

NEAR EAST UNIVERSITY

ENGINEERING FACULTY

**COMPUTER ENGINEERING
DEPARTMENT**

**COM400
GRADUATION PROJECT**

BERRAH HEALTH CENTER

**The patient's illness information using Delphi 6.0 with
Microsoft Access Database**

Student Name: Berrah ÇENGEL

Student Number: 20020491

Supervisor: Asst. Prof. Dr. Elbrus IMANOV

NICOSIA 2007

ACKNOWLEDGEMENT

First off all I want to thank to my supervisor Ass. Prof. Dr. Elbrus IMANOV for his help valuable advices.

Secondly I want to thanks to my parents who give me chance to be a computer engineering in Turkish Republic of Northern CYPRUS.

Thirdly I want to thanks to my friends 'Hakan KILIÇ' who help me about my software design.

Fourthly and finally I want to thanks to my father 'Muzaffer ÇENGEL who is the meaning of my life.

ABSTRACT

This project is a patient registration and illness information which I called Berrah Health Center. This software is programmed at Borland Delphi 6.0 with Microsoft Access XP for creating database and tables. For managing these tables, I use simple SQL queries.

The aim of the software is to prevent the complexity in the patient's record at Pediatrician Doctor's office and is to avoid the wasting too much time for arranging appointments.

Berrah Health Center Software is a useful registration program that works on all platforms. There are about a lot of patients' new registrations and old illness information. There is extra information about Kinds of Vaccines for Doctors. Users can take appointment for next days.

CONTENTS.

Acknowledgement	2
Abstract	3
Contents	4
CHAPTER ONE: DELPHI	6
Introduction to Delphi	6
1.1 What is Delphi	6
1.1.2 What kind of programming can you do with Delphi?	7
1.1.3 How do they differ? 1.2.The VCL to Applications Developers	8
1.2.1 The VCL to Component Writers	10
1.2.2The VCL is made up of components	11
1.3.1Component Types	12
1.3.2Standard Components	12
1.3.3 Custom components	13
1.3.4Graphical components	13
1.3.5Non-visual components	14
1.3.6Structure of a component	14
1.3.7Component properties	14
1.3.8Properties provide access to internal storage field	15
1.4Property-access methods	16
1.5Types of properties	17
1.6Methods	17
1.7 Events	17
1.8Containership	20
1.9 Ownership	20
1.10Parenthood	21
CHAPTER TWO DATABASE DESIGN	22
2.Brief history about database	22
2.1 De-merits of absence of database	23
2.2 Merits of database	23
2.3 Introduction to database design	24

2.4 Database Models	25
2.4.1 Flat Model	25
2.4.2 Network Model	25
2.4.3 Relational Model	26
2.4.3.1 Why we use a Relational Database Design	27
2.5 Relationships between Tables	27
2.5.2 One-To-One Relationships	27
2.5.3 One-To-Many Relationships	28
2.6 Data Modeling	28
2.6.1 Database Normalization	29
2.6.2 Primary Key	29
2.6.3 Foreign Key	30
2.6.4 Compound Key	31
2. 2.7 Visual Basic Editor	31
2.8 Structured Query Language	33
2.9 Description of SQL	33
2.10 SQL Keywords	34
2.10.1 Data Retrieval	34
2.10.2 Data Manipulation	35
2.10.3 Data Transaction	35
2.10.4 Data Definition	36
2.11. Microsoft Access Database System	36
2.11.1 A Few Terms	36
2.11.2 Introductory Microsoft Access	37
2.11.3 Introduction to Tables	38
2.11.4 Table's data types	39
CHAPTER 3: DESCRIPTION ABOUT BERRAH HEALTH CENTER SOFTWARE	43
3.1. Software Requirement Document	43
3.2. Starting a Borland Delphi 6.0 with Microsoft Access XP	43
3.2.1. Designing Forms	44
3.3. Working with Berrah Health Center	54
THE CODES OF THE MAIN FORM	65
THE CODES OF THE RM FORM	70
THE CODES OF THE SEARCH FORM	71

CHAPTER ONE:INTRODUCTION TO DELPHI 6.0

In this project I will answer some basic questions about Delphi, to give a feel for where it came from, what it has to offer, and where it is going in the future. This is an essential part of any course. We feel it is important for those studying a new programming language to understand the ideology and intended use of the language. Too many programmers are tempted to use the language that they know, rather than learn a new one to cope with the specific demands of the project that they have. At the end of this lecture, you should have gained sufficient understanding of the Delphi ideology to decide if it is a suitable language for a specific project that you have.

1.1 What is Delphi?

Delphi is an object oriented, component based, visual, rapid development environment for event driven Windows applications, based on the Pascal language. Unlike other popular competing Rapid Application Development (RAD) tools, Delphi compiles the code you write and produces really tight, natively executable code for the target platform. In fact the most recent versions of Delphi optimise the compiled code and the resulting executables are as efficient as those compiled with any other compiler currently on the market. The term "visual" describes Delphi very well. All of the user interface development is conducted in a What You See Is What You Get environment (WYSIWYG), which means you can create polished, user friendly interfaces in a very short time, or prototype whole applications in a few hours.

Delphi is, in effect, the latest in a long and distinguished line of Pascal compilers (the previous versions of which went by the name "Turbo Pascal") from the company formerly known as Borland, now known as Inprise. In common with the Turbo Pascal compilers that preceded it, Delphi is not just a compiler, but a complete development environment. Some of the facilities that are included in the "Integrated Development Environment" (IDE) are listed below:

- A syntax sensitive program file editor
- A rapid optimising compiler

- Built in debugging /tracing facilities
- A visual interface developer
- Syntax sensitive help files
- Database creation and editing tools
- Image/Icon/Cursor creation / editing tools
- Version Control CASE tools

The development environment itself is extensible, and there are a number of add ins available to perform functions such as memory leak detection and profiling. In short, Delphi includes just about everything you need to write applications that will run on an Intel platform under Windows, but if your target platform is a Silicon Graphics running IRIX, or a Sun Sparc running SOLARIS, or even a PC running LINUX, then you will need to look elsewhere for your development tool.

This specialisation on one platform and one operating system, makes Delphi a very strong tool. The code it generates runs very rapidly, and is very stable, once your own bugs have been ironed out

!

1.1.1 What kind of programming can you do with Delphi?

The simple answer is "more or less anything". Because the code is compiled, it runs quickly, and is therefore suitable for writing more or less any program that you would consider a candidate for the Windows operating system.

You probably won't be using it to write embedded systems for washing machines, toasters or fuel injection systems, but for more or less anything else, it can be used.

Some projects to which Delphi is suited:

- Simple, single user database applications
- Intermediate multi-user database applications
- Large scale multi-tier, multi-user database applications
- Internet applications

- Graphics Applications
- Multimedia Applications
- Image processing/Image recognition
- Data analysis
- System tools
- Communications tools using the Internet, Telephone or LAN
- Web based applications

This is not intended to be an exhaustive list, more an indication of the depth and breadth of Delphi's applicability. Because it is possible to access any and all of the Windows API, and because if all else fails, Delphi will allow you to drop a few lines of assembler code directly into your ordinary Pascal instructions, it is possible to do more or less anything. Delphi can also be used to write Dynamically Linked Libraries (DLLs) and can call out to DLLs written in other programming languages without difficulty.

Because Delphi is based on the concept of self contained Components (elements of code that can be dropped directly on to a form in your application, and exist in object form, performing their function until they are no longer required), it is possible to build applications very rapidly. Because Delphi has been available for quite some time, the number of pre-written components has been increasing to the point that now there is a component to do more or less anything you can imagine. The job of the programmer has become one of gluing together appropriate components with code that operates them as required.

1.1.2 How do they differ?

Borland (as they were then) has a long tradition in the creation of high speed compilers. One of their best known products was Turbo Pascal - a tool that many programmers cut their teeth on. With the rise in importance of the Windows environment, it was only a matter of time before development tools started to appear that were specific to this new environment. In the very beginning, Windows produced SDKs (software development kits) that were totally non-visual (user interface development was totally separated from the development of the actual application), and required great patience and some genius to get anything working with. Whilst these

tools slowly improved, they still required a really good understanding of the inner workings of Windows a great extent these criticisms were dispatched by the release of Microsoft's Visual Basic product, which attempted to bring Windows development to the masses. It achieved this to a great extent too, and remains a popular product today. However, it suffered from several drawbacks:

- 1) It wasn't as stable as it might have been
- 2) It was an interpreted language and hence was slow to run
- 3) It had as its underlying language BASIC, and most "real" programmers weren't so keen!

Into this environment arrived the eye opening Delphi I product, and in many ways the standard for visual development tools for Windows was set. This first version was a 16 bit compiler, and produced executable code that would run on Windows 3.1 and Windows 3.11. Of course, Microsoft have ensured (up to now) that their 32 bit operating systems (Win95, Win98, and Win NT) will all run 16 bit applications, however, many of the features that were introduced in these newer operating systems are not accessible to the 16 bit applications developed with Delphi I. Delphi 2 was released quite soon after Delphi I, and in fact included a full distribution of Delphi I on the same CD. Delphi 2, (and all subsequent versions) have been 32 bit compilers, producing code that runs exclusively on 32bit Windows platforms. (We ignore for simplicity the WIN32S DLLs which allow Win 3.1x to run some 32 bit applications).

Delphi is currently standing at Version 4.0, with a new release (version 5.0) expected shortly. In its latest version, Delphi has become somewhat feature loaded, and as a result, we would argue, less stable than the earlier versions. However, in its defence, Delphi (and Borland products in general) have always been more stable than their competitors products, and the majority of Delphi 4's glitches are minor and forgivable - just don't try and copy/paste a selection of your code, midway through a debugging session!

The reasons for the version progression include the addition of new components, improvements in the development environment, the inclusion of more

internet related support and improvements in the documentation. Delphi at version 4 is a very mature product, and Inprise has always been responsive in developing the product in the direction that the market requires it to go. Predominantly this means right now, the inclusion of more and more Internet, Web and CORBA related tools and components - a trend we are assured continues with the release of version 5.0

For each version of Delphi there are several sub-versions, varying in cost and features, from the most basic "Developer" version to the most complete (and expensive) "Client Server" version. The variation in price is substantial, and if you are contemplating a purchase, you should study the feature list carefully to ensure you are not paying for features you will never use. Even the most basic "Developer" version contains the vast majority of the features you are likely to need on a day to day basis. Don't assume that you will need Client Server, simply because you are intending to write a large database application - The developer edition is quite capable of this.

1.2.The VCL to Applications Developers

Applications Developers create complete applications by interacting with the Delphi visual environment (as mentioned earlier, this is a concept nonexistent in many other frameworks). These people use the VCL to create their user-interface and the other elements of their application: database connectivity, data validation, business rules, etc..

Applications Developers should know which properties, events, and methods each component makes available. Additionally, by understanding the VCL architecture, Applications Developers will be able to easily identify where they can improve their applications by extending components or creating new ones. Then they can maximize the capabilities of these components, and create better applications.

1.2.1 The VCL to Component Writers

Component Writers expand on the existing VCL, either by developing new components, or by increasing the functionality of existing ones. Many component writers make their components available for Applications Developers to use.

A Component Writer must take their knowledge of the VCL a step further than that of the Application Developer. For example, they must know whether to write a new component or to extend an existing one when the need for a certain characteristic arises. This requires a greater knowledge of the VCL's inner workings.

1.2.2The VCL is made up of components

Components are the building blocks that developers use to design the user-interface and to provide some non-visual capabilities to their applications. To an Application Developer, a component is an object most commonly dragged from the Component palette and placed onto a form. Once on the form, one can manipulate the component's properties and add code to the component's various events to give the component a specific behavior. To a Component Writer, components are objects in Object Pascal code. Some components encapsulate the behavior of elements provided by the system, such as the standard Windows 95 controls. Other objects introduce entirely new visual or non-visual elements, in which case the component's code makes up the entire behavior of the component.

The complexity of different components varies widely. Some might be simple while others might encapsulate an elaborate task. There is no limit to what a component can do or be made up of. You can have a very simple component like a **TLabel**, or a much more complex component which encapsulates the complete functionality of a spreadsheet.

1.2.2The VCL is made up of components

Components are really just special types of objects. In fact, a component's structure is based on the rules that apply to Object Pascal. There are three fundamental keys to understanding the VCL.

First, you should know the special characteristics of the four basic component types: standard controls, custom controls, graphical controls and non-visual components.

Second, you must understand the VCL structure with which components are built. This really ties into your understanding of Object Pascal's implementation.

Third, you should be familiar with the VCL hierarchy and you should also know where the four component types previously mentioned fit into the VCL hierarchy. The following paragraphs will discuss each of these keys to understanding the VCL.

1.3.1 Component Types

As a component writer, there four primary types of components that you will work with in Delphi: standard controls, custom controls, graphical controls, and non-visual components. Although these component types are primarily of interest to component writers, it's not a bad idea for applications developers to be familiar with them. They are the foundations on which applications are built.

1.3.2 Standard Components,

Some of the components provided by Delphi 2.0 encapsulate the behavior of the standard Windows controls: TButton, TListbox As a component writer, there four primary types of components that you will work with in Delphi: standard controls, custom controls, graphical controls, and non-visual components. Although these component types are primarily of interest to component writers, it's not a bad idea for applications developers to be familiar with them. They are the foundations on which applications are built.

For example. You will find these components on the Standard page of the Component Palette. These components are Windows' common controls with Object Pascal wrappers around them.

Each standard component looks and works like the Windows' common control which it encapsulates. The VCL wrapper's simply makes the control available to you in the form of a Delphi component-it doesn't define the common control's appearance or functionality, but rather, surfaces the ability to modify a control's

appearance/functionality in the form of methods and properties. If you have the VCL source code, you can examine how the VCL wraps these controls in the file `STDCTRLS.PAS`.

If you want to use these standard components unchanged, there is no need to understand how the VCL wraps them. If, however, you want to extend or change one of these components, then you must understand how the Windows common control is wrapped by the VCL into a Delphi component.

For example, the Windows class `LISTBOX` can display the list box items in multiple columns. This capability, however, isn't surfaced by Delphi's `TListBox` component (which encapsulates the Windows `LISTBOX` class). (`TListBox` only displays items in a single column.) Surfacing this capability requires that you override the default creation of the `TListBox` component.

This example also serves to illustrate why it is important for Applications Developers to understand the VCL. Just knowing this tidbit of information helps you to identify where enhancements to the existing library of components can help make your life easier and more productive.

1.3.3 Custom components

Unlike standard components, custom components are controls that don't already have a method for displaying themselves, nor do they have a defined behavior. The Component Writer must provide code that tells the component how to draw itself and determines how the component behaves when the user interacts with it. Examples of existing custom components are the `TPanel` and `TStringGrid` components.

It should be mentioned here that both standard and custom components are *windowed* controls. A "windowed control" has a window associated with it and, therefore, has a window handle. Windowed controls have three characteristics: they can receive the input focus, they use system resources, and they can be parents to other controls. (Parents is related to containership, discussed later in this paper.) An example of a component which can be a container is the `TPanel` component.

1.3.4 Graphical components

Graphical components are visual controls which cannot receive the input focus from the user. They are non-windowed controls. Graphical components allow you to display something to the user without using up any system resources; they have less "overhead" than standard or custom components. Graphical components don't require a window handle-thus, they cannot get focus. Some examples of graphical components are the TLabel and TShape components.

Graphical components cannot be containers of other components. This means that they cannot own other components which are placed on top of them.

1.3.5 Non-visual components

Non-visual components are components that do not appear on the form as controls at run-time. These components allow you to encapsulate some functionality of an entity within an object. You can manipulate how the component will behave, at design-time, through the Object Inspector. Using the Object Inspector, you can modify a non-visual component's properties and provide event handlers for its events. Examples of such components are the TOpenDialog, TTable, and TTimer components

1.3.6 Structure of a component

All components share a similar structure. Each component consists of common elements that allow developers to manipulate its appearance and function via properties, methods and events. The following sections in this paper will discuss these common elements as well as talk about a few other characteristics of components which don't apply to all components

1.3.7 Component properties

Properties provide an extension of an object's fields. Unlike fields, properties do not store data: they provide other capabilities. For example, properties may use methods to read or write data to an object field to which the user has no access. This

adds a certain level of protection as to how a given field is assigned data. Properties also cause "side effects" to occur when the user makes a particular assignment to the property. Thus what appears as a simple field assignment to the component user could trigger a complex operation to occur behind the scenes.

1.3.8 Properties provide access to internal storage fields

There are two ways that properties provide access to internal storage fields of components: directly or through access methods. Examine the code below which illustrates this process.

```
TCustomEdit = class(TWinControl)
private
    FMaxLength: Integer;
protected
    procedure SetMaxLength(Value: Integer);
...
published
    property MaxLength: Integer read
        FMaxLength write SetMaxLength default 0;
...
end;
```

The code above is snippet of the TCustomEdit component class. TCustomEdit is the base class for edit boxes and memo components such as TEdit, and TMemo.

TCustomEdit has an internal field FMaxLength of type Integer which specifies the maximum length of characters which the user can enter into the control. The user doesn't directly access the FMaxLength field to specify this value. Instead, a value is added to this field by making an assignment to the MaxLength property.

The property MaxLength provides the access to the storage field FMaxLength. The property definition is comprised of the property name, the property type, a read declaration, a write declaration and optional default value.

The read declaration specifies how the property is used to read the value of an internal storage field. For instance, the MaxLength property has direct read access to

FMaxLength. The write declaration for MaxLength shows that assignments made to the MaxLength property result in a call to an *access method* which is responsible for assigning a value to the FMaxLength storage field. This access method is SetMaxLength.

1.4Property-access methods

Access methods take a single parameter of the same type as the property. One of the primary reasons for write access methods is to cause some side-effect to occur as a result of an assignment to a property. Write access methods also provide a method layer over assignments made to a component's fields. Instead of the component user making the assignment to the field directly, the property's write access method will assign the value to the storage field if the property refers to a particular storage field. For example, examine the implementation of the SetMaxLength method below.

```
procedure TCustomEdit.SetMaxLength(Value: Integer);
begin
    if FMaxLength <> Value then
    begin
        FMaxLength := Value;
        if HandleAllocated then
            SendMessage(Handle, EM_LIMITTEXT, Value, 0);
    end;
end;
```

The code in the SetMaxLength method checks if the user is assigning the same value as that which the property already holds. This is done as a simple optimization. The method then assigns the new value to the internal storage field, FMaxLength. Additionally, the method then sends an EM_LIMITTEXT Windows message to the window which the TCustomEdit encapsulates. The EM_LIMITTEXT message places a limit on the amount of text that a user can enter into an edit control. This last step is what is referred to as a *side-effect* when assigning property values. Side effects are any additional actions that occur when assigning a value to a property and can be quite sophisticated.

Providing access to internal storage fields through property access methods offers the advantage that the Component Writer can modify the implementation of a class without modifying the interface. It is also possible to have access methods for the read access of a property. The read access method might, for example, return a type which is different than that of a properties storage field. For instance, it could return the string representation of an integer storage field.

Another fundamental reason for properties is that properties are accessible for modification at run-time through Delphi's Object Inspector. This occurs whenever the declaration of the property appears in the published section of a component's declaration.

1.5 Types of properties

Properties can be of the standard data types defined by the Object Pascal rules. Property types also determine how they are edited in Delphi's Object Inspector. The table below shows the different property types as they are defined in Delphi's online help.

1.6 Methods

Since components are really just objects, they can have methods. We will discuss some of the more commonly used methods later in this paper when we discuss the different levels of the VCL hierarchy.

1.7 Events

Events provide a means for a component to notify the user of some pre-defined occurrence within the component. Such an occurrence might be a button click or the pressing of a key on a keyboard.

Components contain special properties called events to which the component user assigns code. This code will be executed whenever a certain event occurs. For instance, if you look at the events page of a TEdit component, you'll see such events as

OnChange, OnClick and OnDblClick. These events are nothing more than pointers to methods.

When the user of a component assigns code to one of those events, the user's code is referred to as an event handler. For example, by double clicking on the events page for a particular event causes Delphi to generate a method and places you in the Code Editor where you can add your code for that method. An example of this is shown in the code below, which is an OnClick event for a TButton component.

```
TButton component.  
TForm1 = class(TForm)  
    Button1: Tbutton;  
    procedure Button1Click(Sender: TObject);  
end;  
...  
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    { Event code goes here }  
end;
```

It becomes clearer that events are method pointers when you assign an event handler to an event programmatically. The above example was Delphi generated code. To link your own an event handler to a TButton's OnClick event at run time you must first create a method that you will assign to this event. Since this is a method, it must belong to an existing object. This object can be the form which owns the TButton component although it doesn't have to be. In fact, the event handlers which Delphi creates belong to the form on which the component resides. The code below illustrates how you would create an event handler method.

```
TForm1 = class(TForm)  
    Button1: TButton;  
    ...  
private  
    MyOnClickEvent(Sender: TObject); //  
    Your method declaration  
end;  
...
```

```

{ Your method definition below }
procedure TForm1.MyOnClickEvent(Sender: TObject);
begin
    { Your code goes here }
end;

```

The MyOnClickEvent method becomes the event handler for Button1.OnClick when it is assigned to Button1.OnClick in code as shown below.

```

Button1.OnClick := MyOnClickEvent

```

This assignment can be made anytime at runtime, such as in the form's OnCreate event handler. This is essentially the same thing that happens when you create an event handler through Delphi's Object Inspector except that Delphi generates the method declaration.

When you define methods for event handlers, these methods must be defined as the same type as the event property and the field to which the event property refers. For instance, the OnClick event refers to an internal data field, FOnClick. Both the property OnClick, and field FOnClick are of the type TNotifyEvent. TNotifyEvent is a procedural type as shown below:

```

TNotifyEvent = procedure (Sender: TObject) of object;

```

Therefore, if you are creating a method for an OnClick event, it must be defined with the same type and number of parameters as shown below.

```

TForm1 = class(TForm)
...
    procedure (Sender: TObject);
end;

```

Note the use of the of object specification. This tells the compiler that the procedure definition is actually a method and performs some additional logic like ensuring that an implicit Self parameter is also passed to this method when called. Self is just a pointer reference to the class to which a method belongs.

1.8 Containership

Some components in the VCL can own other components as well as be parents to other components. These two concepts have a different meaning as will be discussed in the section to follow.

1.9 Ownership

All components may be owned by other components but not all components can own other components. A component's Owner property contains a reference to the component which owns it.

The basic responsibility of the owner is one of resource management. The owner is responsible for freeing those components which it owns whenever it is destroyed. Typically, the form owns all components which appear on it, even if those components are placed on another component such as a TPanel. At design-time, the form automatically becomes the owner for components which you place on it. At run-time, when you create a component, you pass the owner as a parameter to the component's constructor. For instance, the below shows how to create a TButton component at run-time and passes the form's implicit Self variable to the TButton's Create constructor. TButton.Create will then assign whatever is passed to it, in this case Self or rather the form, and assign it to the button's Owner property.

```
MyButton := TButton.Create(self);
```

When the form that now owns this TButton component gets freed, MyButton will also be freed.

You can create a component without an owner by passing nil to the component's Create constructor, however, you must ensure that the component is freed when it is no longer needed. The code below shows you how to do this for a TTable component.

```
MyTable := TTable.Create(nil)
try
    { Do stuff with MyTable }
```



```

finally
    MyTable.Free;
end;

```

As shown in the code above, it is best to use a try..finally block to ensure that the component gets freed even if an exception were to be raised.

The Components property of a component is an array property which contains a list of the components which it owns. For instance, the code below shows how to loop through a form's components and then shows their class name.

```

var
    I: integer;
begin
    for I := 0 to ComponentCount - 1 do
        ShowMessage(Components[i].ClassName);
    end;

```

1.10 Parenthood

Parenthood is a much different concept from ownership. It applies only to windowed components, which can be parents to other components. Later, when we discuss the VCL hierarchy, you will see the level in the hierarchy which introduces windowed controls.

Parent components are responsible for the display of other components. They call the appropriate methods internally that cause the children components to draw themselves. The Parent property of a component refers to the component which is its parent. Also, a component's parent does not have to be its owner. Although the parent component is mainly responsible for the display of components, it also frees children components when it is destroyed.

Windowed components are controls which are visible user interface elements such as edit controls, list boxes and memo controls. In order for a windowed component to be displayed, it must be assigned a parent on which to display itself.

CHAPTER 3: DESCRIPTION ABOUT BERRAH HEALTH CENTER SOFTWARE

Now, I want to describe my project in details step by step. My project is Berrah Health Center Software. I create this program by using Borland Delphi 6.0 with Microsoft Access XP.

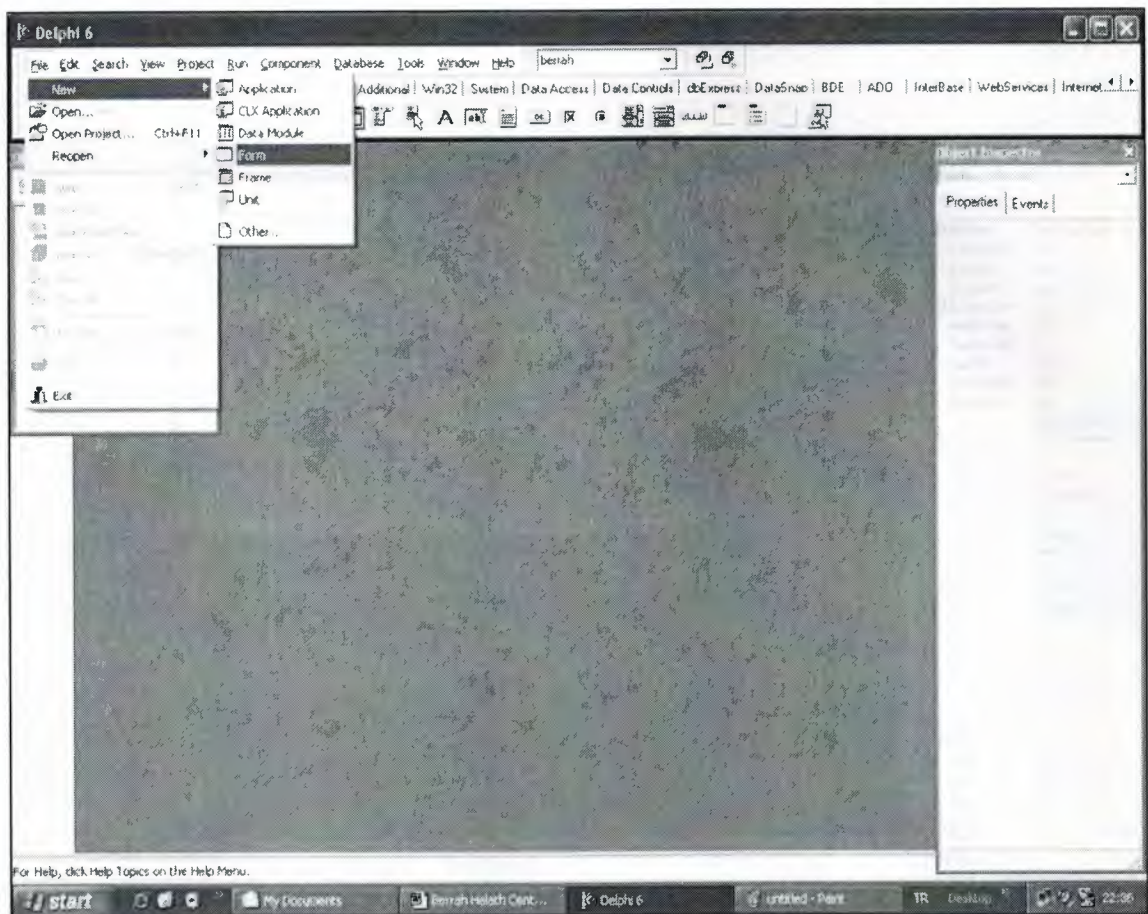
3.1. Software Requirement Document

First step on my project was read Software requirement document which is prepared by me with Assoc. Prof. Kudret Çağlar who is Pediatrician and Pediatric oncologist. I learned about Doctor's inspector operations more. What they need to use program. They need registration and remember the patient's illness. They want to see the personal information's like addresses and phone numbers easily. They can receive that informations very quickly with using my software. These forms are used in the software

- Patient Registration
- Search Edit
- Illness info
- Inoculations info
- Inoculations kinds
- Calendar
- Report

3.2. Starting a Borland Delphi 6.0 with Microsoft Access Xp

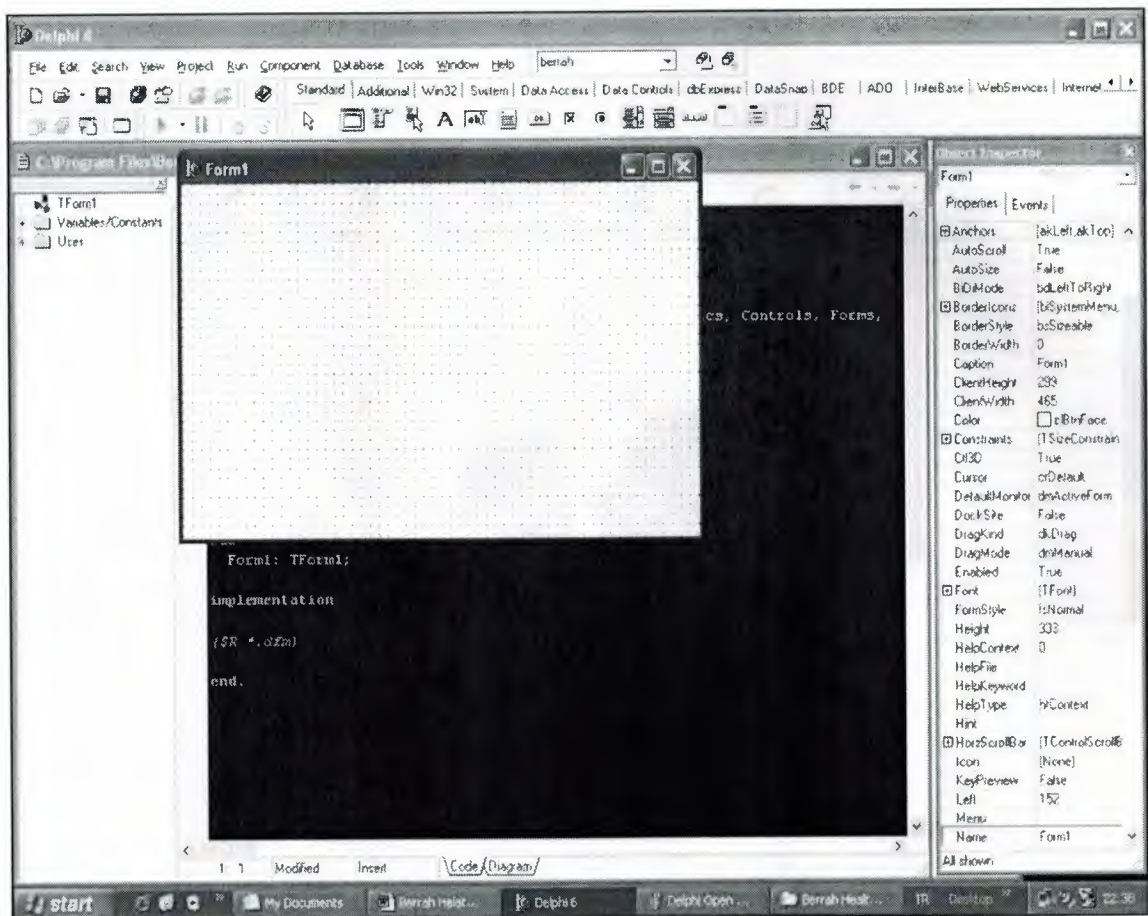
I mention about Borland Delphi 6.0 but here I mention it an easy way. Borland Delphi 6.0 with Microsoft Access XP has many special tools and many components to create a project. After opening Borland Delphi 6.0, you see the blank screen. For creating a new form. You would click the file menu and then Open>Form button.



Opening new form

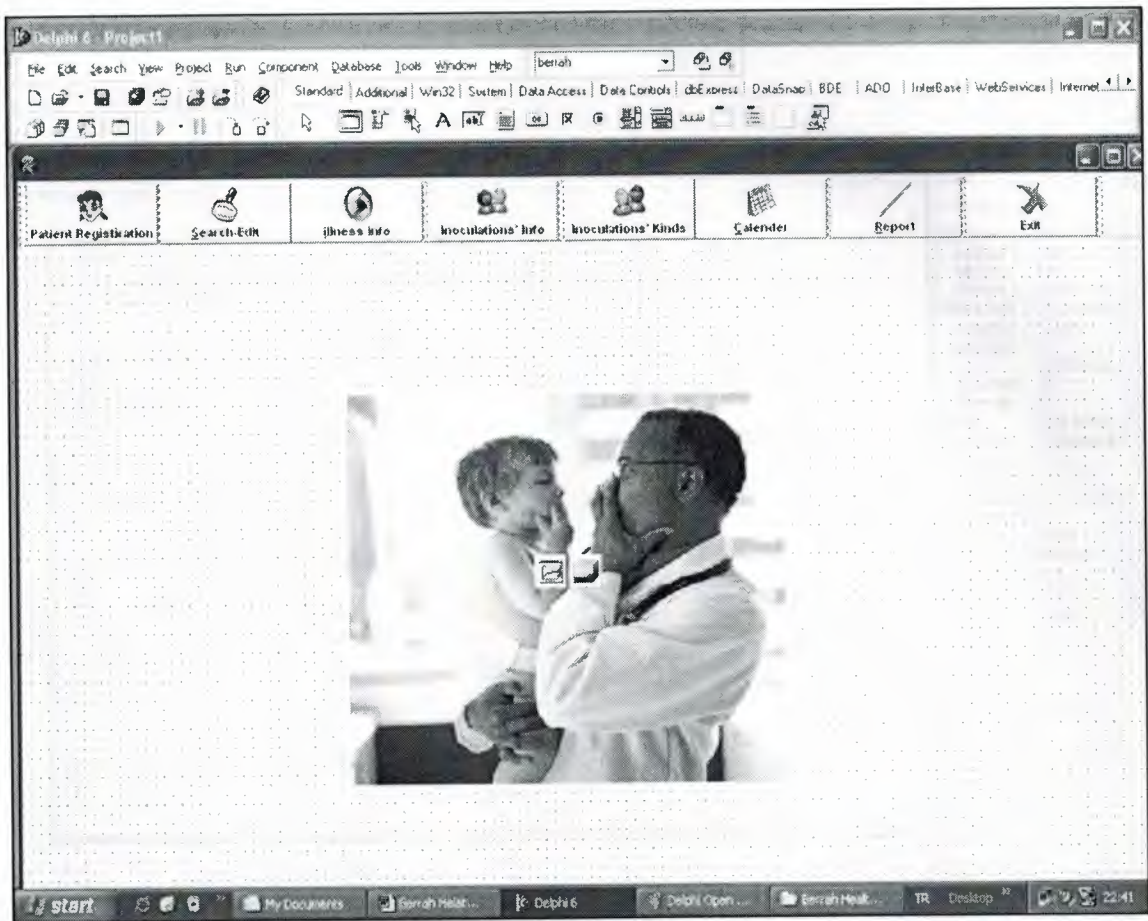
3.2.1. Designing Forms

Designing a form is very easy with Borland Delphi 6.0. After starting a project it has already a form which is called form1 ready to use.



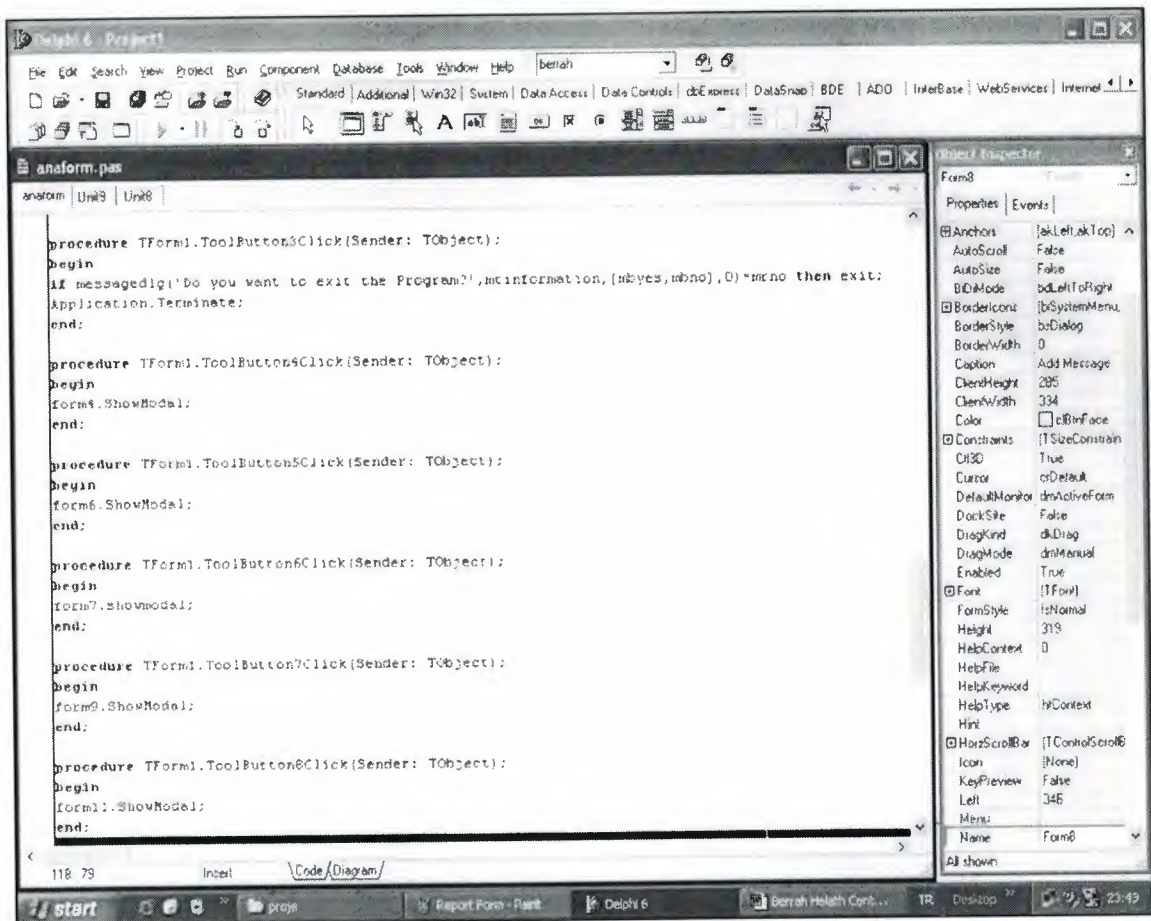
Opened Form

By using tools at top of the screen, I can add controls to my form. In below figure you can see the tools which I used in my project.



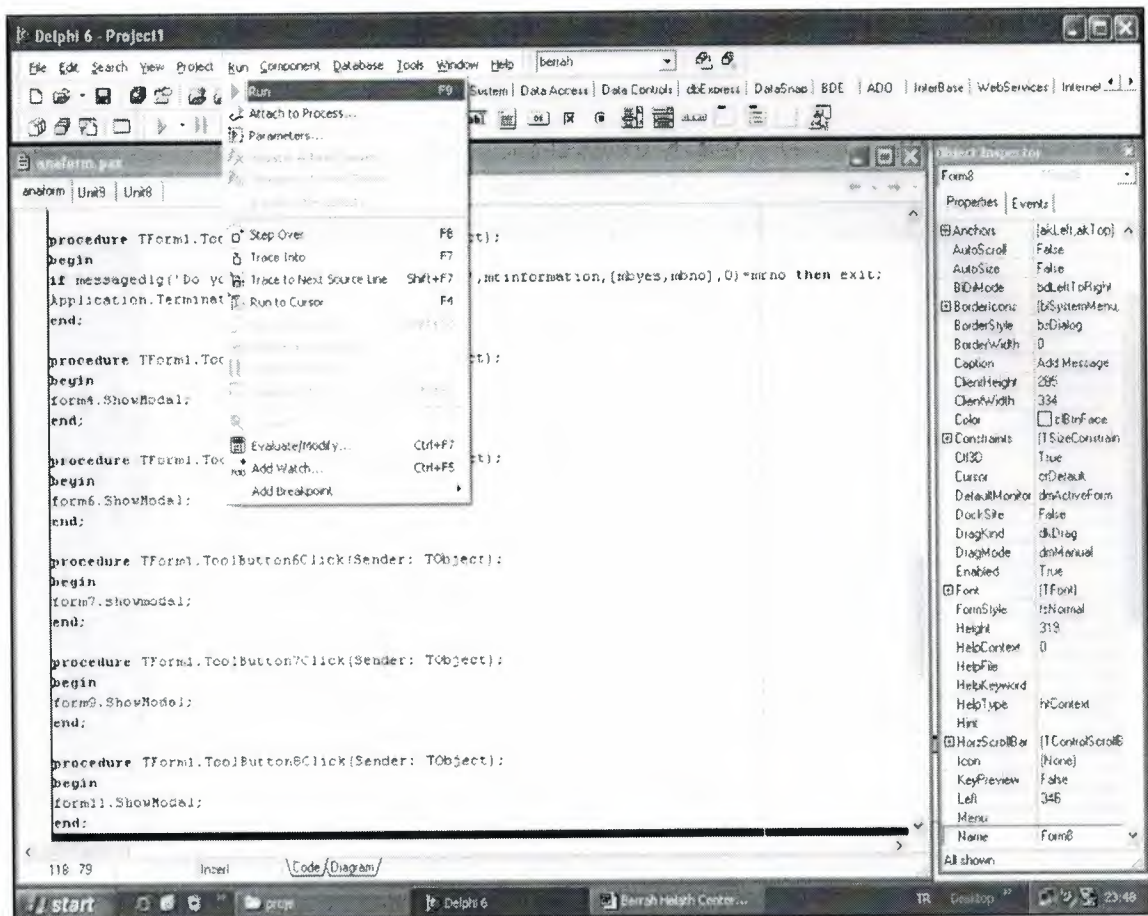
Main form

In below figure you can see the main form which I prepared other detailed forms on it. User can easily access the other forms with using pictured buttons on the top of the software.



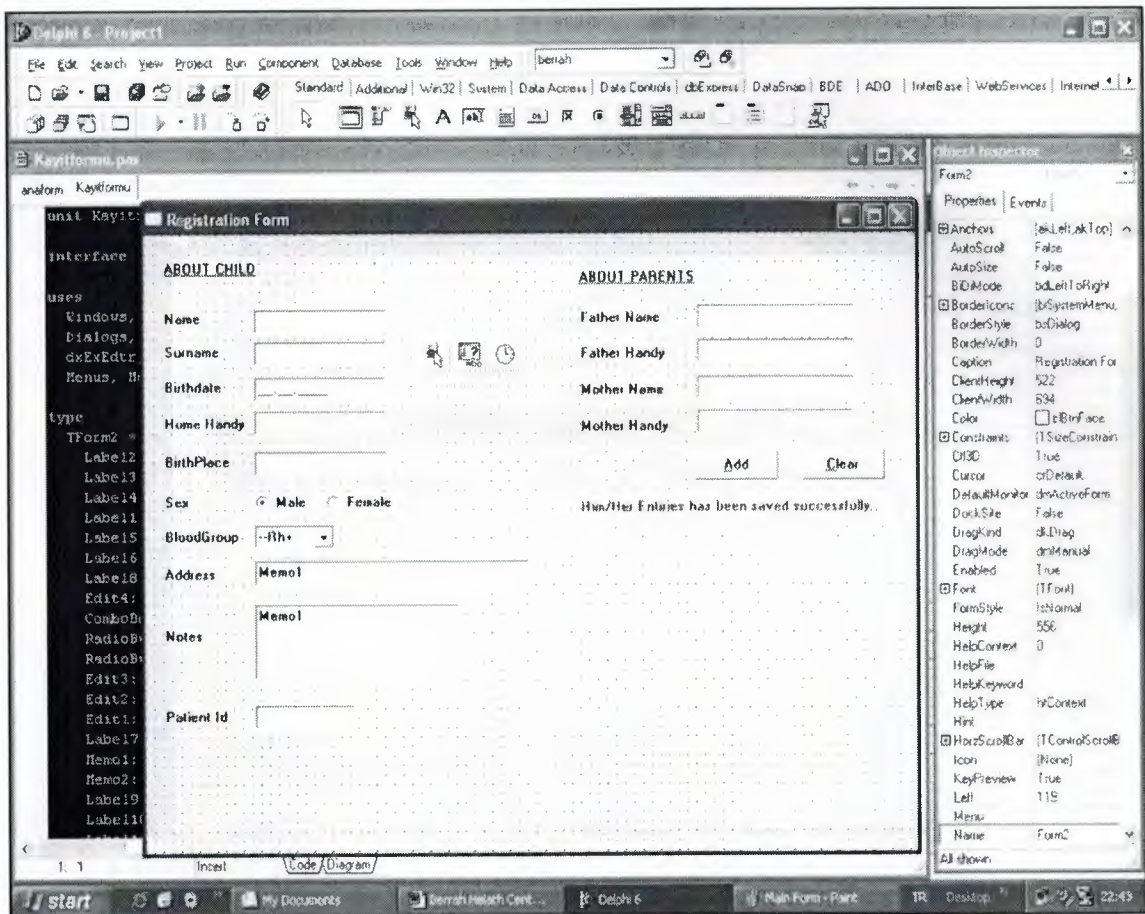
Main form Codes

To see and to change the codes, you can use code windows (units). You can access the units with view menu then clicking Units or you can access with Ctrl+F12. The project starts to run if there is no error on code.



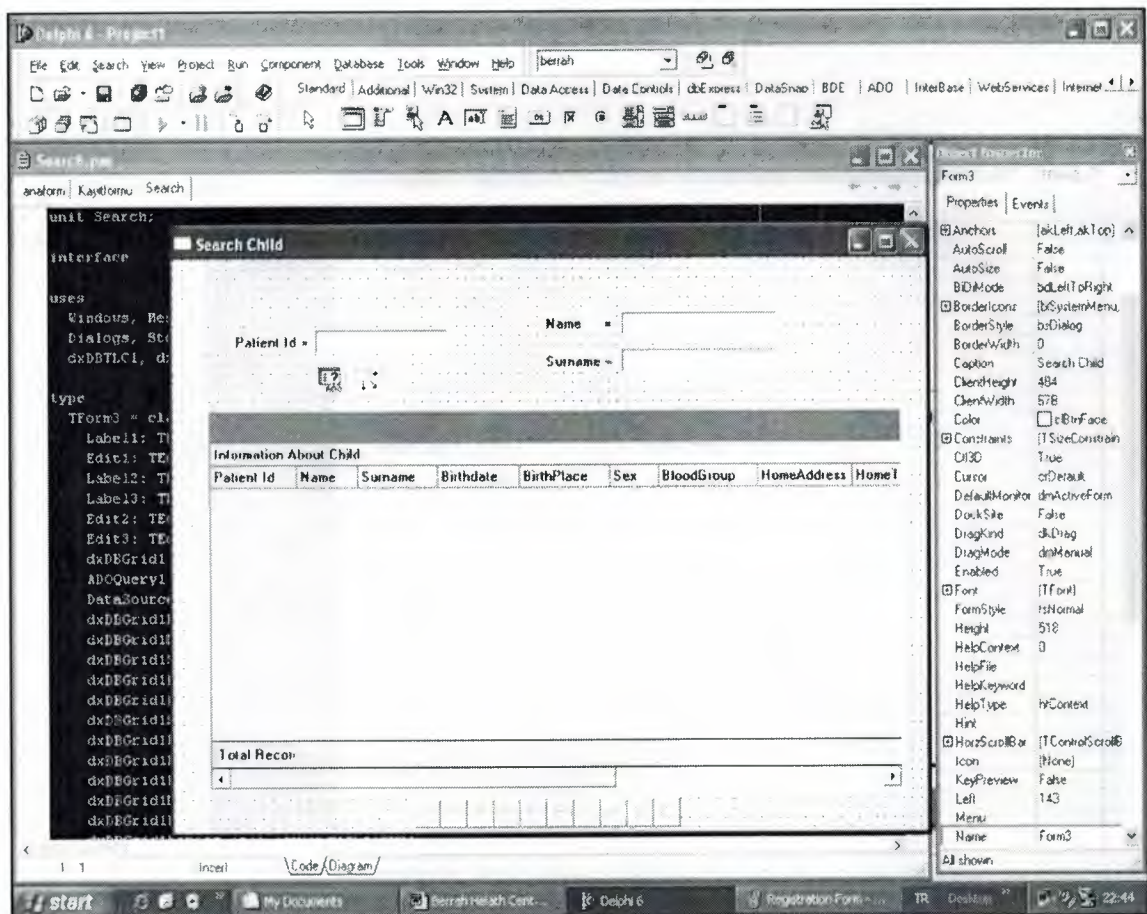
Running program

After test the project, if every think is okay about the changing the code. You would click run menu or press F9 key.



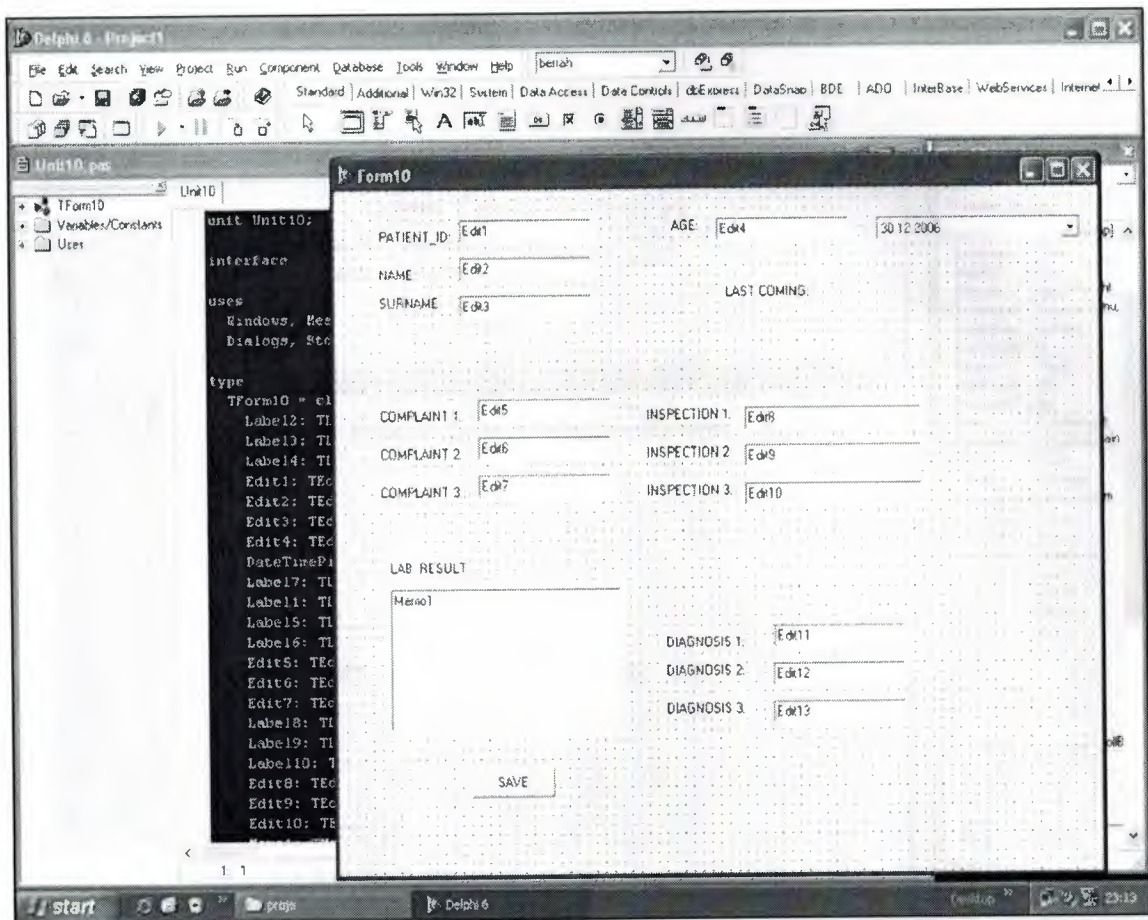
Patient Registration form design

As you see in the figure there are labels, edits ,buttons, combo boxes and other components. This form also connected the database with ADO connection.



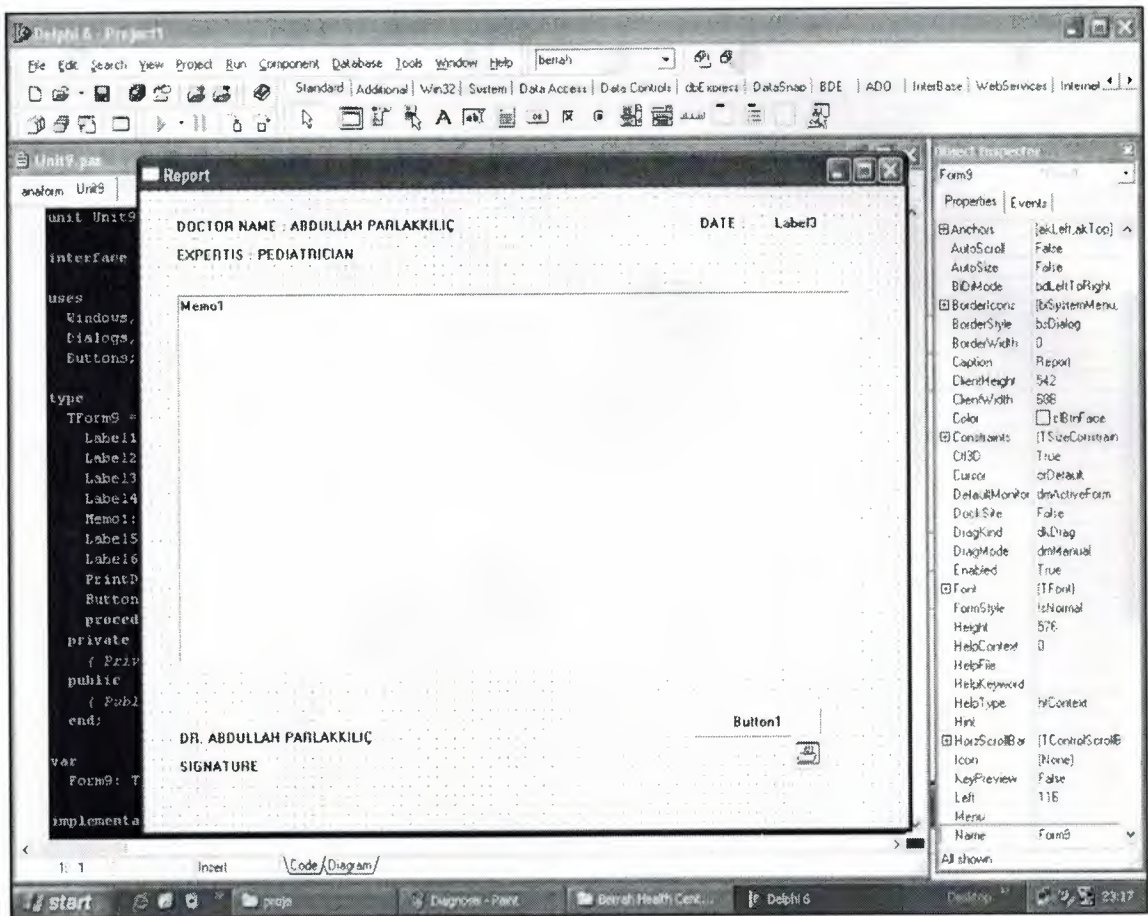
Search-Edit form design

While I was preparing this form I use labels, edits, dbgrid. User can easily add, remove or update the records.



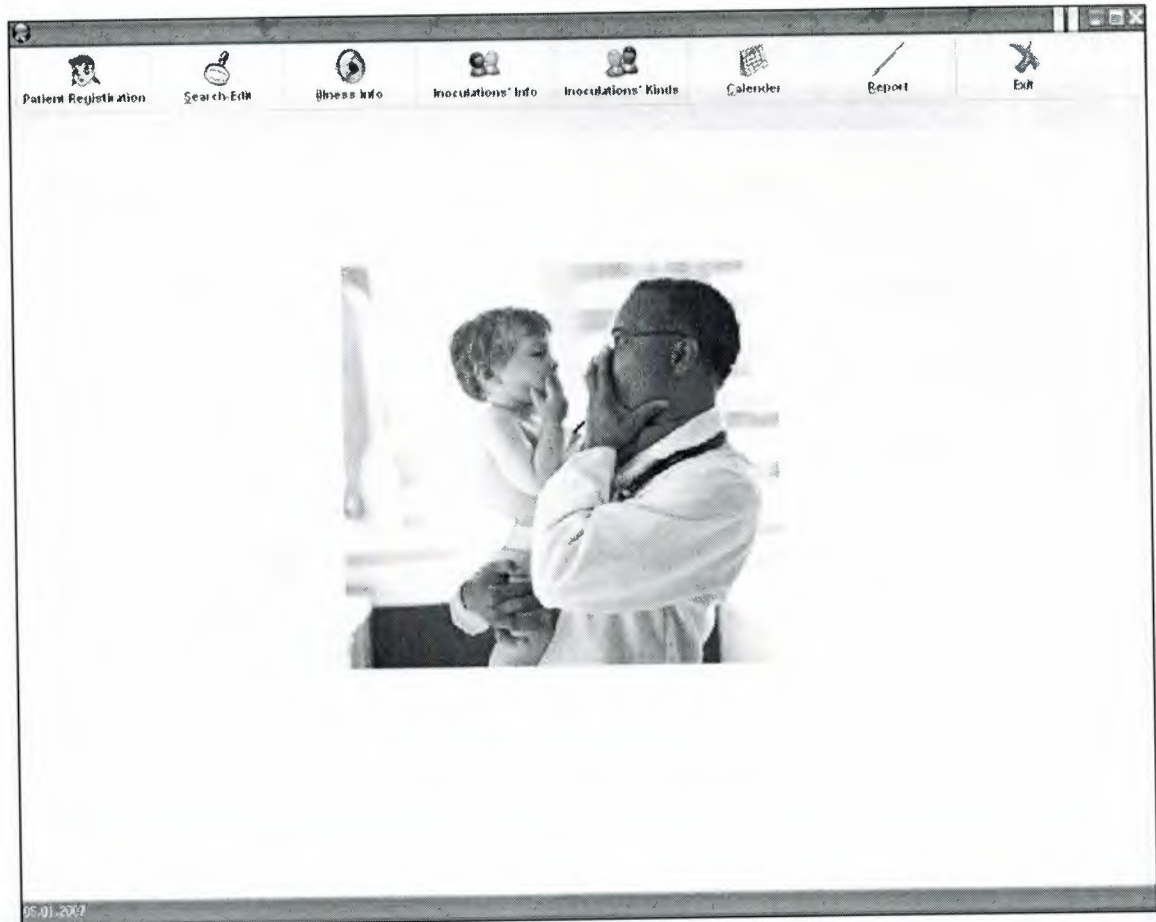
Diagnosis form design

I used labels, edits, memo ,buttons and date time picker for preparing this form. This form is designed for recording the lab results then making diagnosis.



Report form design


This form is designed for printing the information's of doctor's report.



Main form

3.3.Working with Berrah Health Center

To start the using program , beginning page is main menu which is shown in the below figure .the important and useful buttons are at the top of the page .what you want to select you can just click it.

Registration Form			
<u>ABOUT CHILD</u>		<u>ABOUT PARENTS</u>	
Name	Berrah	Father Name	Fatih
Surname	ÇENGEL	Father Handy	05332330122
Birthdate	03.02.2002	Mother Name	Sevda
Home Handy	03562137533	Mother Handy	05428341573
BirthPlace	Sakarya	<input type="button" value="Add"/> <input type="button" value="Clear"/>	
Sex	<input type="radio"/> Male <input checked="" type="radio"/> Female		
BloodGroup	A Rh+ 		
Address	Lefkoşa		
Notes	Premature		
Patient Id			

Registration Form

In registration form we record a new patient. There is information about children's personal and necessary information as blood group , birth date (when we insert the birth date program calculates automatically the age) , for receiving address , telephone numbers , and extra information about family. After filling the blanks when you click add button program saves the data's. When you click the clear button the page is cleaned and ready to insert the new informations. patient id is given by program own self. You do not insert any data to that edit. add this id is as a personal identity.

Search Child

Patient Id = Name =
Surname =

Information About Child

Patient Id	Name	Surname	Birthdate	BirthPlace	Sex	BloodGroup	HomeAddress	HomeT
2	Aleyna	Bayram	12.06.2005	Ankara	Female	0 Rh+	Kadıköy	021634

atal Record 1

Navigation icons: back, forward, search, delete, update, refresh

Search form

We use this form for searching the old patients. when you insert just name it opens automatically information's about child. you can change any data with at the bottom of the page .also you can add , delete or updated the information's. It is for if you want to search by name from name label's edit component .every patient which includes that text is shown at the list table.

Search Child

Patient Id =

Name =

Surname =

Information About Child

Patient Id	Name	Surname	Birthdate	BirthPlace	Sex	BloodGroup	HomeAddress	HomeT
6	Berrah	ÇENGEL	03.02.2002	Sakarya	Female	A Rh+	Lefkoşa	035621

1 of 1 Record 1

Navigation: [Previous] [Next] [First] [Last] [Refresh]

Search form with patient id

As you see at the previous page you can search with the name. for this form you may search just patient id. It opens data about child ,and as previous page you can edit any information's. This form for make search operations. The user can select search criteria by a edits or on the table. All the patients which are includes that is shown at the list box. You can see how many patients found in this search at the final.

illness informatio

Patient Id : 2 Dr Name :

illness information

Id	Date	Name	Surname	Complaint
2	04.01.2007	Aleyna	Bayram	wlşwaefijipewlf

05.01.2007

COMPLAINT: Headache DIAGNOSIS: Hepatid

INSPECTION: Blood Lab Test

SAVE CLOSE

Illness information

This form is used for old patients illness information's . Firstly when you enter patient id which we identified as a identity info , it opens the patients illness data .or you can search it with just name .I use it for patients who could not remember the patient id number., it is helping for users. At the bottom of the page there is complaints data , user will enter the complaints to there. Inspection is for cures. Diagnosis is after doctor cures the patient which he decide for diagnosis .there is a date. When I filled the blanks and save them it automatically saves the date. finally if we save them we can see the new data's on the table.

Choose a Patient

Name = Surname =

Patient Id =

Patient Info				
Patient Id	Name	Surname	Father Name	Mother Name
1	Cem	Keskin	Yılmaz	Ayşegül
2	Aleyna	Bayram	Sedat	Filiz
6	Berrah	ÇENGEL	Fatih	Sevda

Inoculate form

In this form when you enter just name or patient id, and you click enter button next page opens.

Form5

Name-----> Berrah Home Tel-----> 03562137533 Notes
Surname--> ÇENGEL Mother Tel-----> 05428341573 Premature
Age (Berrah) Father Tel-----> 05332330122
5 years old

Inoculations					
Patient Id	Name of Inoculation	Age of Inoculation	Kind of Inoculations	Date	Called

Kind of Inoculations: Name of Inoculations:

☒ Called ☐ Done It Age of Inoculations:

Date:

Opened inoculation page

In this form is saving calling family and making vaccine to the child .It is like giving appointment. for example at the previous page you choose the patient and click the enter button. After you did them this page opens with the patients data as name , surname , family's telephone numbers. There is an important detail for me. that is calculating the age of child. At the kind of inoculation you chose the vaccines kind. when you just select the kind , it automatically writes the name and the age of vaccine. called and done it radio buttons are for appointment and finishing the operating. also you can select the date. program automatically saves date which you chose. finally clicking the add button it takes all the data to database.

Add Inoculation

Navigation icons: Previous, First, Next, Last, Add, Subtract, Up, Refresh

Inoculations		
Inoculations' Name	Inoculations' Age	Aliens
sarlık	4	sarlık b
verem	2	berem-b
nnujolo	5	byuui
grip	7	ngksligin

Ready inoculate list form

This form is completely usefully for doctors. There is extra information for doctors about vaccines. if the doctor forget any inoculate user may open this form and remember the forgotten vaccine. it helps to users.

Report

DOCTOR NAME : ABDULLAH PARLAKKILIÇ

DATE : 05.01.2007

EXPERTIS : PEDIATRICIAN

Akut tonsillit teşhisi konmuş olup.
6 gün yatak istirahati uygundur.

DR. ABDULLAH PARLAKKILIÇ

SIGNATURE

Button1

Report Form

In this form is prepared for the children's schools. It is a report for permission from the school direction. there is a doctor name and his or her sign place. after printing out, doctor will sign it and it would given to the family. and they bring it to the child's school.

THE CODES OF THE MAIN FORM

unit anaform;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, DB, ADODB, ImgList, ComCtrls, ToolWin, WinSkinData, Menus, jpeg,
ExtCtrls;

type

```
TForm1 = class(TForm)
  ToolBar1: TToolBar;
  ToolButton1: TToolButton;
  ToolButton2: TToolButton;
  ToolButton3: TToolButton;
  ToolButton4: TToolButton;
  ToolButton5: TToolButton;
  ToolButton6: TToolButton;
  ToolButton7: TToolButton;
  ToolButton8: TToolButton;
  ImageList1: TImageList;
  SkinData1: TSkinData;
  StatusBar1: TStatusBar;
  ToolButton19: TToolButton;
  ToolButton9: TToolButton;
  ToolButton15: TToolButton;
  ToolButton20: TToolButton;
  ToolButton21: TToolButton;
  ToolButton12: TToolButton;
  ToolButton14: TToolButton;
  ToolButton13: TToolButton;
  ToolButton10: TToolButton;
  Image2: TImage;
  procedure ToolButton2Click(Sender: TObject);
  procedure MteriKayt1Click(Sender: TObject);
  procedure Exit1Click(Sender: TObject);
  procedure ToolButton1Click(Sender: TObject);
  procedure MteriArama1Click(Sender: TObject);
  procedure ToolButton3Click(Sender: TObject);
  procedure ToolButton4Click(Sender: TObject);
  procedure ToolButton5Click(Sender: TObject);
  procedure ToolButton6Click(Sender: TObject);
  procedure ToolButton7Click(Sender: TObject);
  procedure ToolButton8Click(Sender: TObject);
  procedure FormShow(Sender: TObject);
private
```



```

    { Private declarations }
public
    { Public declarations }

end;

var
    Form1: TForm1;

implementation
uses Unit11,Kayitformu,Search,Unit4,Unit6, Unit7,Unit9;
{$R *.dfm}

procedure TForm1.ToolButton2Click(Sender: TObject);
begin
    form2.ShowModal;//Birinci formdan ikinci forma geçer...
end;

procedure TForm1.MteriKayt1Click(Sender: TObject);
begin
    form2.ShowModal;
end;

procedure TForm1.Exit1Click(Sender: TObject);
begin
    //Çıkmadan önce sor...
    if      messagedlg('Do      you      want      to      exit      the
Program?',mtinformation,[mbytes,mbno],0)=mrno then exit;
    Application.Terminate;
end;

procedure TForm1.ToolButton1Click(Sender: TObject);
begin
    form3.ShowModal;
end;

procedure TForm1.MteriArama1Click(Sender: TObject);
begin
    form3.ShowModal;
end;

procedure TForm1.ToolButton3Click(Sender: TObject);
begin
    if      messagedlg('Do      you      want      to      exit      the
Program?',mtinformation,[mbytes,mbno],0)=mrno then exit;
    Application.Terminate;
end;

```

```
procedure TForm1.ToolButton4Click(Sender: TObject);
begin
form4.ShowModal;
end;
```

```
procedure TForm1.ToolButton5Click(Sender: TObject);
begin
form6.ShowModal;
end;
```

```
procedure TForm1.ToolButton6Click(Sender: TObject);
begin
form7.showmodal;
end;
```

```
procedure TForm1.ToolButton7Click(Sender: TObject);
begin
form9.ShowModal;
end;
```

```
procedure TForm1.ToolButton8Click(Sender: TObject);
begin
form11.ShowModal;
end;
```

```
procedure TForm1.FormShow(Sender: TObject);
begin
StatusBar1.Panels[0].Text:=datetostr(date);
end;
```

```
end.
```

THE CODE S OF THE KAYIT FORM

```
unit Kayitformu;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, ComCtrls, StdCtrls, ExtCtrls, DB, ADODB, dxCntner, dxEditor,
dxExEdtr, dxEdLib, dxTL, dxDBCtrl, dxDBGrid, dxDBTLCl, dxGrClms, ExtDlgs,
Menus, Mask;
```

```
type
```

```
TForm2 = class(TForm)
Label2: TLabel;
```

```

Label3: TLabel;
Label4: TLabel;
Label1: TLabel;
Label5: TLabel;
Label6: TLabel;
Label8: TLabel;
Edit4: TEdit;
ComboBox1: TComboBox;
RadioButton2: TRadioButton;
RadioButton1: TRadioButton;
Edit3: TEdit;
Edit2: TEdit;
Edit1: TEdit;
Label7: TLabel;
Memo1: TMemo;
Memo2: TMemo;
Label9: TLabel;
Label10: TLabel;
Label11: TLabel;
Label12: TLabel;
Label13: TLabel;
Label15: TLabel;
Label16: TLabel;
Edit5: TEdit;
Edit6: TEdit;
Edit7: TEdit;
Edit8: TEdit;
Button1: TButton;
ADOQuery1: TADOQuery;
Label14: TLabel;
Edit9: TEdit;
dxDateEdit1: TdxDateEdit;
Timer1: TTimer;
Label17: TLabel;
Button2: TButton;
PopupMenu1: TPopupMenu;
MaskEdit1: TMaskEdit;
procedure FormActivate(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure FormKeyUp(Sender: TObject; var Key: Word;
  Shift: TShiftState);
procedure MaskEdit1Exit(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

```



```

var
  Form2: TForm2;

implementation

{$R *.dfm}
uses Rm,Search;
procedure TForm2.FormActivate(Sender: TObject);
begin
  memo1.Text:="";
  memo2.Text:="";

end;

procedure TForm2.Button1Click(Sender: TObject);
begin
  adoquery1.Close;
  adoquery1.SQL.Clear;
  adoquery1.sql.Text:='select * from Costumer_Info';
  adoquery1.Open;
  adoquery1.Insert;
  adoquery1.FieldByName('Name').AsString:=edit1.Text;
  adoquery1.FieldByName('Surname').AsString:=edit2.Text;
  adoquery1.FieldByName('Birthdate').AsString:=dxdateedit1.Text;
  adoquery1.FieldByName('BirthPlace').AsString:=edit3.Text;
  if RadioButton1.Checked=true then adoquery1.FieldByName('Sex').AsString:='Male'
else adoquery1.FieldByName('Sex').AsString:='Female';
  adoquery1.FieldByName('BloodGroup').AsString:=combobox1.Text;
  adoquery1.FieldByName('HomeAddress').AsString:=memo2.Text;
  adoquery1.FieldByName('HomeTel').AsString:=edit4.Text;
  adoquery1.FieldByName('Notes').AsString:=memo1.Text;
  adoquery1.FieldByName('Father_Name').AsString:=edit6.Text;
  adoquery1.FieldByName('Father_Tel').AsString:=edit5.Text;
  adoquery1.FieldByName('Mother_Name').AsString:=edit8.Text;
  adoquery1.FieldByName('Mother_Tel').AsString:=edit7.Text;
  if edit1.Text="" then // eğer çocuğun ismi yoksa

  begin
    messagedlg('Please Enter The Name of The Child.',mterror,[mbok],0);
    edit1.SetFocus;
    exit;
  end;
  adoquery1.Post;
  edit9.Text:=adoquery1.fieldbyname('Patient_Id').AsString;
  adoquery1.Close;
  label17.Visible:=True;
  timer1.Enabled:=true;

  form3.ADOQuery1.Close;

```

end;

```
procedure TForm2.Timer1Timer(Sender: TObject);
begin
label17.Visible:=false;
```

end;

```
procedure TForm2.Button2Click(Sender: TObject);
begin
edit9.Text:="";
edit1.Text:="";
edit2.Text:="";
edit3.Text:="";
edit4.Text:="";
edit5.Text:="";
edit6.Text:="";
edit7.Text:="";
edit8.Text:="";
dxDateEdit1.Text:="";
memo1.Text:="";
memo2.Text:="";
MaskEdit1.Text:="";
combobox1.Text:='--Rh+';
RadioButton1.Checked:=true;
radiobutton2.Checked:=false;
edit1.SetFocus;
end;
```

```
procedure TForm2.FormKeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then button1 click(sender);
end;
```

```
procedure TForm2.FormKeyUp(Sender: TObject; var Key: Word;
Shift: TShiftState);
var a:string;
begin
a:=vartostr(key);
case strtoint(a) of
27:close;
end;
end;
```

```
procedure TForm2.MaskEdit1Exit(Sender: TObject);
begin
dxdateedit1.Text:=maskedit1.Text;
end;
```

end.

THE CODES OF THE RM FORM

unit Rm;

interface

uses

SysUtils, Classes, DB, ADODB;

type

TDm = class(TDataModule)

ADOConnection1: TADOConnection;

ADOQuery1: TADOQuery;

private

{ Private declarations }

public

{ Public declarations }

procedure SorguR(str:string);

end;

var

Dm: TDm;

implementation

{ \$R *.dfm }

Procedure TDm.SorguR(str:string);

begin

with ADOQuery1 do begin

close;

sql.clear;

sql.text:=str;

open;

end;

end;

end.

THE CODES OF THE SEARCH FORM

```
unit Search;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, dxCntner, dxTL, dxDBCtrl, dxDBGrid, DB, ADODB,  
dxDBTLCL, dxGrClms, ExtCtrls, DBCtrls;
```

```
type
```

```
TForm3 = class(TForm)  
  Label1: TLabel;  
  Edit1: TEdit;  
  Label2: TLabel;  
  Label3: TLabel;  
  Edit2: TEdit;  
  Edit3: TEdit;  
  dxDBGrid1: TdxDBGrid;  
  ADOQuery1: TADOQuery;  
  DataSource1: TDataSource;  
  dxDBGrid1Patient_Id: TdxDBGridMaskColumn;  
  dxDBGrid1Name: TdxDBGridColumn;  
  dxDBGrid1Surname: TdxDBGridColumn;  
  dxDBGrid1Birthdate: TdxDBGridDateColumn;  
  dxDBGrid1BirthPlace: TdxDBGridColumn;  
  dxDBGrid1Sex: TdxDBGridColumn;  
  dxDBGrid1BloodGroup: TdxDBGridColumn;  
  dxDBGrid1HomeAddress: TdxDBGridMemoColumn;  
  dxDBGrid1HomeTel: TdxDBGridColumn;  
  dxDBGrid1Notes: TdxDBGridMemoColumn;  
  dxDBGrid1Father_Name: TdxDBGridColumn;  
  dxDBGrid1Father_Tel: TdxDBGridColumn;  
  dxDBGrid1Mother_Name: TdxDBGridColumn;  
  dxDBGrid1Mother_Tel: TdxDBGridColumn;  
  dxDBGrid1Picture: TdxDBGridColumn;  
  DBNavigator1: TDBNavigator;  
  procedure Edit1Change(Sender: TObject);  
  procedure Edit3Change(Sender: TObject);
```

```

procedure Edit2Change(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure FormShow(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form3: TForm3;

implementation

{$R *.dfm}
uses Rm;
procedure TForm3.Edit1Change(Sender: TObject);

var tt:String;
var ss:string;
var sss:string;
var ssss:string;
begin

tt:="";
if edit2.Text<>" then
begin
sss:='Name like' +tt+ edit2.Text +'%'+tt + sss + ' and+ ' ;
end else begin
sss:="";
end;
if edit3.Text<>" then
begin
SS:='Surname LIKE ' + tt + edit3.Text +'%'+ tt +ss+' and + ' ;
end else begin
ss:="";
end;
ssss:=ss+sss;
ADOQuery1.Close;
ADOQuery1.SQL.Clear;
ADOQuery1.SQL.Text:='select * from Costumer_Info where '+ssss+' Patient_Id LIKE
' + tt + edit1.Text +'%'+ tt;
ADOQuery1.open;
end;

procedure TForm3.Edit3Change(Sender: TObject);
var tt:String;
var ss:string;
var sss:string;
var ssss:string;

```

```

begin
tt:="";
if edit1.Text<>" then
begin
sss:='Patient_Id like' +tt+ edit1.Text +'%'+tt + sss +' and+ ';
end else begin
sss:="";
end;
if edit2.Text<>" then
begin
SS:='Name LIKE ' + tt + edit2.Text +'%'+ tt +ss+' and + ';
end else begin
ss:="";
end;
ssss:=ss+sss;
ADOQuery1.Close;
ADOQuery1.SQL.Clear;
ADOQuery1.SQL.Text:='select * from Costumer_Info where '+ssss+' Surname LIKE '
+ tt + edit3.Text +'%'+ tt;
ADOQuery1.open;
end;
procedure TForm3.Edit2Change(Sender: TObject);
var tt:String;
var ss:string;
var sss:string;
var ssss:string;
begin
tt:="";
if edit1.Text<>" then
begin
sss:='Patient_Id like' +tt+ edit1.Text +'%'+tt + sss +' and+ ';
end else begin
sss:="";
end;
if edit3.Text<>" then
begin
SS:='Surname LIKE ' + tt + edit3.Text +'%'+ tt +ss+' and + ';
end else begin
ss:="";
end;
ssss:=ss+sss;
ADOQuery1.Close;
ADOQuery1.SQL.Clear;
ADOQuery1.SQL.Text:='select * from Costumer_Info where '+ssss+' Name LIKE ' +
tt + edit2.Text +'%'+ tt;
ADOQuery1.open;
end;

procedure TForm3.FormClose(Sender: TObject; var Action: TCloseAction);
begin

```



```

edit1.Text:="";
edit2.Text:="";
edit3.Text:="";
end;

procedure TForm3.FormShow(Sender: TObject);
begin
adoquery1.Open;
end;

end.

```

THE CODES OF THE ILLNESS INFORMATION FORM

```

unit Unit10;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls;

type
  TForm10 = class(TForm)
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    DateTimePicker1: TDateTimePicker;
    Label7: TLabel;
    Label1: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Edit5: TEdit;
    Edit6: TEdit;
    Edit7: TEdit;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Edit8: TEdit;
    Edit9: TEdit;
    Edit10: TEdit;
    Memo1: TMemo;
    Label11: TLabel;
  end;

```

```

Edit11: TEdit;
Label12: TLabel;
Edit12: TEdit;
Label13: TLabel;
Edit13: TEdit;
Label14: TLabel;
Button1: TButton;
private
  { Private declarations }
public
  { Public declarations }
end;

```

```

var
  Form10: TForm10;

```

```

implementation

```

```

{$R *.dfm}

```

```

end.

```

THE CODES OF REPORT FORM

```

unit Unit9;

```

```

interface

```

```

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, dxCntner, dxEditor, dxExEdtr, dxEdLib, StdCtrls, ComCtrls,
  Buttons;

```

```

type

```

```

  TForm9 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Memo1: TMemo;
    Label5: TLabel;
    Label6: TLabel;
    PrintDialog1: TPrintDialog;
    Button1: TButton;
    procedure FormActivate(Sender: TObject);

```

```

  private
    { Private declarations }
  public

```

```

    { Public declarations }
end;

var
    Form9: TForm9;

implementation

{$R *.dfm}

procedure TForm9.FormActivate(Sender: TObject);
begin
    label3.Caption:=datetostr(date);
    memo1.Text:="";
end;

end.

```

THE CODES OF THE CHOOSE PATIENT

```

unit Unit4;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, DB, ADODB, dxDBTLCl, dxGrClms, dxDBGrid, dxTL, dxDBCtrl,
    dxCntner, StdCtrls;

type
    TForm4 = class(TForm)
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        Edit1: TEdit;
        Edit2: TEdit;
        Edit3: TEdit;
        dxDBGrid1: TdxDBGrid;
        ADOQuery1: TADOQuery;
        DataSource1: TDataSource;
        Button1: TButton;
        dxDBGrid1Patient_Id: TdxDBGridMaskColumn;
        dxDBGrid1Name: TdxDBGridColumn;
        dxDBGrid1Surname: TdxDBGridColumn;
        dxDBGrid1Birthdate: TdxDBGridDateColumn;
        dxDBGrid1BirthPlace: TdxDBGridColumn;
        dxDBGrid1Sex: TdxDBGridColumn;
        dxDBGrid1BloodGroup: TdxDBGridColumn;
    end;

```

```

dxDBGrid1HomeAddress: TdxDBGridMemoColumn;
dxDBGrid1HomeTel: TdxDBGridColumn;
dxDBGrid1Notes: TdxDBGridMemoColumn;
dxDBGrid1Father_Name: TdxDBGridColumn;
dxDBGrid1Father_Tel: TdxDBGridColumn;
dxDBGrid1Mother_Name: TdxDBGridColumn;
dxDBGrid1Mother_Tel: TdxDBGridColumn;
dxDBGrid1Picture: TdxDBGridColumn;
Edit4: TEdit;
procedure Edit1Change(Sender: TObject);
procedure Edit2Change(Sender: TObject);
procedure Edit3Change(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure FormKeyPress(Sender: TObject; var Key: Char);
procedure dxDBGrid1DbClick(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form4: TForm4;

implementation
uses Rm,Unit5;
{$R *.dfm}

procedure TForm4.Edit1Change(Sender: TObject);
var tt:String;
var ss:string;
var sss:string;
var ssss:string;
begin

  tt:="";
  if edit2.Text<>" then
  begin
    sss:='Name like' +tt+ edit2.Text +'%'+tt + sss +' and+ ' ;
  end else begin
    sss:="";
  end;
  if edit3.Text<>" then
  begin
    SS:='Surname LIKE ' + tt + edit3.Text +'%'+ tt +ss+' and + ' ;
  end else begin
    ss:="";
  end;
end;

```



```

ssss:=ss+sss;
ADOQuery1.Close;
ADOQuery1.SQL.Clear;
ADOQuery1.SQL.Text:='select * from Costumer_Info where '+ssss+' Patient_Id LIKE
' + tt + edit1.Text + '%' + tt;
ADOQuery1.open;
end;
procedure TForm4.Edit2Change(Sender: TObject);
var tt:String;
var ss:string;
var sss:string;
var ssss:string;
begin
tt:="";
if edit1.Text<>" then
begin
sss:='Patient_Id like' +tt+ edit1.Text + '%' + tt + ss + ' and + ';
end else begin
sss:="";
end;
if edit3.Text<>" then
begin
SS:='Surname LIKE ' + tt + edit3.Text + '%' + tt + ss + ' and + ';
end else begin
ss:="";
end;
ssss:=ss+sss;
ADOQuery1.Close;
ADOQuery1.SQL.Clear;
ADOQuery1.SQL.Text:='select * from Costumer_Info where '+ssss+' Name LIKE ' +
tt + edit2.Text + '%' + tt;
ADOQuery1.open;
end;

procedure TForm4.Edit3Change(Sender: TObject);
var tt:String;
var ss:string;
var sss:string;
var ssss:string;
begin
tt:="";
if edit1.Text<>" then
begin
sss:='Patient_Id like' +tt+ edit1.Text + '%' + tt + ss + ' and + ';
end else begin
sss:="";
end;
if edit2.Text<>" then
begin
SS:='Name LIKE ' + tt + edit2.Text + '%' + tt + ss + ' and + ';

```

```

end else begin
ss:="";
end;
ssss:=ss+ss;
ADOQuery1.Close;
ADOQuery1.SQL.Clear;
ADOQuery1.SQL.Text:='select * from Costumer_Info where '+ssss+' Surname LIKE '
+ tt + edit3.Text + '%' + tt;
ADOQuery1.open;
end;

procedure TForm4.Button1Click(Sender: TObject);
begin

if edit1.Text="" then
begin
messagedlg('Please Enter The Patient Id',mterror,[mbok],0);
edit1.SetFocus;
exit;
end else begin

adoquery1.Close;
adoquery1.SQL.Clear;
adoquery1.SQL.Text:='select * from Costumer_Info where Patient_Id
='+edit1.Text+'';
adoquery1.Open;
edit4.Text:=edit1.Text;
form5.ShowModal;

end;

end;

procedure TForm4.FormKeyPress(Sender: TObject; var Key: Char);
begin
if (key=#13) then button1click(sender);
end;

procedure TForm4.dxDDBGrid1DbClick(Sender: TObject);
begin
edit1.Text:=adoquery1.fieldbyname('Patient_Id').AsString;
edit4.Text:=adoquery1.fieldbyname('Patient_Id').AsString;
form5.edit7.Text:=form4.Edit4.Text;
form5.ShowModal;
end;

procedure TForm4.FormShow(Sender: TObject);
begin
adoquery1.Close;
adoquery1.SQL.Clear;

```

```

adoquery1.SQL.Text:='select * from Costumer_Info';
adoquery1.Open;
end;

```

```

procedure TForm4.FormClose(Sender: TObject; var Action: TCloseAction);
begin
edit2.Text:="";
edit1.Text:="";
edit3.Text:="";
adoquery1.Close;
end;

end.

```

THE CODES OF THE INOCULATE

```

unit Unit6;

```

```

interface

```

```

uses

```

```

  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, dxDBTLCL, dxGrClms, dxTL, dxDBCtrl, dxDBGrid, DB, ADODB,
  ExtCtrls, DBCtrls, dxCntner;

```

```

type

```

```

  TForm6 = class(TForm)
    dxDBGrid1: TdxDBGrid;
    DBNavigator1: TDBNavigator;
    ADOQuery1: TADOQuery;
    DataSource1: TDataSource;
    dxDBGrid1Asi_Adi: TdxDBGridColumn;
    dxDBGrid1Asi_Yasi: TdxDBGridColumn;
    dxDBGrid1Alians: TdxDBGridColumn;
    dxDBGrid1Aktif: TdxDBGridCheckColumn;

```

```

  private

```

```

    { Private declarations }

```

```

  public

```

```

    { Public declarations }

```

```

  end;

```

```

var

```

```

  Form6: TForm6;

```

```

implementation

```

```
uses Rm;  
{ $R *.dfm }
```

```
end.
```

THE CODES OF THE INOCULATIONS INFO

```
unit Unit5;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, DB, ADODB, dxCntner, dxTL, dxDBCtrI, dxDBGrid,  
ComCtrls, dxDBTLCl, dxGrClms, dxEditor, dxExEdtr, dxEdLib;
```

```
type
```

```
TForm5 = class(TForm)  
    Label1: TLabel;  
    Label2: TLabel;  
    Label3: TLabel;  
    Label4: TLabel;  
    Label5: TLabel;  
    Label6: TLabel;  
    Edit7: TEdit;  
    Label7: TLabel;  
    Label8: TLabel;  
    Label9: TLabel;  
    Label11: TLabel;  
    Label12: TLabel;  
    Label13: TLabel;  
    Edit1: TEdit;  
    ComboBox1: TComboBox;  
    Label10: TLabel;  
    Edit2: TEdit;  
    Label14: TLabel;  
    Edit3: TEdit;  
    Label15: TLabel;  
    Memo1: TMemo;  
    Label16: TLabel;  
    Button1: TButton;  
    ADOQuery1: TADOQuery;  
    DataSource1: TDataSource;  
    ADOQuery2: TADOQuery;  
    RadioButton1: TRadioButton;  
    RadioButton2: TRadioButton;
```



```

dxDateEdit1: TdxDateEdit;
Label17: TLabel;
dxDBGrid1: TdxDBGrid;
dxDBGrid1Asi_Adi: TdxDBGridColumn;
dxDBGrid1Asi_Yasi: TdxDBGridColumn;
dxDBGrid1Tarih: TdxDBGridDateColumn;
dxDBGrid1Called: TdxDBGridColumn;
dxDBGrid1YapId: TdxDBGridColumn;
dxDBGrid1Id: TdxDBGridColumn;
dxDBGrid1ATr: TdxDBGridColumn;
Edit4: TEdit;
Button2: TButton;
procedure FormShow(Sender: TObject);
procedure ComboBox1Click(Sender: TObject);
procedure ComboBox1Enter(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure Button2Click(Sender: TObject);
procedure FormActivate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

```

```

var
  Form5: TForm5;

```

```

implementation
Uses Rm,Unit4;
{$R *.dfm}

```

```

procedure TForm5.FormShow(Sender: TObject);
begin
  edit7.Text:=form4.Edit1.Text;
  form4.edit1.Text:="";
  form4.edit2.Text:="";
  form4.edit3.Text:="";
  dm.adoquery1.Close;
  dm.adoquery1.SQL.Clear;
  dm.adoquery1.SQL.Text:='select      *      from      Costumer_Info      where
Patient Id= '+edit7.Text+';
  dm.adoquery1.Open;
  label12.Caption:=dm.adoquery1.fieldbyname('Name').AsString;
  label13.Caption:=dm.ADOQuery1.fieldbyname('Surname').AsString;
  label7.Caption:=dm.ADOQuery1.fieldbyname('HomeTel').AsString;
  label8.Caption:=dm.ADOQuery1.fieldbyname('Mother_Tel').AsString;
  label9.Caption:=dm.ADOQuery1.fieldbyname('Father_Tel').AsString;
  edit4.Text:=dm.ADOQuery1.fieldbyname('Birthdate').AsString;
  label11.Caption:='('+label12.Caption+')';
  memo1.Text:=dm.ADOQuery1.fieldbyname('Notes').AsString;

```

```

dm.ADOQuery1.Close;
adoquery1.Close;
adoquery1.SQL.Clear;
adoquery1.SQL.Text:='select * from asilar where Id="'+edit7.Text+"'";
adoquery1.Open;
button2.Click;
end;

procedure TForm5.ComboBox1Click(Sender: TObject);
begin
    dm.adoquery1.Close;
    dm.adoquery1.SQL.Clear;
    dm.adoquery1.SQL.Text:='select *from Asi where alians="'+combobox1.Text+"'";
    dm.adoquery1.Open;
    Edit2.Text:=dm.adoquery1.fieldbyname('Asi_Adi').AsString;
    edit3.Text:=dm.ADOQuery1.fieldbyname('Asi_Yasi').AsString;
end;

procedure TForm5.ComboBox1Enter(Sender: TObject);
var a:integer;
begin
    combobox1.Clear;
    dm.adoquery1.Close;
    dm.adoquery1.SQL.Clear;
    dm.adoquery1.SQL.Text:='SELECT Asi.Asi_Adi, Asi.aliens '+
    ' FROM Asi '+
    ' GROUP BY Asi.Asi_Adi, Asi.aliens '+
    ' HAVING (((Asi.aliens) Is Not Null))';
    dm.adoquery1.Open;
    dm.adoquery1.First;
    for a:=0 to dm.adoquery1.RecordCount-1 do
    begin
        combobox1.Items.Add(dm.adoquery1.fieldbyname('aliens').AsString);
        dm.adoquery1.Next;
    end;
end;

procedure TForm5.Button1Click(Sender: TObject);
begin
    adoquery2.Close;
    adoquery2.SQL.Clear;
    adoquery2.SQL.Text:='Select * from asilar';
    adoquery2.Open;
    adoquery2.Insert;
    adoquery2.FieldName('Asi_Adi').AsString:=edit2.Text;
    adoquery2.FieldName('Asi_Yasi').AsString:=edit3.Text;
    if RadioButton1.Checked=true then
    begin
        adoquery2.FieldName('Called').AsString:='Called';
        adoquery2.FieldName('Yapıldı').AsString:='-----';
    end;
end;

```

```

end else begin
adoquery2.FieldByName('Yapıldı').AsString:='Done It';
adoquery2.FieldByName('Called').AsString:='-----';
end;
if dxdateedit1.Text="" then
begin
messagedlg('You have to choose date',mtwarning,[mbok],0);

dxdateedit1.SetFocus;
exit;
end else begin
adoquery2.FieldByName('Tarih').AsString:=dxdateedit1.Text;
end;
adoquery2.FieldByName('Id').AsString:=edit7.Text;
adoquery2.FieldByName('Aşı Türü').AsString:=combobox1.Text;
adoquery2.Post;

adoquery2.Close;
adoquery1.Close;
adoquery1.SQL.Clear;
adoquery1.SQL.Text:='select * from Asilar where Id="'+edit7.text+"'";
adoquery1.Open;
end;

procedure TForm5.FormClose(Sender: TObject; var Action: TCloseAction);
begin
edit2.Text:="";
edit3.Text:="";
combobox1.Text:="";
end;
function yashesap(gelen:string):string;
var
yas,sonuc,a,s1,s2:string;
x,i,y:integer;
begin
x:=7;
while(x<11) do begin
s1:=s1+gelen[x];
x:=x+1;
end;
a:=datetostr(date);
i:=7;
while(i<11) do begin
s2:=s2+a[i];
i:=i+1;
end;
yas:=inttostr(strtoint(s2)-strtoint(s1));
sonuc:=yas;
if strtoint(yas)>1 then
begin
form5.Edit1.Text:=yas+' years old';

```

```

end else begin
form5.Edit1.Text:=yas+' year old';
end;
end;

```

```

procedure TForm5.Button2Click(Sender: TObject);
begin
yashesap(edit4.Text);
end;

```

```

procedure TForm5.FormActivate(Sender: TObject);
begin
dxDateEdit1.Date:=date;
end;

```

```

end.

```

THE CODES OF THE CALENDER

```

unit Unit7;

```

```

interface

```

```

uses

```

```

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, ComCtrls, dxCntner, dxTL, dxDBCtrl, dxDBGrid, DB,
ADODB, dxDBTLCL, dxGrClms;

```

```

type

```

```

TForm7 = class(TForm)
Button1: TButton;
Edit1: TEdit;
ADOQuery1: TADOQuery;
DataSource1: TDataSource;
MonthCalendar1: TMonthCalendar;
dxDBGrid1: TdxDBGrid;
dxDBGrid1id: TdxDBGridColumn;
dxDBGrid1Date: TdxDBGridDateColumn;
dxDBGrid1explanation: TdxDBGridMemoColumn;
dxDBGrid1name: TdxDBGridColumn;
dxDBGrid1surname: TdxDBGridColumn;
procedure Button1Click(Sender: TObject);
procedure FormShow(Sender: TObject);
private

```



```

    { Private declarations }
public
    { Public declarations }
end;

var
    Form7: TForm7;

implementation
    uses Rm,Unit8;
    {$R *.dfm}

procedure TForm7.Button1Click(Sender: TObject);
begin

    edit1.Text:=datetostr(monthcalendar1.Date);
    form8.ShowModal;
end;

procedure TForm7.FormShow(Sender: TObject);
begin
    adoquery1.Open;
end;

end.

```

THE CODES OF ADD MESSAGE FORM

```

unit Unit8;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls, DB, ADODB;

type
    TForm8 = class(TForm)
        Memo1: TMemo;
        Label1: TLabel;
        Edit1: TEdit;
        Label2: TLabel;
        Edit2: TEdit;
        Button1: TButton;
        Label3: TLabel;
        ADOQuery1: TADOQuery;

```

```

Edit3: TEdit;
Edit4: TEdit;
Label4: TLabel;
procedure FormActivate(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure Button1Click(Sender: TObject);
procedure Edit1Exit(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form8: TForm8;

implementation
uses Rm,Unit7;
{$R *.dfm}

procedure TForm8.FormActivate(Sender: TObject);
begin
  memo1.Text:="";
  edit1.SetFocus;
end;

procedure TForm8.FormShow(Sender: TObject);
begin
  edit2.Text:=form7.Edit1.Text;
  label3.Caption:=edit2.Text;
end;

procedure TForm8.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  edit1.Text:="";
  memo1.Text:="";
  edit2.Text:="";
  form7.ADOQuery1.close;
  form7.ADOQuery1.SQL.Clear;
  form7.ADOQuery1.SQL.Text:='select * from calendar';
  form7.ADOQuery1.Open;
end;

procedure TForm8.Button1Click(Sender: TObject);
begin
  dm.ADOQuery1.Close;
  dm.ADOQuery1.SQL.Clear;
  dm.ADOQuery1.SQL.Text:='select * from calendar';

```

```

dm.adoquery1.Open;
dm.ADOQuery1.Insert;
if edit1.Text="" then
begin
messedlg('Please enter the Id number',mtinformation,[mbok],0);
edit1.SetFocus;
exit;
end;
dm.ADOQuery1.FieldName('id').AsString:=edit1.Text;
dm.ADOQuery1.FieldName('Date').AsString:=edit2.Text;
dm.ADOQuery1.FieldName('explanation').AsString:=memo1.Text;
if (edit3.Text="" or (edit4.Text="" then
begin
messedlg('Please Enter The Correct Patient Id...',mtinformation,[mbok],0);
edit1.Text:="";
edit1.SetFocus;
exit;
end;
dm.ADOQuery1.FieldName('name').AsString:=edit3.Text;
dm.ADOQuery1.FieldName('surname').AsString:=edit4.Text;
dm.ADOQuery1.Post;

dm.ADOQuery1.Close;
showmessage('Randevu işleminiz tamamlanmıştır');
edit1.Text:="";
edit2.Text:="";
memo1.Text:="";
form8.Close;
end;

procedure TForm8.Edit1Exit(Sender: TObject);
begin
if edit1.Text<>"" then
begin
adoquery1.Close;
adoquery1.SQL.Clear;

adoquery1.SQL.Text:='select * from Costumer_Info where Patient_Id='+edit1.Text+";
adoquery1.Open;
edit3.Text:=adoquery1.fieldbyname('name').AsString;
edit4.Text:=adoquery1.fieldbyname('surname').AsString;
adoquery1.Close;
label4.Visible:=True;
label4.Caption:=edit3.Text+' '+edit4.Text
end;
end;
end.

```