there are cases when the number of times around the loop is unknown, and another method of looping is needed. The While .. do loop can be used for this.

The format of the While .. do is: While <condition> do <statement>

While the condition is true, the statement (or statements within begin .. end) is executed.

An example will make this clearer.

```
while month = December do SingCarols;
```

### 1.6.6 Repeat .. Until

The final looping command is the repeat .. until.

The format of the command is: Repeat <statement> until <expression>

This is similar to the while do loop, except that the test is performed at the end of the loop. i.e. it will always perform the statement(s) once, before testing.

There are five methods for changing program flow and looping:

(if .. then .. else; case .. of ; for .. to .. do ; while .. do .. else ; and repeat .. until). Be careful of the use of semicolons in if .. then .. else statements. Do not change a variable in a loop.



## 1.7 Procedures and Functions

It will have been noticed by now that the events such as `Button1.Click` are called *procedures*. Functions provided by Delphi language such as `sqrt(x)`, `StrToInt(x)` and `round(x)` have also been encountered.

Procedures and functions are ways of breaking down programs into smaller and hence more manageable parts. Breaking the program up into smaller parts is good programming practice and not only helps in debugging the program, but also makes subsequent changes easier. This is known as 'modular' design. The smaller (debugging and working) procedures can be used in other programs, making those quicker and easier to write.

A procedure takes the form of:

```
Procedure name (parameters); declarations;  
begin  
  {procedure code goes here}  
end;
```

It is executed from another part of the program by using its name including any parameters in brackets.

### 1.7.1 Parameter Passing

The parameters can have a different name within the procedure. This makes procedures more versatile, the calling program can use the same routine with different parameters.

The variables declared in the procedure only exist while the procedure is being executed, and the memory used by the procedures is freed up when the procedure finishes.

Consider a procedure 'Getnumber' which requests a number from an edit box both passed as parameters. It can be called for a variety of purposes, for example to ask for age, date of birth, and salary, and may be used as follows:

```
Procedure Getnumber(var Number : integer; Edit1:TEdit);  
begin  
Number := StrToInt (edit1.text);end;
```

```
procedure TForm1.Button1Click(Sender: TObject);  
var salary, age, YearOfBirth : integer;  
begin  
Getnumber (salary, edit1);  
Getnumber (YearOfBirth, edit3);  
Edit1.text := 'You were born in ' + IntToStr(YearOfBirth) + ', you earn ' +  
IntToStr(salary) + ' ;  
end;
```

The procedure performs the same task every time (gets a number from an edit box) but returns the value into a different variable.

### 1.7.2 Functions

A function is a procedure that returns a single variable and when called is used on the right hand side of an expression. Consider the Sqrt( ) function used in the following program:

```
Procedure TForm1.Button1Click(Sender: TObject);  
var  
x:single;  
begin  
x:= strtofloat(edit1.text);  
edit1.text:= floattostr(sqrt (x)); { sqrt function used here }  
end;
```

The value  $x$  is passed to the `sqrt` function, returning the square root of  $x$ , which is converted (by the `floattostr` function) and placed in the edit box.

The following procedure 'Getnumber' returns a single value, and it can be coded as a function,

### 1.7.3 Recursion

Functions and procedures can call themselves, in which case they are known as *recursive*.

As an example of recursion, consider the a function which works out the factorial of a number. The factorial of a number is written as  $N!$ , and defined as  $0! = 1$ ,  $N! = N \times (N-1)!$  for  $N > 0$ . A function to perform this is shown below.

```
Function Factorial(number: integer):integer;
{variable ' number' is declared in the function declaration.}
{ The function returns an integer value }
begin
if number= 0 then factorial := 1 else
factorial := number * factorial(number-1);
{if number <>0 , call the function again }
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
var num , factor: integer;
begin
  num := StrToInt(edit1.text);
  factor := factorial (num);
  edit1.text := IntToStr (factor);
end;
```

### 1.7.4 Variable Declarations

The constants and variables used in the procedures have to be declared. If they are declared within the procedure, they only exist within that procedure and are known as *local variables*. If they are declared as part of the main program they are known as *global variables*. Functions return a single result, but could also alter global variables

## 1.8 Debugging



### 1.8.1 Program Errors

Sooner or later your programs will have a bug and either produce a run-time error, hang the computer, or just fail to work.

The compiler will detect any syntax and anything else it can't understand, but your program may compile but still fail due to a run-time error or a logic error. Typical run-time errors are divide by zero, edit box conversion error, or opening a non-existent file.

A dialog box is displayed when a run-time error occurs, with the line in error highlighted. The error can be examined in more detail using the debugger.

### 1.8.2 Using The Debugger

The debugger provides options to:

Run the program to a single point (Run To Cursor)

Run the program to specified points (known as breakpoints).



To view, disable, enable and delete breakpoints

To set conditions on breakpoint operation.

To pause the program.

To restart after meeting a breakpoint or pause.

To view data either when stopped or as the program runs.

To evaluate and modify variables.

### 1.8.3 Running The Program To A Breakpoint

The simplest way of running the program to a specific point is to select the line of code where the breakpoint is desired ( place the cursor on it) and then choose Run | Run to Cursor or use the F4 function key, or use the Speed Menu.

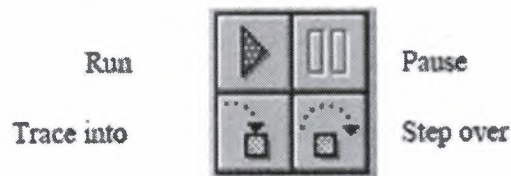


Figure 1.15 Breakpoint buttons

### 1.8.4 Setting Breakpoints

If more than one breakpoint is needed, then breakpoints must be set. To set the breakpoint click on the left of the line of code. The line will be highlighted in red. To display a SpeedMenu, either right-click the mouse in the desired window, or press Alt+F10 when the cursor is in the window.

The breakpoints must be on executable code; the breakpoint will not be met if set on a comment as they are not executed. Breakpoints can be set before or at run-time

## 1.8.5 Breakpoint Options

### 1.8.5.1 Viewing Breakpoints

A complicated debugging session may generate a lot of Breakpoints. When a Breakpoint is set, it is highlighted in the code editor, in addition to this if View | Breakpoint is chosen, a list of breakpoints is displayed.

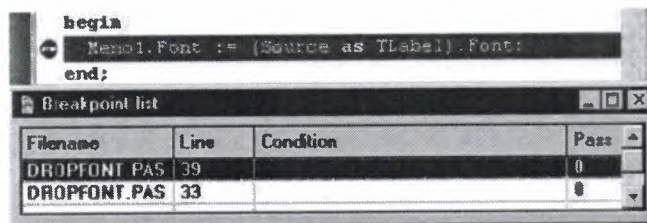
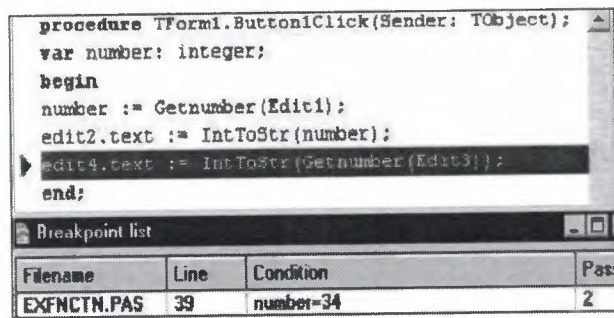


Figure 1.16 Break point view

### 1.8.5.2 Conditional Breakpoints

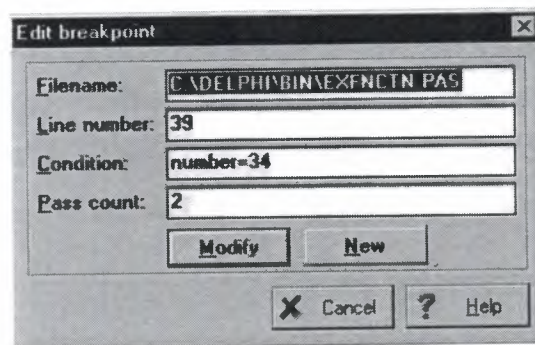
There will be times when it is not required to break every time a breakpoint is met. In particular if the breakpoint is in a loop, the breakpoint may only be needed at a certain value of loop counter. It is possible to put a condition on a breakpoint so that the program stops only when the condition is met.

It is also possible to put a pass count on the line of execution so that it executes that line that number of times before stopping. Both conditions can be set, in which case the break occurs only if both conditions are met. In the following example, the break occurs only when number = 34 and pass count =2; i.e. the second time this line is encountered with number = 34.



**Figure 1.17** Example of conditional break point

Having created the breakpoint, the condition and pass counts are set in the Edit Breakpoint dialog box.



**Figure 1.18** Breakpoint dialog box

The Edit Breakpoint box is displayed by using the Speed Menu from the Breakpoint List as above, or by double-clicking the required breakpoint in the list.

## 1.9 File I/O

Sooner or later, information will have to be read from or written to disk. A typical application is that of a database, but files have other uses as well; they can be used to hold hardware set-up information, or to store temporary data to relieve the pressure on memory. Data logging applications can easily collect a large amount of data and may have to be saved on disk before analyzed or transmitted elsewhere.

### 1.9.1 File Attributes

A file can have several attributes. The main file attributes are :

- Read-only
- Hidden
- System
- Archive

The attributes of a file are stored in a byte which can be read and written using FileGetAttr and FileSetAttr commands. The individual file attribute bits are available as: faReadOnly, faHidden, faSysFile, faArchive.

#### File Types

##### Text Files

Text files simply contain ASCII characters (those that can be typed from the keyboard) with lines terminated with Carriage Return (CR) and Line Feed characters (LF), the ASCII codes \$0D and \$0A. These files are normally created by simple editors. Most applications can use text files, and it is a good way of transferring data between applications.

### 1.9.2 File Operation

Table4 Common procedures for using typed, text files

AssignFile	Assigns a filename to a file variable before use.
Reset	Opens an existing file. Text files are read only.
Rewrite	Creates and opens a new file. Existing files with the same name will be overwritten.
Append	Opens an existing file for write only, so that new text can be added.
Readln	Reads a line ending with CR/LF from the file
Writeln	Writes text to the file and terminates with CR/LF.
CloseFile	Updates file,closes it.All files should be closed on terminating program.



### 1.9.3 Typed Files

Not all files contain text, or want to be read and amended in a sequential method. In this case they can be accessed in a random rather than sequential method. Such files consist of types such as integer or char, but may also be arrays or records.

#### 1.9.3.1 Typed file commands

Table5 Typed file commands

Procedure	Description
AssignFile	Assigns a filename to a file variable before use.
Reset	Opens an existing and assigned file.
Rewrite	Creates and opens a new file. Existing files with the same name will be overwritten.
Read	Reads record(s) from the file
Write	Writes record(s) to the file.
Seek	Moves the file's pointer to a specific record
Eof	Tests if current file position is at the end-of-file.
Closefile	Updates file and closes it. All files should be closed on terminating the program.

### 1.9.4. Untyped Files

Untyped files provide more flexibility in working with files, blocks of data rather than records are read from the disk into a memory area known as a buffer. The desired operations are then performed on the data, and the blocks are then written back to disk from the buffer. Pointers are used to determine the position within the file.

The procedures used are BlockRead , BlockWrite and FilePos:

The BlockRead command reads a number of bytes from the file assigned to 'Filename' into a buffer. The number of bytes read is reported in BytesRead, which will be the size of the buffer, unless eof is met.

```
BlockRead (Filename, Buffer, SizeOf(Buffer), BytesRead);
```

The position of the pointer is found using the FilePos command:

```
File_position := FilePos (Filename);
```

Data is written from the buffer to the file using the BlockWrite command:

```
BlockWrite (Filename, Buffer, NoOfBytesWritten);
```

### 1.9.5. File Handling Components

Delphi provides components to select files. These are the file handling components and are found on the System Palette. To see them in action, start a new project, and add a file list box, a drive combo box, a directory list box, and a filter combo box on the form. Following code for the OnChange event for the last three components.

```
procedure TForm1.DriveComboBox1Change(Sender: TObject);  
begin  
DirectoryListBox1.Drive := DriveComboBox1.Drive;  
end;
```

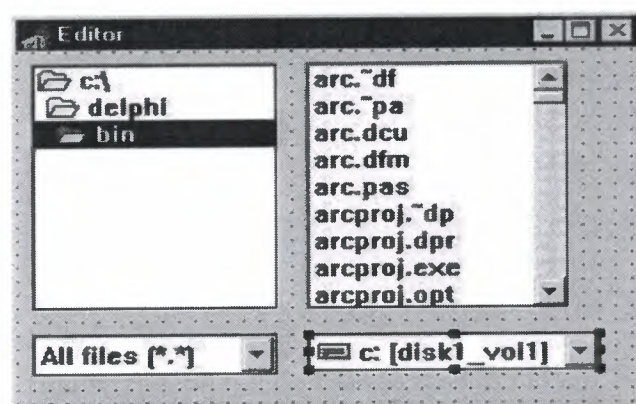
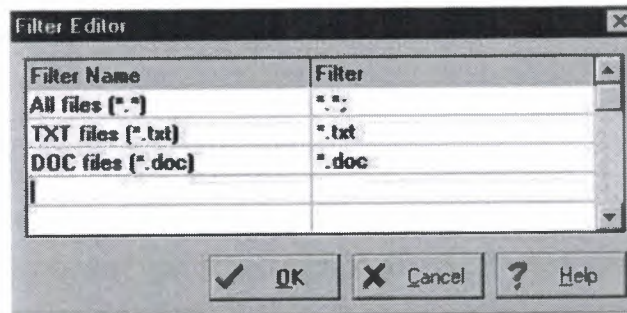


Figure 1.19 View of file componentet

The FilterComboBox only has the default filter 'All files (\*.\*)'. Change the filter properties so that .TXT and .DOC files alone can be selected. Click the three dots by the Filter property of the FilterComboBox to bring up a dialog box where the required files can be added. Add .TXT and .DOC to the list so that it looks like



**Figure 1.20** Filter editor

The *FileType* property of the FileListBox can be used to determine which files are displayed in the file list box based on the attributes of the files. Double-clicking FileType property in the Object Inspector displays the individual attributes to be displayed which can be set True or False.

## **CHAPTER 2**

### **2. DATABASE CONCEPT OF DELPHI 7**

#### **2.1 About Dbase And Paradox**

##### **2.1.1 dBASE IV Table Specification**

The dBASE IV table format was introduced in dBASE IV for DOS. Following are the specifications for dBASE IV tables:

- 2GB file size.
- Two billion records per file.
- A maximum of 255 fields per record.

Maintained indexes can have up to 47 indexes per file. Each index can be created using field expressions of virtually any combination, including conditional expressions of up to 255 characters per expression that result in an index of up to 100 bytes. Unlimited nonmaintained indexes can be stored on disk. You can use up to 47 of them simultaneously.

##### **2.1.2. dBase V Table Specifications**

The dBASE V table format was introduced in dBASE V for Windows. Following are the specifications for dBASE V tables.

- Up to one billion records per file.
- A maximum of 1,024 fields per record.
- Up to 32,767 bytes per record.
- Unlimited nonmaintained indexes can be stored on disk. You can use up to 47 of them simultaneously.



- Up to 10 master index files open per database. Each master index can have up to 47 indexes.
- Maintained indexes can have up to 47 indexes per file. Each index can be created using field expressions of virtually any combination, including conditional expressions of up to 255 characters per expression that result in an index of up to 100 bytes.

### **2.1.3. dBASE Field Types**

#### **Character (C)**

dBASE III+, IV, and V field type that can contain up to 254 characters (including blank spaces). This field is similar to the Paradox Alpha field type.

#### **Date (D)**

Paradox 3.5, 4, 5, and 7 as well as dBASE III+, IV, and V. dBASE tables can store dates from January 1, 100, to December 31, 9999. Paradox 5 tables can store from 12/31/9999 B.C. to 12/31/9999 A.D.

#### **Float (F)**

dBASE IV, and V floating-point numeric field type provides up to 20 significant digits.

#### **Logical (L)**

Paradox 5 and 7 and dBASE III+, IV, and V field type can store values representing True or False (yes or no). By default, valid entries include T and F (case is not important).

#### **Memo (M)**

Paradox 4, 5, and 7 as well as dBASE III+, IV, and V field. A Paradox field type is an Alpha variable-length field up to 256MB per field. dBASE Memo fields can contain binary as well as memo data.

### **OLE (O)**

Paradox 1, 5, and 7 as well as dBASE V field type that can store OLE data.

### **Number (N)**

Paradox 3.5, 4, 5, and 7 as well as dBASE III+, IV, and V field type can store up to 15 significant digits -10307 to + 10308 with up to 15 significant digits.

dBASE number fields contain numeric data in a Binary Coded Decimal (BCD) format. Use number fields when you need to perform precise calculations on the field data. Calculations on number fields are performed more slowly but with greater precision than are calculations on float number fields. The size of a dBASE number field can be from 1 to 20. Remember, however, that BCD is in Paradox 5 and 7 only for compatibility and is mapped directly to the Number field type.

### **Short (S)**

Paradox 3.5, 4, 5, and 7 field type that can contain integers from -- 32,767 through 32,767 (no decimal).

## **2.2. Paradox Standard Table Specifications**

**Also known as Paradox 4 table structure.**

The Paradox standard table format was introduced in Paradox for DOS version 4. Other products that use the standard format include Paradox for DOS version 4.5, ObjectVision 2.1, and Paradox for Windows versions 1.0 and 4.5.

Earlier versions of the Paradox table type are referred to as the Compatible table type. In the BDE Configuration Utility, the level option for the Paradox driver dictates what default table type is created by Paradox for Windows. Use 3 for Compatible tables, 4 for Standard tables (the default).

Following are the specifications for standard Paradox tables:

- 256MB file size limit if the table is in Paradox format and using a 4K block size.
- Up to 255 fields per record.
- Up to 64 validity checks per table.
- A primary index can have up to 16 fields.
- Tables can have up to 127 secondary indexes.
- Up to two billion records per file.

Because of the 256MB file size limit and other factors such as block size, however, the limit is much smaller. Tables of 190,000 records are easily achievable (and you can have more if you don't use up the 1,350-bytes-per-record limit for a keyed table). Tables with close to a million records are common.

Block size can be 1024, 2048, 3072, or 4096. Paradox stores data in fixed records. Even if part or all of the record is empty, the space is claimed. Knowing the interworkings can save you disk space. Paradox stores records in fixed blocks of 1024, 2048, 3072, 4096 in size. After a block size is set for a table, that size is fixed, and all blocks in the table will be of that size. To conserve disk space, you want to try to get your record size as close to a multiple of block size as possible (minus 6 bytes, which are used by Paradox to manage the table).

Record size. 1,350 for keyed tables and 4,000 for unkeyed tables. When figuring out the size (the number of bytes or characters) of a table, remember that Alpha fields take up their size (for example, an A10 = 10 bytes), numeric field types take up 8 bytes, short number field types take up 2 bytes, money takes up 8, and dates take up 4 bytes.

Memos, BLOBs, and so on take 10 bytes plus however much of the memo is stored in the .DB. For example, M15 takes 25 bytes.

### 2.2.1. Paradox 5 Table Specifications

The Paradox 5 table format was introduced in Paradox for Windows version 5.

Following are the specifications for Paradox 5 tables:

- Up to two billion records per file.
- File size is limited to two gigabytes.
- Up to 255 fields per record.

**Record size:** Up to 10,800 bytes per record for indexed tables and 32,750 bytes per record for nonindexed tables. When figuring out the size (the number of bytes or characters) of a table, remember that Alpha fields take up their size (for example, an A10 = 10 bytes), numeric field types take up 8 bytes, short number field types take up 2 bytes, money takes up 8, and dates take up 4 bytes.

Memos, BLOBs, and so on take 10 bytes plus however much of the memo is stored in the .DB. For example, M15 takes 25 bytes.

Up to 64 validity checks per table for Paradox for Windows tables.

A primary index can have up to 16 fields.

Tables can have up to 127 secondary indexes.

Block size can be from 1K to 32K in steps of 1K. For example, 1024, 2048, 3072, 4096, 5120...32768.

### 2.2.2. Paradox 7 and Above Table Specifications

The Paradox 7 table format was introduced in Paradox version 7 for Windows 95/NT. The Paradox 7 table format has all the same specifications as the Paradox 5 table format with two additions.

Following are the specification additions for the Paradox 7 table format:

- Added descending secondary indexes.
- Added unique secondary indexes



### **2.2.2.1. Paradox Field Types**

#### **Alpha (A)**

Paradox 3.5, 4, 5, and 7 field type that can contain up to 255 letters and numbers. This field type was called Alphanumeric in versions of Paradox before version 5. It is similar to the Character field type in dBASE.

#### **Autoincrement (+)**

Field type introduced in the Paradox 5 table format that adds one to the highest number in the table whenever a record is inserted. The starting range can from -2,147,483,647 to 2,147,483,647. Deleting a record does not change the field values of other records.

#### **BCD (#)**

Paradox 5 and 7 field type which is provided only for compatibility with other applications that use BCD data. Paradox correctly interprets BCD data from other applications that use the BCD type. When Paradox performs calculations on BCD data, it converts the data to the numeric float type, then converts the result back to BCD. When this field type is fully supported, it will support up to 32 significant digits.

#### **Binary (B)**

Paradox 1, 5, and 7 field type that can store binary data up to 256MB per field.

#### **Bytes (Y)**

Paradox 5 and 7 field type for storing binary data up to 255 bytes. Unlike binary fields, bytes fields are stored in the Paradox table (rather than in the separate .MB file), allowing for faster access.

#### **Date (D)**

Paradox 3.5, 4, 5, and 7 as well as dBASE III+, IV, and V. dBASE tables can store dates from January 1, 100, to December 31, 9999. Paradox 5 tables can store from 12/31/9999 B.C. to 12/31/9999 A.D.

### **Formatted Memo (F)**

Paradox 1, 4.5, 5, and 7 field type is like a memo field except that you can format the text. You can alter and store the text attributes of typeface, style, color, and size. This rich text document has a variable-length up to 256MB per field.

### **Graphic (G)**

Paradox 1, 5, and 7 field type can contain pictures in .BMP (up to 24 bit), .TIF (up to 256 color), .GIF (up to 256 color), .PCX, and .EPS file formats. Not all graphic variations are available. For example, currently you cannot store a 24-bit .TIF graphic. When you paste a graphic into a graphic field, Paradox converts the graphic into the .BMP format.

### **Logical (L)**

Paradox 5 and 7 and dBASE III+, IV, and V field type can store values representing True or False (yes or no). By default, valid entries include T and F (case is not important).

### **Memo (M)**

Paradox 4, 5, and 7 as well as dBASE III+, IV, and V field. A Paradox field type is an Alpha variable-length field up to 256MB per field. dBASE Memo fields can contain binary as well as memo data.

For Paradox tables, the file is divided into blocks of 512 characters. Each block is referenced by a sequential number, beginning at zero. Block 0 begins with a 4-byte number in hexadecimal format, in which the least significant byte comes first. This number specifies the number of the next available block. It is, in effect, a pointer to the end of the memo file. The remainder of Block 0 isn't used.

### **Money (\$)**

Paradox 3.5, 4, 5, and 7 field type, like number fields, can contain only numbers. They can hold positive or negative values. Paradox recognizes up to six decimal places when performing internal calculations on money fields. This field type was called Currency in previous versions of Paradox.

### **OLE (O)**

Paradox 1, 5, and 7 as well as dBASE V field type that can store OLE data.

### **Number (N)**

Paradox 3.5, 4, 5, and 7 as well as dBASE III+, IV, and V field type can store up to 15 significant digits -10307 to + 10308 with up to 15 significant digits.

dBASE number fields contain numeric data in a Binary Coded Decimal (BCD) format.

Use number fields when you need to perform precise calculations on the field data.

Calculations on number fields are performed more slowly but with greater precision than are calculations on float number fields. The size of a dBASE number field can be from 1 to 20. Remember, however, that BCD is in Paradox 5 and 7 only for compatibility and is mapped directly to the Number field type.

### **Short (S)**

Paradox 3.5, 4, 5, and 7 field type that can contain integers from -- 32,767 through 32,767 (no decimal).

### **Time (T)**

Paradox 5 and 7 field type that can contain time times of day, stored in milliseconds since midnight and limited to 24 hours.

This field type does not store duration which is the difference between two times. For example, if you need to store the duration of a song, use an Alpha field. Whenever you need to store time, make a distinction between clock time and duration. The Time field type is perfect for clock time. Duration can be stored in an Alpha field and manipulated with code.

### **TimeStamp (@)**

Paradox 5 field type comprised of both date and time values. Rules for this field type are the same as those for date fields and time fields.

## **CHAPTER 3**

### **3. DATABASE DESIGN OF THE PROGRAM**

#### **3.1 Database Design of The Program**

The specialized school info system database consists of five tables those are Student table, Teacher table, Login table, Payment\_Plan table, Student\_Attendance table.

**Student contains ten fields :**

- St\_number
- St\_name
- St\_surname
- Birthdate
- Parent\_name
- Class
- Department
- Address
- Tel\_number
- Reg\_date

**Attendance table contains five fields :**

- St\_number
- St\_name
- St\_surname
- Absence\_day
- Absence\_lesson



**Login table contains four fields :**

- User\_Id
- Uname
- Upassword
- Upass[Re]

**Payment\_Plan table contains eleven fields :**

- St\_number
- St\_name
- St\_surname
- Total\_pay
- Num\_instalment
- Instalment
- Day\_payment
- September
- December
- March
- May

**Teacher table contains nine fields :**

- Teacher\_Id
- T\_name
- T\_surname
- Social\_Id
- Branch
- Sex
- Address

The relationships between tables will as follows:

In Student Table St\_number field is a primary key.

In Student Table St\_name field is a primary key.

In Teacher Table Teacher\_Id is a primary key.

In Teacher Table T\_name is a primary key.

In Login Table Uname is a primary key.

In Login Table Upassword is a primary key.

In Payment\_Plan St\_number field is a primary key.

In Payment\_Plan St\_name field is a primary key.

In Attendance St\_number field is a primary key.

In Attendance St\_name field is a primary key.

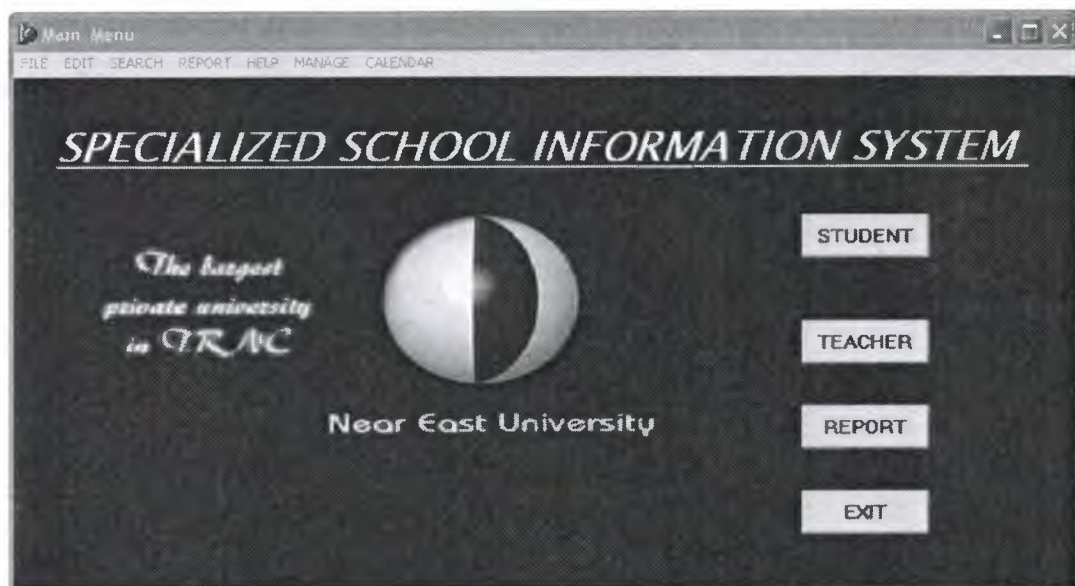
When you execute the program Login Form opens, then it will ask you username, password and password[Re]. You can see it in Fig 3.1.



**Fig 3.1** Login Form

If you do not know username, password and password[Re] you can not login this program. In fig3.1 write username, password and password[Re] then Click Login Button or Press Enter Key to login this program.

In this program there is File, Edit, Search, Report, Help, Manage, Calendar menus and for other applications Student, Teacher, Report, Exit buttons. (shown in Fig 3.2)



**Figure 3.2** Main Menu



When you select Edit Submenu then Student/Teacher add form is appeared

Student and Teacher Management

STUDENT TEACHER REPORT ABOUT

STUDENT TEACHER REPORT

Student Number: 20010713

Student Name: sevgi

Student Surname: bezginsoy

Birthdate: 27.08.1985

Parent Name: ismail

Select Class: Graduated

Select Department: Mathematic and Turki

Address: gaziantep

Telephone Number: (342)312-2096

Registration Date: 12.07.2001

Save

Search

List all Student

Attendance

Payment Plan

Close

Exit Program

**Figure 3.3** Student and Teacher Mangement form

In this form we can add new Student\Teacher to the database then it is shown in the Fig 3.3 on the form. And also we can search, list all student, student attendance ,close for turning before form and exit program to finish program.



When you select Search button then Search Student form is appeared.

St_number	St_name	St_surname	Class	Department
20010486	pelin	bezginsoy	Graduated	Science and Math

**Figure 3.4** Student Search form

We can search by number or by name of student . Also we can go next and prior by pressing next and previous button of the navigator. If we want to delete or update the Student which was added before after select on the table. Moreover we can delete or update student by pressing delete or update buttons which we want to do delete or update student.And when we press the Main menu it returns main menu form that is figure 3.2. and close for turning before form and exit program to finish program.



When you select Update button then Update Student form appeared.

**Update Student**

**Search by student number**

Student number

**Search**

**Search by name**

Student Name

**Search**

Student number

Student Name

Student Surname

Birthdate

Parent Name

Select Class

Select Department

Address

Telephone Number

Registration Date

**UPDATE**

**MAIN MENU** **CLOSE** **EXIT PROGRAM**

Figure 3.5 Update\_Student form

If we press UPDATE button we can save updated record of Student which was added before after select on the table.



When you select Delete button then Delete Student form appeared

**Delete Student**

**Search by student number**  
Student number   
**Search**

**Search by name**  
Student Name   
**Search**

Student number

Student Name

Student Surname

Birthdate

Parent Name

Select Class

Select Department

Address

Telephone Number

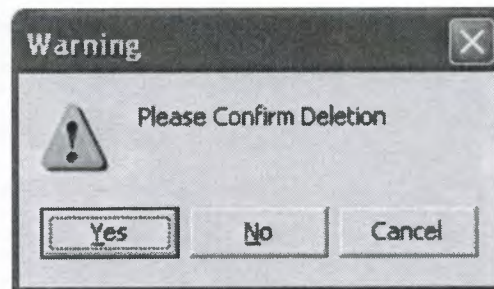
Registration Date

**DELETE**

**MAIN MENU** **CLOSE** **EXIT PROGRAM**

Figure 3.6 Delete\_Student form

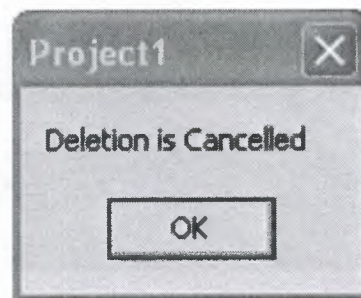
We can delete record of Student which was added before after select on the table. When you press the “DELETE” button you will see such as a “Please Confirm Deletion” message:



**Fig 3.7** Confirm Deletion Message

If you press “Yes” button you will get such as a “Deleted Successfully” message. Then the record is deleted from the database table.

If you press “No” button the deletion is cancelled



**Fig 3.8** Deletion Cancelled Message



When you select List All Student button then Search Student form appeared

St_number	St_name	St_surname	Class	Department
20010712	ayse	haskologlu	Graduated	Science and Math
20010713	sevgi	bezginsoy	Graduated	Mathematic and T
20020001	gizem	bezginsoy	Thrd Year	Science and Math
20021627	hanife	guney	Graduated	Turkish and Socia
20030002	seçil	belen	First Year	TURKISH AND SI
20040001	elif	bezginsoy	Thrd Year	Science and Math
20050001	basak	bezginsoy	Second Year	Science and Math

Figure 3.9 Student Search form

We can go next and prior by pressing next and previous button of the navigator. If we want to reach one student informations we can search by number or name of student.

When you select Payment Plan button then Student Payment Plan form appeared

Student Payment Plan

Student Number: 20020001

Student Name: elif

Student Surname: bezginsoy

Total Payment: 600

Number of Instalment: 1

Instalment: 600

Day of Instalment: 15

Select to pay month:

- ☒ September
- ☒ December
- ☒ March
- ☒ May

ADD SEARCH

EXIT PROGRAM MAIN MENU RECORD PAID MONTH

ST_NUMBER	ST_NAME	ST_SURNAME	TOTAL_PAY	NUM_INSTAL	INSTALMENT	DAY_PAYN
20010486	pelin	bezginsoy	1200	4	300	
* 20020001	elif	bezginsoy	600	1	600	
20010712	ayse	haskol	800	2	400	

Figure 3.10 Student Payment Plan form

In this form we can add new Student Payment Plan to the database then it is shown in the Fig 3.10 on the form. And also we can search, or record paid month of student payment plan.



When you select Search button then Update or Delete Payment Plan form appeared

Update or Delete Payment Plan

Search by St\_Number

ST\_Number

Search

Search by ST\_Name

ST\_Name

Search

Student Number

Student Name

Student Surname

Total Payment

Number of Instalment

Instalment

Day of Instalment

Select the paid month

☐ September

☒ December

☒ March

☐ May

Select to pay month

☒ September

☒ December

☒ March

☒ May

UPDATE

DELETE

MAIN MENU

CLOSE

EXIT PROGRAM

**Figure 3.11** Update or Delete Payment Plan form

With this form we can change paid or must to pay month by using checkboxes then if we press update button we can save updated record of Student Payment Plan which was added before after select on the table.

When you select Record Paid Month button then Record Paid Month form appeared

ST_NUMBER	ST_NAME	ST_SURNAME	TOTAL_PAY	NUM_INSTAL	INSTALMENT	DAY_PAYMEN	SEPTEMBER	DECEMBER
20010486	pelin	bezginsoy	1200	4	300	15	Instalment	Instalment
20010712	ayse	haskol	800	2	400	25	Instalment	

**Figure 3.12** Record Paid Month form

If we press UPDATE button we can save updated record which is changed of paid month text instalment to paid or any information of student which was added before after select on the table.



When you select Attendance button then Student Attendance form appeared

St_number	St_name	St_surname	Absence_day	Absence_lesson
20020753	esvet	çilikli	02.06.2006	Mathematic

Figure 3.13 Student Attendance form

In this form we can add new Attendance for student to the database then it is shown in the Fig 3.13 on the form. And also we can search by student number or name and update or delete the attendance information of student.

Also we can Add ,Delete, Update Teacher and List all teacher with other forms. When you select Time Table button that is located at Figure 3.3 Student and Teacher Mangement form on the teacher caption then Techer Time Table form appeared

Teacher Time Table

## TEACHERS TIME TABLE

	MONDAY	TUESDAY	WEDNESDA	THURSDAY	FRIDAY	SATURDAY	SUNDAY
9:30-10:20	Mathematic		Mathematic				Exams time
10:30-11:20	Mathematic	Mathematic		Goemetri	Mathematic		Exams time
11:30-12:20		Goemetri		Mathematic			Exams time
12:30-13:20	lunch time	lunch time	lunch time	lunch time	lunch time	lunch time	Exams time
13:30-14:20	Goemetri				Goemetri		Exams time
14:30-15:20				Goemetri	Mathematic		Exams time
15:30-16:20							Exams time

**Figure 3.14** Teacher Time Table Form

With Teacher Time Table Form we can learn time table of teachers and also we can save changed or new time table of teachers using the save button and we can look time tables of teacher by pressing open button.



When you select Salary Chart button that is located at Figure 3.3 Student and Teacher Mangement form on the teacher caption then Teacher Salary Chart form appeared

Teacher Name	Daily Wage	Day Count	Salary
ayse	120	45	5400
pelin	200	45	9000
nurgül	300	30	9000
selva	150	90	13500

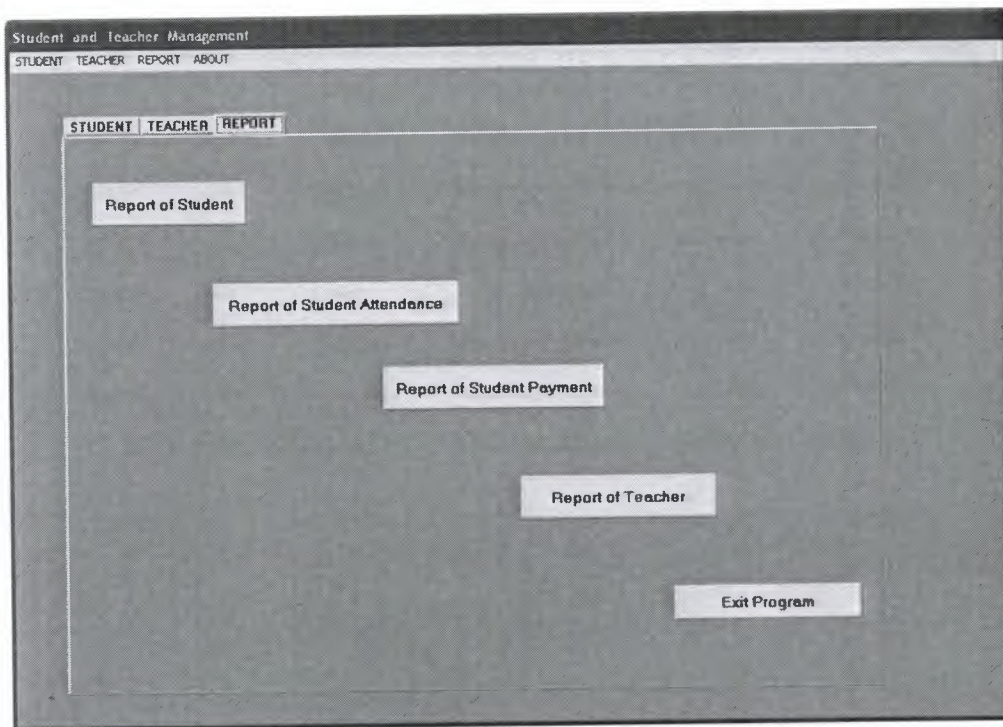
OPEN SAVE Main Menu EXIT PROGRAM

SUM:36900

**Figure 3.15** Teacher Salary Chart Form

With Teacher Salary Chart Form we can learn Salaries of teachers and also we can save changed or new Salaries of teachers using the save button and we can look Salary Chart of teacher by pressing open button. Behind this operations we can learn total payment of teacher which the specialized school must to pay the teachers by looking the 'SUM' caption.

In main menu form that is figure 3.2. if we choose report tab then we can select buttons which we want to see report of student, teacher, report of student payment plan or student attendance report.



If we select report of student in figure 3.2 in main menu or figure 3.3 student management form the report of student form appeared.

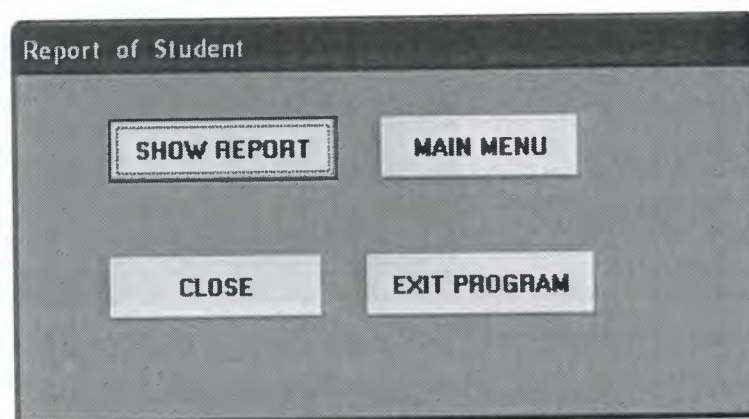
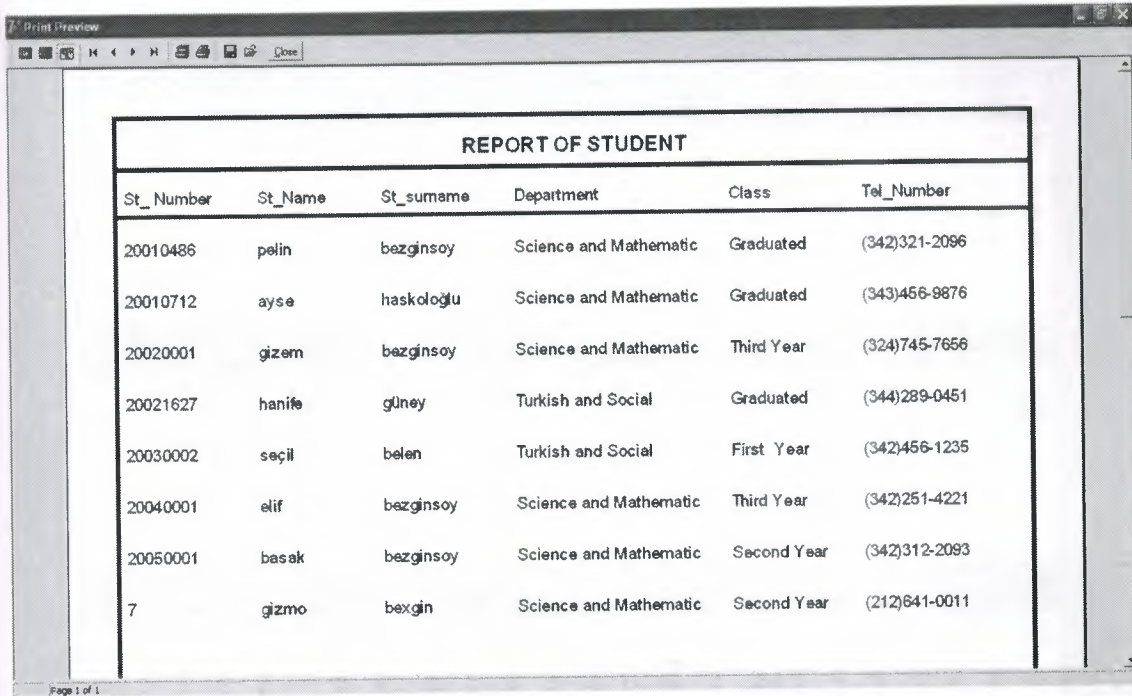


Figure 3.16 Report of Student form



When we select Show Report button the print preview form appeared.

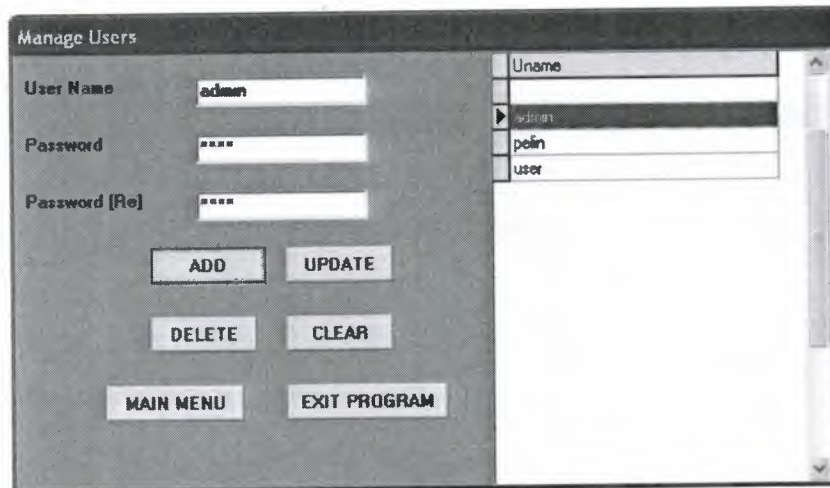


The screenshot shows a 'Print Preview' window with a table titled 'REPORT OF STUDENT'. The table has six columns: St\_Number, St\_Name, St\_surname, Department, Class, and Tel\_Number. It contains seven rows of student data.

St_Number	St_Name	St_surname	Department	Class	Tel_Number
20010486	polin	bezginsoy	Science and Mathematic	Graduated	(342)321-2096
20010712	ayse	haskologlu	Science and Mathematic	Graduated	(343)456-9876
20020001	gizem	bezginsoy	Science and Mathematic	Third Year	(324)745-7656
20021627	hanife	guney	Turkish and Social	Graduated	(344)289-0451
20030002	seçil	belen	Turkish and Social	First Year	(342)456-1235
20040001	elif	bezginsoy	Science and Mathematic	Third Year	(342)251-4221
20050001	basak	bezginsoy	Science and Mathematic	Second Year	(342)312-2093
7	gizmo	bexgin	Science and Mathematic	Second Year	(212)641-0011

**Figure 3.17** Print Preview form

If we select manage tab in figure 3.2 in main menu the manage user form appeared.



The screenshot shows the 'Manage Users' form. It has three input fields: 'User Name' (containing 'admin'), 'Password' (containing four asterisks), and 'Password [Re]' (containing four asterisks). Below these fields are six buttons: 'ADD', 'UPDATE', 'DELETE', 'CLEAR', 'MAIN MENU', and 'EXIT PROGRAM'. On the right side, there is a list box titled 'Uname' containing the following items: 'admin', 'polin', and 'user'. The 'admin' item is currently selected.

**Figure 3.18** Manage Users form

In "Manage User Form" we can add user, update user and delete user.

If we select Calendar tab in figure 3.2 in main menu the Calendar form appeared.

**Calendar**

Ocak 2006							Şubat 2006							Mart 2006									
Pzt	Sal	Çar	Per	Cum	Cmt	Paz	Pzt	Sal	Çar	Per	Cum	Cmt	Paz	Pzt	Sal	Çar	Per	Cum	Cmt	Paz			
52	25	26	27	28	29	30	1	5	6	7	8	9	10	11	12	9	1	2	3	4	5		
1	2	3	4	5	6	7	8	6	6	7	8	9	10	11	12	10	6	7	8	9	10	11	12
2	9	10	11	12	13	14	15	7	13	14	15	16	17	18	19	11	13	14	15	16	17	18	19
3	16	17	18	19	20	21	22	8	20	21	22	23	24	25	26	12	20	21	22	23	24	25	26
4	23	24	25	26	27	28	29	9	27	28						13	27	28	29	30	31		
5	30	31																					

Nisan 2006							Mayıs 2006							Haziran 2006									
Pzt	Sal	Çar	Per	Cum	Cmt	Paz	Pzt	Sal	Çar	Per	Cum	Cmt	Paz	Pzt	Sal	Çar	Per	Cum	Cmt	Paz			
13					1	2	18	1	2	3	4	5	6	7	22			1	2	3	4	5	
14	3	4	5	6	7	8	9	19	8	9	10	11	12	13	14	23	5	6	7	8	9	10	11
15	10	11	12	13	14	15	16	20	15	16	17	18	19	20	21	24	12	13	14	15	16	17	18
16	17	18	19	20	21	22	23	21	22	23	24	25	26	27	28	25	19	20	21	22	23	24	25
17	24	25	26	27	28	29	30	22	29	30	31					26	26	27	28	29	30		

Temmuz 2006							Ağustos 2006							Eylül 2006									
Pzt	Sal	Çar	Per	Cum	Cmt	Paz	Pzt	Sal	Çar	Per	Cum	Cmt	Paz	Pzt	Sal	Çar	Per	Cum	Cmt	Paz			
26					1	2	31	1	2	3	4	5	6	35			1	2	3				
27	3	4	5	6	7	8	9	32	7	8	9	10	11	12	13	36	4	5	6	7	8	9	10
28	10	11	12	13	14	15	16	33	14	15	16	17	18	19	20	37	11	12	13	14	15	16	17
29	17	18	19	20	21	22	23	34	21	22	23	24	25	26	27	38	18	19	20	21	22	23	24
30	24	25	26	27	28	29	30	35	28	29	30	31				39	25	26	27	28	29	30	
31	31																						

Ekim 2006							Kasım 2006							Aralık 2006									
Pzt	Sal	Çar	Per	Cum	Cmt	Paz	Pzt	Sal	Çar	Per	Cum	Cmt	Paz	Pzt	Sal	Çar	Per	Cum	Cmt	Paz			
39					1	8	44	1	2	3	4	5	48			1	2	3					
40	2	3	4	5	6	7	8	45	6	7	8	9	10	11	12	49	4	5	6	7	8	9	10
41	9	10	11	12	13	14	15	46	13	14	15	16	17	18	19	50	11	12	13	14	15	16	17
42	16	17	18	19	20	21	22	47	20	21	22	23	24	25	26	51	18	19	20	21	22	23	24
43	23	24	25	26	27	28	29	48	27	28	29	30				52	25	26	27	28	29	30	31
44	30	31															1	2	3	4	5	6	7

Bugün: 04.06.2006

Figure 3.19 Calendar

In Calendar Form there is shown the Calendar as we see above Fig 3.19. We can see the date or specific date which we want to learn.

If we select Help tab in figure 3.2 in main menu the Help form appeared. We can learn information about program that is how we can use program



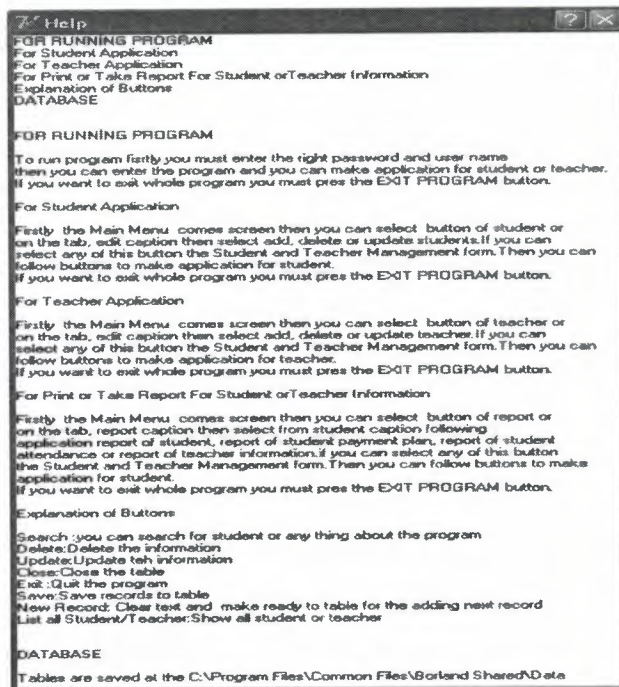


Figure 3.20 Help form

If we select Help tab in figure 3.2 in main menu then from sub menu if we select about then sub menu of about we can select program or programmer the Help form appeared.

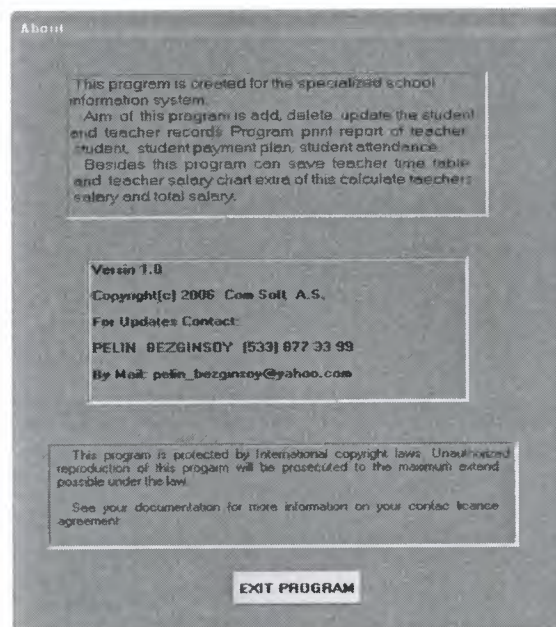


Figure 3.21 About form

Gives the user a brief description about the programmer.

## CONCLUSION

A DBMS is computerized record keeping system that stores, maintains and provides access to the information. A Database system consists of four major components that are Data, Hardware, Software, and Users. DBMS are used by any reasonably self contained commercial, scientific, technical or other organization for a single individual to a large company. Practically implementation of software for business though it is related to any field needs a devoted and complete life cycle.

The software created after a deep analysis, so that all-important requirements to the specialized school information system with student and teacher information can be accomplished. Student and teacher, name and ID have been included in the program to overcome the errors, which can occur. Reports are also generated with the help of the Queries for the update purpose.



## **REFERENCES**

### **Reference to Book:**

[1] Turkmen Kitabevi, Borland Delphi 7

### **Reference to Electronic-Book:**

[1] Manchester Metropolitan University Advance Software Engineering Delphi .PDF

### **Reference to Electronic Source- Online source from Web:**

[1] <http://www.Delphi Basics Object Orientation .com>

### **Reference to Book:**

[1] Seçkin Kitabevi, Borland Delphi 8

## APPENDIX A: PROGRAM CODES

### FORM 1. STUDENT AND TEACHER MANAGEMENT FORM

```
unit Unit1;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, Buttons, ComCtrls, ExtCtrls, Mask, DBCtrls, Grids,  
DBGrids, DB, DBTables, Menus;
```

```
type
```

```
TStudent_Teacher_Management = class(TForm)
```

```
    PageControl1: TPageControl;
```

```
    TabSheet1: TTabSheet;
```

```
    TabSheet2: TTabSheet;
```

```
    TabSheet3: TTabSheet;
```

```
    Label1: TLabel;
```

```
    Label2: TLabel;
```

```
    Label3: TLabel;
```

```
    Label4: TLabel;
```

```
    Label5: TLabel;
```

```
    Label6: TLabel;
```

```
    Label7: TLabel;
```

```
    Label8: TLabel;
```

```
    Label9: TLabel;
```

```
    Label10: TLabel;
```

```
    BitBtn1: TBitBtn;
```

```
    Label12: TLabel;
```

```
    Label13: TLabel;
```

```
    Label14: TLabel;
```

```
    Label15: TLabel;
```

```
    Label16: TLabel;
```

```
    Label17: TLabel;
```

Label18: TLabel;  
Label19: TLabel;  
Label20: TLabel;  
BitBtn5: TBitBtn;  
MainMenu1: TMainMenu;  
Student1: TMenuItem;  
LSTALLSTUDENT1: TMenuItem;  
PAYMENTPLAN1: TMenuItem;  
ATTENDANCE1: TMenuItem;  
EXIT1: TMenuItem;  
EACHER1: TMenuItem;  
IMETABLE1: TMenuItem;  
SALARYCHART1: TMenuItem;  
REPORT1: TMenuItem;  
BitBtn12: TBitBtn;  
BitBtn13: TBitBtn;  
BitBtn14: TBitBtn;  
BitBtn15: TBitBtn;  
REPORTOFSTUDENT1: TMenuItem;  
REPORTOFTEACHER1: TMenuItem;  
REPORTOFSTUDENTATTENDANCE1: TMenuItem;  
REPORTOFSTUDENTPAYMENT1: TMenuItem;  
HELP1: TMenuItem;  
ABOUT1: TMenuItem;  
PROGRAM1: TMenuItem;  
PROGRAM2: TMenuItem;  
Panel1: TPanel;  
SpeedButton3: TSpeedButton;  
BitBtn3: TBitBtn;  
BitBtn4: TBitBtn;  
BitBtn2: TBitBtn;  
BitBtn9: TBitBtn;  
BitBtn8: TBitBtn;

Panel2: TPanel;  
BitBtn16: TBitBtn;  
BitBtn7: TBitBtn;  
BitBtn6: TBitBtn;  
BitBtn17: TBitBtn;  
BitBtn18: TBitBtn;  
DataSource1: TDataSource;  
Student: TTable;  
Table2: TTable;  
DataSource2: TDataSource;  
MaskEdit1: TMaskEdit;  
MaskEdit2: TMaskEdit;  
MaskEdit3: TMaskEdit;  
MaskEdit4: TMaskEdit;  
MaskEdit5: TMaskEdit;  
MaskEdit6: TMaskEdit;  
MaskEdit7: TMaskEdit;  
Edit1: TEdit;  
ComboBox1: TComboBox;  
ComboBox2: TComboBox;  
Edit2: TEdit;  
Edit3: TEdit;  
Edit4: TEdit;  
Memo1: TMemo;  
ComboBox3: TComboBox;  
ComboBox4: TComboBox;  
Memo2: TMemo;  
Edit5: TEdit;  
Edit6: TEdit;  
procedure BitBtn8Click(Sender: TObject);  
procedure BitBtn9Click(Sender: TObject);  
procedure BitBtn5Click(Sender: TObject);  
procedure BitBtn1Click(Sender: TObject);



```

procedure EXIT1Click(Sender: TObject);
procedure DBEdit1KeyPress(Sender: TObject; var Key: Char);
procedure LSTALLSTUDENT1Click(Sender: TObject);
procedure BitBtn6Click(Sender: TObject);
procedure BitBtn7Click(Sender: TObject);
procedure PROGRAM1Click(Sender: TObject);
procedure PROGRAM2Click(Sender: TObject);
procedure IMETABLE1Click(Sender: TObject);
procedure SALARYCHART1Click(Sender: TObject);
procedure BitBtn12Click(Sender: TObject);
procedure BitBtn13Click(Sender: TObject);
procedure SpeedButton3Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure BitBtn16Click(Sender: TObject);
procedure BitBtn17Click(Sender: TObject);
procedure BitBtn18Click(Sender: TObject);
procedure BitBtn4Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure BitBtn3Click(Sender: TObject);
procedure ATTENDANCE1Click(Sender: TObject);
procedure BitBtn14Click(Sender: TObject);

```

```
private
```

```
    { Private declarations }
```

```
public
```

```
    { Public declarations }
```

```
end;
```

```
var
```

```
    Student_Teacher_Management: TStudent_Teacher_Management;
```

```
implementation
```

```
uses st_list, Unit3, Unit9, Unit10, Unit11, Unit12, Unit14, Unit15, Unit19,  
Unit18;  
{$R *.dfm}
```

```
procedure TStudent_Teacher_Management.BitBtn8Click(Sender: TObject);  
begin  
application.Terminate;  
end;
```

```
procedure TStudent_Teacher_Management.BitBtn9Click(Sender: TObject);  
begin  
Student.Close;  
close;  
end;
```

```
procedure TStudent_Teacher_Management.BitBtn5Click(Sender: TObject);  
begin  
Table2.insert;  
Table2.FieldValues['Teacher_ID']:=MaskEdit5.Text;  
Table2.FieldValues['T_name']:=Edit5.Text;  
Table2.FieldValues['T_surname']:=Edit6.Text;  
Table2.FieldValues['Social_ID']:=MaskEdit4.Text;  
Table2.FieldValues['Branch']:=Combobox3.Text;  
Table2.FieldValues['Sex']:=Combobox4.Text;  
Table2.FieldValues['Address']:=Memo1.Text;  
Table2.FieldValues['Mobile_phone']:=MaskEdit6.Text;  
Table2.FieldValues['Home_phone']:=MaskEdit7.Text;  
Table2.Post;  
Edit5.Clear;  
Edit6.Clear;  
Memo1.Clear;  
MaskEdit4.Clear;  
MaskEdit5.Clear;
```

```

MaskEdit6.Clear;
MaskEdit7.Clear;
MaskEdit5.SetFocus;
end;

```

```

procedure TStudent_Teacher_Management.BitBtn1Click(Sender: TObject);
begin
Student.insert;
Student.FieldValues['St_number']:=Edit1.Text;
Student.FieldValues['St_name']:=Edit2.Text;
Student.FieldValues['St_surname']:=Edit3.Text;
Student.FieldValues['Birthdate']:=MaskEdit1.Text;
Student.FieldValues['Parent_name']:=Edit4.Text;
Student.FieldValues['Class']:=Combobox1.Text;
Student.FieldValues['Department']:=Combobox2.Text;
Student.FieldValues['Address']:=Memo1.Text;
Student.FieldValues['Tel_number']:=MaskEdit2.Text;
Student.FieldValues['Reg_date']:=MaskEdit3.Text;
Student.Post;
MaskEdit1.Clear;
MaskEdit2.Clear;
MaskEdit3.Clear;
Edit1.Clear;
Edit2.Clear;
Edit3.Clear;
Edit4.Clear;
Memo1.Clear;
Edit1.SetFocus;
end;

```

```

procedure TStudent_Teacher_Management.EXIT1Click(Sender: TObject);

```



```

begin
Student.Close;
table2.Close;
close;
end;

procedure TStudent_Teacher_Management.DBEdit1KeyPress(Sender: TObject; var
Key: Char);
begin
if(key<'0')or(key>'9')then
begin
key:=#0;
beep;
end;
end;

procedure TStudent_Teacher_Management.LSTALLSTUDENT1Click(Sender:
TObject);
begin
Src_Student.Show;
end;

procedure TStudent_Teacher_Management.BitBtn6Click(Sender: TObject);
begin
Src_Teacher.show;
end;

procedure TStudent_Teacher_Management.BitBtn7Click(Sender: TObject);
begin
Teacher_Time_Table.show;
end;

```

```
procedure TStudent_Teacher_Management.PROGRAM1Click(Sender: TObject);  
begin  
About_p.show;  
end;
```

```
procedure TStudent_Teacher_Management.PROGRAM2Click(Sender: TObject);  
begin  
About_p.show;  
end;
```

```
procedure TStudent_Teacher_Management.IMETABLE1Click(Sender: TObject);  
begin  
Teacher_Time_Table.show;  
end;
```

```
procedure TStudent_Teacher_Management.SALARYCHART1Click(Sender: TObject);  
begin  
Teacher_Salary_Chart.show;  
end;
```

```
procedure TStudent_Teacher_Management.BitBtn12Click(Sender: TObject);  
begin  
Report_Student.show;  
end;
```

```
procedure TStudent_Teacher_Management.BitBtn13Click(Sender: TObject);  
begin  
Report_Teacher.show;  
end;
```

```
procedure TStudent_Teacher_Management.SpeedButton3Click(Sender: TObject);  
begin  
Src_Student.Show;  
end;
```

```
procedure TStudent_Teacher_Management.BitBtn2Click(Sender: TObject);  
begin  
  Src_Student.Show;  
  Student_Teacher_Management.Hide;  
end;
```

```
procedure TStudent_Teacher_Management.BitBtn16Click(Sender: TObject);  
begin  
  Teacher_Salary_Chart.show;  
end;
```

```
procedure TStudent_Teacher_Management.BitBtn17Click(Sender: TObject);  
begin  
  application.Terminate;  
end;
```

```
procedure TStudent_Teacher_Management.BitBtn18Click(Sender: TObject);  
begin  
  application.Terminate;  
end;
```

```
procedure TStudent_Teacher_Management.BitBtn4Click(Sender: TObject);  
begin  
  Student_Payment_Plan.show;  
end;
```

```
procedure TStudent_Teacher_Management.FormCreate(Sender: TObject);  
begin  
  Student_Payment_Plan.show;  
end;
```



```
procedure TStudent_Teacher_Management.FormClose(Sender: TObject; var Action:
CloseAction);
```

```
begin
Student.Close;
table2.Close;
action:=cafree;
end;
```

```
procedure TStudent_Teacher_Management.BitBtn3Click(Sender: TObject);
begin
Student_Attendance.show;
end;
```

```
procedure TStudent_Teacher_Management.ATTENDANCE1Click(Sender: TObject);
begin
Student_Attendance.show;
end;
```

```
procedure TStudent_Teacher_Management.BitBtn14Click(Sender: TObject);
begin
Report_Student_Payment_Plan.show;
end;
end.
```

## **FORM 6. DELETE STUDENT FORM**

```
unit Unit6;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, DB, DBTables, StdCtrls, Buttons, DBCtrls, Mask, ExtCtrls;
```

```
type
```

```
TDelete_Student = class(TForm)
```

```
Label1: TLabel;
```

Label2: TLabel;  
Label3: TLabel;  
Label4: TLabel;  
Label5: TLabel;  
Label6: TLabel;  
Label7: TLabel;  
Label8: TLabel;  
Label9: TLabel;  
Label10: TLabel;  
DBEdit1: TDBEdit;  
DBEdit2: TDBEdit;  
DBEdit3: TDBEdit;  
DBEdit4: TDBEdit;  
DBEdit5: TDBEdit;  
DBEdit6: TDBEdit;  
DBEdit7: TDBEdit;  
DBMemo1: TDBMemo;  
DBComboBox1: TDBComboBox;  
DBComboBox2: TDBComboBox;  
GroupBox1: TGroupBox;  
Label21: TLabel;  
BitBtn16: TBitBtn;  
Edit1: TEdit;  
BitBtn1: TBitBtn;  
BitBtn2: TBitBtn;  
GroupBox2: TGroupBox;  
Label11: TLabel;  
Edit2: TEdit;  
BitBtn4: TBitBtn;  
SpeedButton1: TSpeedButton;  
SpeedButton3: TSpeedButton;  
DataSource1: TDataSource;  
Table1: TTable;

```

procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure BitBtn16Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure BitBtn4Click(Sender: TObject);
procedure SpeedButton1Click(Sender: TObject);
procedure SpeedButton3Click(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure FormCreate(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Delete_Student: TDelete_Student;
implementation
uses Unit1, Unit5;
{$R *.dfm}

procedure TDelete_Student.FormKeyPress(Sender: TObject; var Key: Char);
begin
    dbedit1.Clear;
    dbedit2.Clear;
    dbedit3.Clear;
    dbedit4.Clear;
    dbedit5.Clear;
    dbedit6.Clear;
    dbedit7.Clear;
    dbmemo1.Clear;
end;

```



```

procedure TDelete_Student.BitBtn16Click(Sender: TObject);
begin
table1.filter:='St_number='+ quotedstr(Edit1.Text );
table1.Filtered:=true;
if table1.RecordCount =0 then
    showmessage('No Record Found')
else
    Table1.GotoKey;
    Edit1.Clear;
    Edit1.SetFocus;
end;

```

```

procedure TDelete_Student.BitBtn1Click(Sender: TObject);
begin
Delete_Student.Hide;
Main_Menu.show;
end;

```

```

procedure TDelete_Student.BitBtn2Click(Sender: TObject);
begin
table1.Close;
close;
end;

```

```

procedure TDelete_Student.BitBtn4Click(Sender: TObject);
begin
table1.filter:='St_name='+ quotedstr(Edit2.Text );
table1.Filtered:=true;
if table1.RecordCount =0 then
    showmessage('No Record Found')
else
    Table1.GotoKey;
    Edit2.Clear;

```

```
Edit2.SetFocus;
```

```
end;
```

```
procedure TDelete_Student.SpeedButton1Click(Sender: TObject);
```

```
begin
```

```
if MessageDlg('Please Confirm Deletion',mtwarning, [mbYes, mbNo,mbCancel],0)=6
```

```
then
```

```
begin
```

```
//if accepts delete record
```

```
table1.Delete;
```

```
//list and refresh the product comboboxes
```

```
dbcombobox1.OnClick(sender);
```

```
dbcombobox2.OnClick(sender);
```

```
table1.Close;
```

```
table1.Open;
```

```
dbedit1.Clear;
```

```
dbedit2.Clear;
```

```
dbedit3.Clear;
```

```
dbedit4.Clear;
```

```
dbedit5.Clear;
```

```
dbedit6.Clear;
```

```
dbedit7.Clear;
```

```
Showmessage('Deletion is successfull');
```

```
end
```

```
else
```

```
Showmessage('Deletion is Cancelled');
```

```
end;
```

```
procedure TDelete_Student.SpeedButton3Click(Sender: TObject);
```

```
begin
```

```
application.Terminate;
```

```
end;
```

```

procedure TDelete_Student.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    table1.Close;
    action:=cafree;
end;
end.

```

## **FORM 16. RECORD THE PAID MONTH FORM**

```

unit Unit16;
interface
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, Grids, DBGrids, StdCtrls, ExtCtrls, DBCtrls, DB, DBTables, Mask,
    Buttons;
type
    TRecord_Payed_Month = class(TForm)
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        Label4: TLabel;
        Label5: TLabel;
        Label6: TLabel;
        DBEdit1: TDBEdit;
        DBEdit2: TDBEdit;
        DBEdit3: TDBEdit;
        DBEdit4: TDBEdit;
        DBEdit5: TDBEdit;
        DBEdit6: TDBEdit;
        DataSource1: TDataSource;
        Table1: TTable;
        DBGrid1: TDBGrid;
        BitBtn1: TBitBtn;
        GroupBox1: TGroupBox;
    end;

```



```

GroupBox2: TGroupBox;
DBEdit7: TDBEdit;
DBEdit8: TDBEdit;
Label7: TLabel;
Label8: TLabel;
BitBtn2: TBitBtn;
BitBtn3: TBitBtn;
BitBtn4: TBitBtn;
GroupBox3: TGroupBox;
DBCheckBox1: TDBCheckBox;
DBCheckBox2: TDBCheckBox;
DBCheckBox3: TDBCheckBox;
DBCheckBox4: TDBCheckBox;
procedure BitBtn2Click(Sender: TObject);
procedure BitBtn3Click(Sender: TObject);
procedure BitBtn4Click(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure FormCreate(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;
var
    Record_Payed_Month: TRecord_Payed_Month;
implementation
    {$R *.dfm}

    procedure TRecord_Payed_Month.BitBtn2Click(Sender: TObject);
    begin
        table1.IndexFieldNames:='ST_NUMBER';
        table1.SetKey;
        table1.IndexFields[0].AsString:=DbEdit1.Text;
    end;

```

```

table1.GotoKey;
if not table1.GotoKey then
showmessage('The Record is not Found');
end;

```

```

procedure TRecord_Payed_Month.BitBtn3Click(Sender: TObject);
begin
table1.IndexFieldNames:='ST_NAME';
table1.SetKey;
table1.IndexFields[0].AsString:=DbEdit1.Text;
table1.GotoKey;
if not table1.GotoKey then
showmessage('The Record is not Found');
end;

```

```

procedure TRecord_Payed_Month.BitBtn4Click(Sender: TObject);
begin
application.Terminate;
end;

```

```

procedure TRecord_Payed_Month.FormClose(Sender: TObject; var Action: TCloseAction);
begin
table1.Close;
action:=cafree;
end;
end.

```

## **FORM 10. TEACHER SALARY CHART FORM**

```

unit Unit10;
interface
uses

```

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, Grids, StdCtrls, Buttons;

type

TTeacher\_Salary\_Chart = class(TForm)

StringGrid1: TStringGrid;

Label1: TLabel;

Label2: TLabel;

BitBtn1: TBitBtn;

BitBtn2: TBitBtn;

BitBtn3: TBitBtn;

BitBtn4: TBitBtn;

procedure FormCreate(Sender: TObject);

procedure StringGrid1 SetEditText(Sender: TObject; ACol, ARow: Integer;  
const Value: String);

procedure BitBtn1Click(Sender: TObject);

procedure BitBtn2Click(Sender: TObject);

procedure BitBtn3Click(Sender: TObject);

procedure BitBtn4Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Teacher\_Salary\_Chart: TTeacher\_Salary\_Chart;

implementation

uses Unit5;

{ \$R \*.dfm }

procedure TTeacher\_Salary\_Chart.FormCreate(Sender: TObject);

begin

with stringGrid1 do

begin



```

options:=options+[goediting];
cells[0,0]:='Teacher Name';
cells[1,0]:='Daily Wage';
cells[2,0]:='Day Count';
cells[3,0]:='Salary';
end;
end;

```

```

procedure TTeacher_Salary_Chart.StringGrid1SetEditText(Sender: TObject; ACol,
  ARow: Integer; const Value: String);
var
  dwage,day,i,sum:integer;
begin
  dwage:=strtointdef(stringGrid1.Cells[1,ARow],0);
  day:=strtointdef(stringGrid1.Cells[2,ARow],0);
  stringGrid1.Cells[3,ARow]:=inttostr(dwage*day);
  sum:=0;
  for i:=1 to stringGrid1.RowCount do
    sum:=sum+strtointdef(stringGrid1.Cells[3,i],0);
  label2.Caption:='SUM:'+inttostr(sum);
end;

```

```

procedure TTeacher_Salary_Chart.BitBtn1Click(Sender: TObject);
var f:textfile;
t,x,y:integer;
tstr:string;
begin
  assignfile(f,'salary.dat');
  reset(f);
  readln(f,t);
  stringGrid1.ColCount:=t;
  readln(f,t);
  stringGrid1.RowCount:=t;

```

```

for x:=0 to stringGrid1.ColCount-1 do
for y:=0 to stringGrid1.RowCount-1 do
begin
readln(F,tstr);
stringGrid1.Cells[x,y]:=tstr;
end;
closefile(f);
end;

```

```

procedure TTeacher_Salary_Chart.BitBtn2Click(Sender: TObject);
var f:textfile;
x,y:integer;
begin
assignfile(f,'salary.dat');
rewrite(f);
writeln(f,stringGrid1.colcount);
writeln(f,stringGrid1.rowcount);
for x:=0 to stringGrid1.ColCount-1 do
for y:=0 to stringGrid1.RowCount-1 do
writeln(F,stringGrid1.cells[x,y]);
closefile(f);
end;

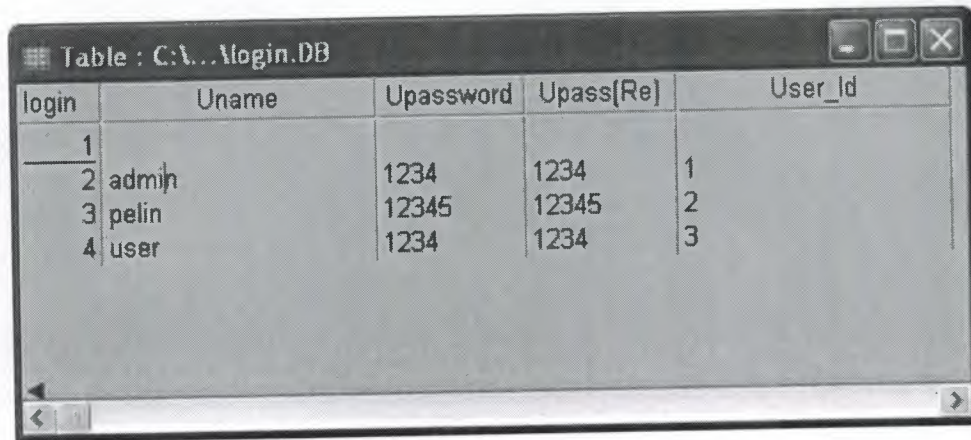
```

```

procedure TTeacher_Salary_Chart.BitBtn3Click(Sender: TObject);
begin
Teacher_Salary_Chart.Hide;
Main_Menu.show;
end;
procedure TTeacher_Salary_Chart.BitBtn4Click(Sender: TObject);
begin
application.Terminate;
end;
end.

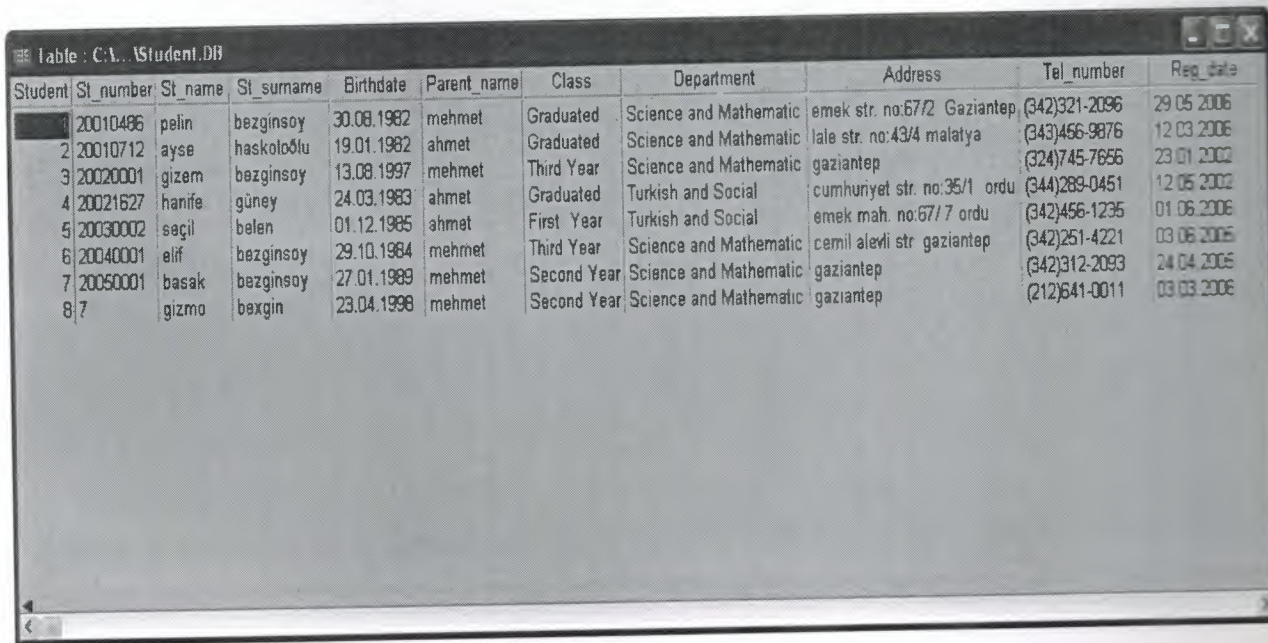
```

## APPENDIX B: DATABASE TABLES



login	Uname	Upassword	Upass(Re)	User_Id
1				
2	admin	1234	1234	1
3	pelin	12345	12345	2
4	user	1234	1234	3

Fig. 4.1 Login Table



Student	St_number	St_name	St_surname	Birthdate	Parent_name	Class	Department	Address	Tel_number	Reg_date
1	20010486	pelin	bezginsoy	30.08.1982	mehmet	Graduated	Science and Mathematic	emek str. no:67/2 Gaziantep	(342)321-2096	29.05.2006
2	20010712	ayse	haskoloğlu	19.01.1982	ahmet	Graduated	Science and Mathematic	lale str. no:43/4 malatya	(343)456-9876	12.03.2006
3	20020001	gizem	bezginsoy	13.08.1997	mehmet	Third Year	Science and Mathematic	gaziantep	(324)745-7856	23.01.2002
4	20021627	hanife	güney	24.03.1983	ahmet	Graduated	Turkish and Social	cumhuriyet str. no:35/1 ordu	(344)289-0451	12.05.2002
5	20030002	saçil	belen	01.12.1985	ahmet	First Year	Turkish and Social	emek mah. no:67/7 ordu	(342)456-1235	01.06.2006
6	20040001	elif	bezginsoy	29.10.1984	mehmet	Third Year	Science and Mathematic	camil alewi str gaziantep	(342)251-4221	03.06.2005
7	20050001	basak	bezginsoy	27.01.1989	mehmet	Second Year	Science and Mathematic	gaziantep	(342)312-2093	24.04.2005
8	7	gizmo	bexgin	23.04.1998	mehmet	Second Year	Science and Mathematic	gaziantep	(212)641-0011	03.03.2006

Fig. 4.2 Student Table



Teacher_ID	T_name	T_surname	Social_ID	Branch	Sex	Address	Mobile_phone	Home_phone
322-67-78-00	esret	çiftiklı	444-01-7310	Turkish	female	emek str no:57/4	(533)899-7788	(344)338-5498
366-68-45-93	yilmaz	yumbul	753-24-5693	geography	male	atatürk str. no:11/2	(505)298-1936	(452)714-0686
378-66-44-19	ibrahim	çolak	567-19-4597	biology	male	kurtulus str. no:54/9	(555)489-9009	(212)697-3871
466-67-49-21	hanife	güney	564-66-3981	Turkish	female	cumhuriyet str.no:52/3	(533)877-3399	(233)345-1205
539-25-35-67	elif	çolak	547-24-2682	physics	female	kurtulus str. no:54/9	(544)877-3443	(212)697-3871
567-76-00-23	pelin	bezginsoy	568-37-8654	history	female	hürriyet str. no:23/1	(532)486-1686	(342)312-0253
567-90-12-34	sevil	belen	245-20-0002	mathematic	female	atatürk str.no:32/8	(542)885-1985	(452)714-1898

**Fig. 4.3 Teacher Table**

Payment_Plan	ST_NUMBER	ST_NAME	ST_SURNAME	TOTAL_PAY	NUM_INSTAL	INSTALMENT	DAY_PAYMEN	SEPTEMBER	DECEMBER	MARCH	MAY
1	20010345	burcu	bilgin	1200	3	400	25	instalment	not	instalment	instalment
2	20010712	ayse	haskol	800	2	400	25	instalment			instalment
3	20010456	basak	beginsoy	1000	1	1000	5	not		not	instalment
4	20044789	bilge	batu	1500	3	500	1		instalment	instalment	instalment
5	20010789	ibrahim	çolak	1600	4	400	10	instalment	instalment	instalment	instalment
6	20030465	ozan	özmen	1800	2	900	30		instalment		instalment
7	20040330	burak	bilgin	500	1	500	15			instalment	

**Fig. 4.4 Student Payment Plan Table**

Table : C:\...Attendance.DB						
Attendance	St_number	St_name	St_surname	Absence_day	Absence_lesson	Absence_time
1						
2	20010486	pelin	bezginsoy	02.04.2006	mathematic	11:40:12
3	20030002	seçil	belen	02.06.2006	chemistry	09:30:00
4	20040459	burak	bilgin	12.05.2006	turkish	15:10:32
5	20050001	basak	bezginsoy	01.06.2006	history	13:30:45
6	20060339	ozan	özmen	22.05.2006	mathematic	09:30:45

**Fig. 4.5** Student Attendance