NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

MEDICINE STOCK CONTROL MANAGAMENT

Graduation Project
COM-400

Student: Yasin Ayhan

Supervisor: Ümit İlhan

Nicossia - 2003

ACKNOWLEDGEMENT

First of all I would like to thank Mr. UMIT ILHAN for his endless and untiring support and help and his persistence, in the course of the preparation of this project.

Under his guidance, I have overcome many difficulties that I faced during the various stages of the preparation of this project.

Finally, I would like to thank my family, especially my father who name is Mr NECMETTIN AYHAN and my brother who name is Mr UGUR AYHAN. Their love and guidance saw me through doubtul times. Their never-ending belief in me and their encouragement has been a crucial and a very strong pillar that has held me together

ABSTRACT

As the information age has effected every aspect of our life, the need for computerizing many information systems has raised.

Once of the important branches that are effected by information revolution is the computer programming languages.

This project is concerned about using compueter program in Pharmacy management system . It is written using Visual Basic 6.0 programming language and used Microsoft Access Database language for databases. Visual Basic is one of the best and easy programming languages.

This project is accomlete Pharmacy management program, that covers all services needed in most Pharmacy, such as computer related information,madicine, goods and many other Pharmacy management related services.

Before coming to this point, this project has gone through some important steps;

- First one was the requirements definition for which I had to go to some Pharmacy and study their systems.

- The second steps were designing the system and sofware that is intended to serve an integrated Pharmacy management system.

- The final steps was the implementation of the design on the computer using Visual Basic Language.

TABLE OF CONTENTS

NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

MEDICINE STOCK CONTROL MANAGAMENT

Graduation Project
COM-400

Student: Yasin Ayhan

Supervisor: Ümit İlhan

Nicossia - 2003

ACKNOWLEDGEMENT

First of all I would like to thank Mr. UMIT ILHAN for his endless and untiring support and help and his persistence, in the course of the preparation of this project.

Under his guidance, I have overcome many difficulties that I faced during the various stages of the preparation of this project.

Finally, I would like to thank my family, especially my father who name is Mr NECMETTIN AYHAN and my brother who name is Mr UGUR AYHAN.Their love and guidance saw me through doubtul times. Their never-ending belief in me and their encouragement has been a crucial and a very strong pillar that has held me together

ABSTRACT

As the information age has effected every aspect of our life, the need for computerizing many information systems has raised.

Once of the important branches that are effected by information revolution is the computer programming languages.

This project is concerned about using compueter program in Pharmacy management system . It is written using Visual Basic 6.0 programming language and used Microsoft Access Database language for databases. Visual Basic is one of the best and easy programming languages.

This project is accomlete Pharmacy management program, that covers all services needed in most Pharmacy, such as computer related information,madicine, goods and many other Pharmacy management related services.

Before coming to this point, this project has gone through some important steps;

- First one was the requirements definition for which I had to go to some Pharmacy and study their systems.

- The second steps were designing the system and sofware that is intended to serve an integrated Pharmacy management system.

- The final steps was the implementation of the design on the computer using Visual Basic Language.

TABLE OF CONTENTS

INTRODUCTION

Visual Basic is a Microsoft Windows programming Language. Visual Basic programs are created in an Integrated Development Environment (IDE). The IDE allows the programmer to create, run and debug Visual Basic programs conveniently. IDEs allow a programmer to create working programs in a fraction of the time that it would normally take to code programs without using IDEs. The process of rapidly creating an application is typically referred to as Rapid Application Development(RAD). Visual Basic is the world's most widely used RAD language.

Visual Basic is derived from the BASIC programming language. Visual Basic is a distinctly different language providing powerfull features such as graphical user interfaces, even handling, access to the Win32 API, object-oriented features, error handling, structured programming, and much more.

The Visual Basic IDE allows Windows programs to be created without the need for the programmer to be a Windows programming export.

Microsoft provides several version of Visual Basic, namely the Learning Edition, the Professional Edition and the Enterprice Edition. The Learning Edition provides fundemantal programming capabilities than the Learning Edition and is the choice of many programmers to write Visual Basic applications. The Enterprice Edition is used for developing large-scale computing systems that meet the needs of substandial organizations.

Visual Basic is an interpreted language. However, the professional and Enterprice Edition allows Visual Basic code to be compiled to native code.

Visual Basic evolved from BASIC(Beginner's All purpose Symbolic Instruction Code). Basic was developed in the mid 1960's by Professors John Kemeny and Thomas Kurtz of Darthmouth College as a language for writing simple programs. BASIC's primary purpose was to help people learn how to program.

The widespread use of BASIC with various types of computers (sometimes called hardware platforms) led to many enhancements to the language. With the development of the Microsoft windows graphical user interface (GUI) in the late

1980s and the early 1990s, the natural evolution of BASIC was Visual Basic, which was created by Microsoft Corporation in 1991.

Until Visual Basic appeared, develoing Microsoft Windows-based applications was a diffucult and cumbersome process. Visual Basic greatly simplifies Windows application development. Since 1991 six versions have been released, with the latest-Visual Basic 6-appearing in september 1998.

After a brief explanation about the Visual Basic 6.0 and the developing layers, I hope that you will find the necessary information that you need all about the Visual Basic even if you are a text based programmer.

CHAPTER1
Visual Basic Program

1.1.VB Advantages

So what makes VB a great programming language? The answer is simply that VB provides more of the actual code for a programmer than any other non-visual programming language.

If you've ever programmed in the older BASIC or other command line programming

language, then you'll remember that the programmer had to write the code for the entire user interface. Todays windows, buttons, lists, and other application features such as menus were not built-in to the BASIC programming language. Programmers had to create the code for these features on their own!

As much as 80% of a programmer's time was spent writing code to create the user interface to his applications (the visual interface). To eliminate this huge drain on a programmer's time, Microsoft has provided Visual Basic with the built-in capability to create the user interface using nothing more than a mouse!

This built-in interface creation capability has had the further benefit of standardizing on the user interface to Windows applications. Today, users can move from one Windows program to another and see the same basic interface tools to work with - allowing them to concentrate solely on the unique capabilities of the application.

The bottom line is that you can create an entire application shell (the user interface) very quickly and then spend most of your time working on the features which differentiate you application from its competition.

1.2.Very First Visual Basic Program

Visual Basic lets you build a complete and functional Windows application by dropping a bunch of controls on a form and writing some code that executes when something happens to those controls or to the form itself. For instance, you can write code that executes when a form loads or unloads or when the user resizes it.

Likewise, you can write code that executes when the user clicks on a control or types while the control has the input focus.

This programming paradigm is also known as *event-driven programming* because your application is made up of several event procedures executed in an order that's dependent on what happens at run time. The order of execution can't, in general, be foreseen when the program is under construction. This programming model contrasts with the procedural approach, which was dominant in the old days.

This section offers a quick review of the event-driven model and uses a sample application as a context for introducing Visual Basic's intrinsic controls, with their properties, methods, and events. This sample application, a very simple one, queries the user for the lengths of the two sides of a rectangle, evaluates its perimeter and area, and displays the results to the user. Like all lengthy code examples and programs illustrated in this book, this application is included on the companion CD.

1.3. The Form Object

After this long introductory description of properties, methods, and events that are common to most Visual Basic objects, it's time to see the particular features of all of them individually. The most important visible object is undoubtedly the Form object because you can't display any control without a parent Form. Conversely, you can write some moderately useful applications using only forms that have no controls on them. In this section, I'll show a number of examples that are centered on forms' singular features.

You create a new form at design time using the Add Form command from the Project menu or by clicking on the corresponding icon on the standard toolbar. You can create forms from scratch, or you can take advantage of the many form templates provided by Visual Basic 6. If you don't see the dialog box shown in Figure 2-7, invoke the Options command from the Tools menu, click the Environment tab, and select the topmost check box on the right.

Feel free to create new form templates when you need them. A form template doesn't necessarily have to be a complex form with many controls on it. Even an empty form with a group of properties carefully set can save you some precious time. For example, see the Dialog Form template provided by Visual Basic. To produce your custom form templates, you just have to create a form, add any necessary controls and code, and then save it in the \Template\Forms directory. (The complete path of Visual Basic's template directory can be read and modified in the Environment tab of the Options dialog box.)
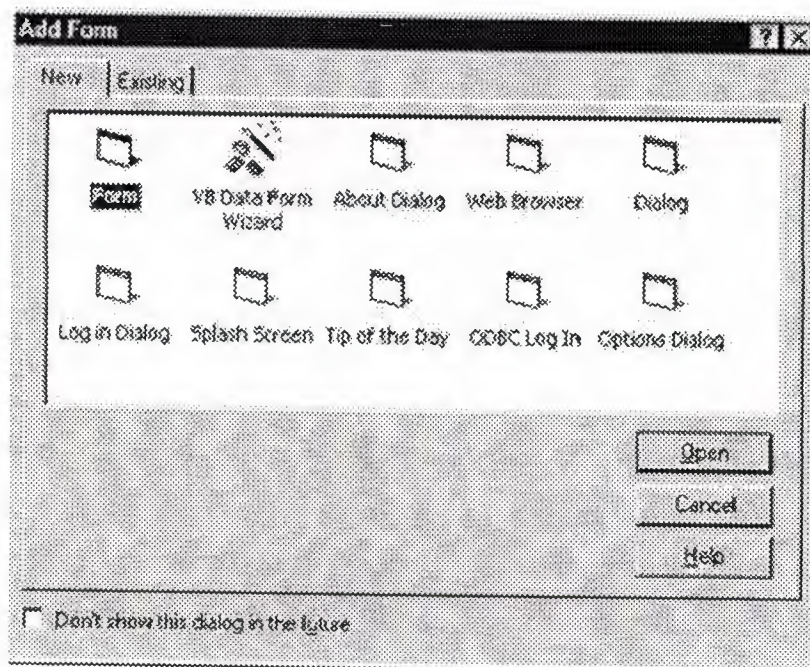
1.4.Adding Controls to a Form

We're ready to get practical. Launch the Visual Basic IDE, and select a Standard EXE project. You should have a blank form near the center of the work area. More accurately, you have a *form* designer, which you use to define the appearance of the main window of your application. You can also create other forms, if you need them, and you can create other objects as well, using different designers (the

5

UserControl and UserDocument designers, for example). Other chapters of this book are devoted to such designers.

One of the greatest strengths of the Visual Basic language is that programmers can design an application and then test it without leaving the environment. But you should be aware that designing and testing a program are two completely different tasks. At *design time,* you create your forms and other visible objects, set their properties, and write code in their event procedures. Conversely, at *run time* you monitor the effects of your programming efforts: What you see on your screen is, more or less, what your end users will see. At run time, you can't invoke the form designer, and you have only a limited ability to modify the code you have written at design time. For instance, you can modify existing statements and add new ones, but you can't add new procedures, forms, or controls. On the other hand, at run time you can use some diagnostic tools that aren't available at design time because they would make no sense in that context (for example, the Locals, the Watches, and the Call Stack windows).

To create one or more controls on a form's surface, you select the control type that you want from the Toolbox window, click on the form, and drag the mouse cursor until the control has the size and shape you want. (Not all controls are resizable. Some, such as the Timer control, will allow you to drag but will return to their original size and shape when you release the mouse button.) Alternatively, you can place a control on the form's surface by double-clicking its icon in the Toolbox: this action creates a control in the center of the form. Regardless of the method you follow, you can then move and resize the control on the form using the mouse.

TIP

If you need to create multiple controls of the same type, you can follow this three-step procedure: First, click on the control's icon on the Toolbox window while you keep the Ctrl key pressed. Next, draw multiple controls by clicking the left button on the form's surface and then dragging the cursor. Finally, when you're finished creating controls, press

the Escape key or click the Pointer icon in the upper left corner of the Toolbox.

To complete our Rectangle sample application, we need four TextBox controls— two for entering the rectangle's width and height and two for showing the resulting perimeter and area, as shown in Figure 1-8. Even if they aren't strictly required from an operational point of view, we also need four Label controls for clarifying the purpose of each TextBox control. Finally we add a CommandButton control named *Evaluate* that starts the computation and shows the results.

Place these controls on the form, and then move and resize them as depicted in Figure 1-8. Don't worry too much if the controls aren't perfectly aligned because you can later move and resize them using the mouse or using the commands in the Format menu.
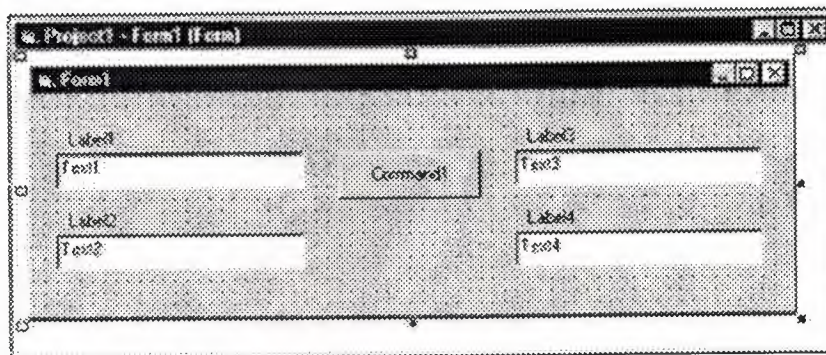
Figure 1-8 The Rectangle Demo form at design time, soon after the placement of its controls.

1.5. Setting Properties of Controls

Each control is characterized by a set of properties that define its behavior and appearance. For instance, Label controls expose a *Caption* property that corresponds to the character string displayed on the control itself, and a *BorderStyle* property that affects the appearance of a border around the label. The TextBox control's most

7

important property is *Text*, which corresponds to the string of characters that appears within the control itself and that can be edited by the user.

In all cases, you can modify one or more properties of a control by selecting the control in the form designer and then pressing F4 to show the Properties window. You can scroll through the contents of the Properties window until the property you're interested in becomes visible. You can then select it and enter a new value.

Using this procedure, you can modify the *Caption* property of all four Label controls to *&Width*, *&Height*, *&Perimeter*, and *&Area*, respectively. You will note that the ampersand character doesn't appear on the control and that its effect is to underline the character that follows it. This operation actually creates a *hot key* and associates it with the control. When a control is associated with a hot key, the user can quickly move the focus to the control by pressing an Alt+*x* key combination, as you normally do within most Windows applications. Notice that only controls exposing a *Caption* property can be associated with a hot key. Such controls include the Label, Frame, CommandButton, OptionButton, and CheckBox.

A quick way to select all the controls on a form is to click anywhere on the form and press the Ctrl+A key combination. After selecting all controls, you can deselect a few of them by clicking on them while pressing the Shift or Ctrl key. Note that this shortcut doesn't selectcontrols that are contained in other controls.When you select a group of controls and then press the F4 key, the Properties windowdisplays only the properties that are common to all the selected controls. The onlyproperties that are exposed by any control are *Left*, *Top*, *Width*, and *Height*. If youselect a group of controls that display a string of characters, such as the TextBox,Label, and CommandButton controls in our Rectangle example, the *Font* property isalso available and can therefore be selected. When you double-click on the *Font* item in the Properties window, a Font dialog box appears. Let's select a Tahoma font and set its size to 11 points.
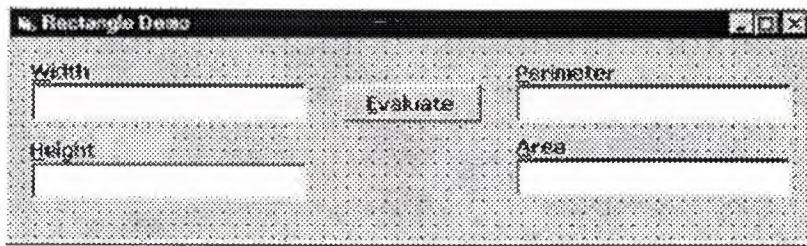
Figure 1-9. The Rectangle Demo form at design time, after setting the controls' properties.

TIP

When a control is created from the Toolbox, its *Font* property reflects he font of the parent form. For this reason, you can often avoid individual font settings by changing the form's *Font* property before placing any controls on the form itself.

1.6.Naming Controls

One property that every control has and that's very important to Visual Basic programmers is the *Name* property. This is the string of characters that identifies the control in code. This property can't be an empty string, and you can't have two or more controls on a form with the same name. The special nature of this property is indirectly confirmed by the fact that it appears as (Name) in the Properties window, where the initial parenthesis serves to move it to the beginning of the property list.

When you create a control, Visual Basic assigns it a default name. For example, the first TextBox control that you place on the form is named *Text1*, the second one is named *Text2*, and so forth. Similarly, the first Label control is named *Label1*, and the first CommandButton control is named *Command1*. This default naming scheme frees you from having to invent a new, unique name each time you create a control. Notice that the *Caption* property of Label and CommandButton controls, as well as

9

the *Text* property of TextBox controls, initially reflect the control's *Name* property, but the two properties are independent of each other. In fact, you have just modified the *Caption* and *Text* properties of the controls in the Rectangle Demo form without affecting their *Name* properties.

Because the *Name* property identifies the control in code, it's a good habit to modify it so that it conveys the meaning of the control itself. This is as important as selecting meaningful names for your variables. In a sense, most controls on a form are special variables whose contents are entered directly by the user.

Microsoft suggests that you always use the same three-letter prefix for all the controls of a given class. The control classes and their recommended prefixes are shown in

Table 1-1.Table 1-1. Standard three-letter prefixes for forms and all intrinsic controls.

Control Class	Prefix	Control Class	Prefix
CommandButton	cmd	Data	dat
TextBox	txt	HScrollBar	hsb
Label	lbl	VScrollBar	vsb
PictureBox	pic	DriveListBox	drv
OptionButton	opt	DirListBox	dir
CheckBox	chk	FileListBox	fil
ComboBox	cbo	Line	lin
ListBox	lst	Shape	shp
Timer	tmr	OLE	ole
Frame	fra	Form	Frm

For instance, you should prefix the name of a TextBox control with txt, the name of a Label control with lbl, and the name of a CommandButton control with cmd. Forms should also follow this convention, and the name of a form should be prefixed with the *frm* string. This convention makes a lot of sense because it lets you deduce both the control's type and meaning from its name. This book sticks to this naming convention, especially for more complex examples when code readability is at stake.

In our example, we will rename the Text1 through Text4 controls as txtWidth, txtHeight, txtPerimeter, and txtArea respectively. The Command1 control will be renamed cmdEvaluate, and the four Label1 through Label4 controls will be renamed lblWidth, lblHeight, lblPerimeter, and lblArea, respectively. However, please note that Label controls are seldom referred to in code, so in most cases you can leave their

1.7. Adding Code

Up to this point, you have created and refined the user interface of your program and created an application that in principle can be run. (Press F5 and run it to convince yourself that it indeed works.) But you don't have a useful application yet. To turn your pretty but useless program into your first working application, you need to add some code. More precisely, you have to add some code in the *Click* event of the cmdEvaluate control. This event fires when the user clicks on the Evaluate button or presses its associated hot key (the Alt+E key combination, in this case).

To write code within the *Click* event, you just select the cmdEvaluate control and then press the F7 key, or right-click on it and then invoke the View Code command from the pop-up menu. Or you simply double-click on the control using the left mouse button. In all cases, the code editor window appears, with the flashing cursor located between the following two lines of code:

```
Private Sub cmdEvaluate_Click()
End Sub
```

Visual Basic has prepared the template of the *Click* event procedure for you, and you have to add one or more lines of code between the *Sub* and *End Sub* statements. In this simple program, you need to extract the values stored in the txtWidth and txtHeight controls, use them to compute the rectangle's perimeter and area, and assign the results to the txtPerimeter and txtArea controls respectively:

```
Private Sub cmdEvaluate_Click()
    ' Declare two floating point variables.
    Dim reWidth As Double, reHeight As Double
    ' Extract values from input TextBox controls.
    reWidth = CDbl(txtWidth.Text)
    reHeight = CDbl(txtHeight.Text)
    ' Evaluate results and assign to output text boxes.
    txtPerimeter.Text = CStr((reWidth + reHeight) * 2)
    txtArea.Text = CStr(reWidth * reHeight)
End Sub
```

1.8. Running and Debugging the Program

You're finally ready to run this sample program. You can start its execution in several ways: By invoking the Start command from the Run menu, by clicking the corresponding icon on the toolbar, or by pressing the F5 key. In all cases, you'll see the form designer disappear and be replaced (but not necessarily in the same position on the screen) by the real form. You can enter any value in the leftmost TextBox controls and then click on the Evaluate button (or press the Alt+E key combination) to see the calculated perimeter and area in the rightmost controls. When you're finished, end the program by closing its main (and only) form.

CAUTION

12

You can also stop any Visual Basic program running in the environment by invoking the End command from the Run menu, but in general this isn't a good approach because it prevents a few form-related events—namely the *QueryUnload* and the *Unload* events—from firing. In some cases, these event procedures contain the so-called *clean-up code,* for example, statements that close a database or delete a temporary file. If you abruptly stop the execution of a program, you're actually preventing the execution of this code. As a general rule, use the End command only if strictly necessary. This program is so simple that you hardly need to test and debug it. Of course, this wouldn't be true for any real-world application. Virtually all programs need to betested and debugged, which is probably the most delicate (and often tedious) part of a programmer's job. Visual Basic can't save you from this nuisance, but at least it offers so many tools that you can often complete it very quickly. To see some Visual Basic debugging tools in action, place a breakpoint on the first line of the *Click* event procedure while the program is in design mode. You can set a breakpoint by moving the text cursor to the appropriate line and then invoking the

Toggle Breakpoint command from the Debug menu or pressing the F9 shortcut key. You can also set and delete breakpoints by left-clicking on the gray vertical strip that runs near the left border of the code editor window. In all cases, the line on which the breakpoint is set will be highlighted in red.

After setting the breakpoint at the beginning of the *Click* event procedure, press F5 to run the program once again, enter some values in the Width and Height fields, and then click on the Evaluate button. You'll see the Visual Basic environment enter break mode, and you are free to perform several actions that let you better understand what's actually going on:

- Press F8 to execute the program one statement at a time. The Visual Basic instruction that's going to be executed next—that is, the current statement — is highlighted in yellow.

13

- Show the value of an expression by highlighting it in the code window and then pressing F9 (or selecting the Quick Watch command from the Debug menu). You can also add the selected expression to the list of values displayed in the Watch window, as you can see in Figure 1-10.

- An alternative way to show the value of a variable or a property is to move the mouse cursor over it in the code window; after a couple of seconds, a yellow *data tip* containing the corresponding value appears.

- Evaluate any expression by clicking on the Immediate window and typing *?* or *Print* followed by the expression. This is necessary when you need to evaluate the value of an expression that doesn't appear in the code window.

- You can view the values of all the local variables (but not expressions) by selecting the Locals command from the View menu. This command is particularly useful when you need to monitor the value of many local variables and you don't want to set up a watching expression for each one.

- You can affect the execution flow by placing the text cursor on the statement that you want to execute next and then selecting the Set Next Statement command from the Debug menu. Or you can press the Ctrl+F9 key combination. You need this technique to skip over a piece of code that you don't want to execute or to reexecute a given block of lines without restarting the program.
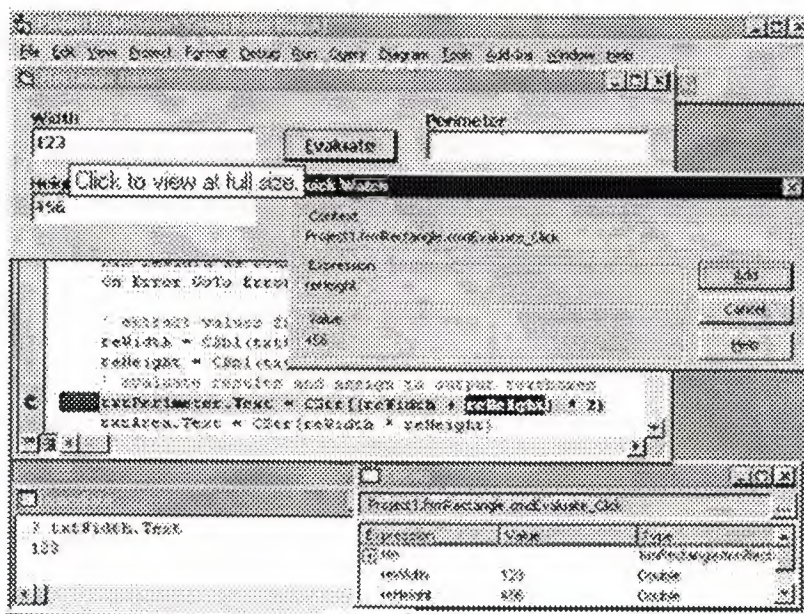
Figure 1-10. The Rectangle Demo program in break mode, with several debug tools activated.

1.9. Refining the Sample Program

Our first Visual Basic project, Rectangle.vbp, is just a sample program, but this is no excuse not to refine it and turn it into a complete and robust, albeit trivial, application.

The first type of refinement is very simple. Because the txtPerimeter and txtArea controls are used to show the results of the computation, it doesn't make sense to make their contents editable by the user. You can make them read-only fields by setting their *Locked* property to True. (A suggestion: select the two controls, press F4, and modify the property just once.) Some programmers prefer to use Label controls to display result values on a form, but using read-only TextBox controls has an advantage: The end user can copy their contents to the clipboard and paste those contents into another application.

A second refinement is geared toward increasing the application's consistency and usability. Let's suppose that your user uses the Rectangle program to determine the perimeter and area of a rectangle, takes note of the results, and then enters a new width or a new height (or both). Unfortunately, an instant before your user clicks on the Evaluate button the phone rings, engaging the user in a long conversation. When he or she hangs up, the form shows a plausible, though incorrect, result. How can you be sure that those values won't be mistaken for good ones? The solution is simple, indeed: as soon as the user modifies either the txtWidth or the txtHeight TextBox controls, the result fields must be cleared. In Visual Basic, you can accomplish this task by trapping each source control's *Change* event and writing a couple of statements in the corresponding event procedure. Since *Change* is the default event for TextBox controls—just as the *Click* event is for CommandButtons controls—you only have to double-click the txtWidth and txtHeight controls on the form designer to

have Visual Basic create the template for the corresponding event procedures. This is the code that you have to add to the procedures:

```
Private Sub txtWidth_Change()
    txtPerimeter.Text = ""
    txtArea.Text = ""
End Sub
Private Sub txtHeight_Change()
    txtPerimeter.Text = ""
    txtArea.Text = ""
End Sub
```

Note that you don't have to retype the statements in the *txtHeight*'s *Change* event procedure: just double-click the control to create the *Sub ... End Sub* template, and then copy and paste the code from the *txtWidth_Click* procedure. When you're finished, press F5 to run the program to check that it now behaves as expected.

The purpose of the next refinement that I am proposing is to increase the program's robustness. To see what I mean, run the Rectangle project and press the Evaluate button without entering width or height values: the program raises a Type Mismatch error when trying to extract a numeric value from the txtWidth control. If this were a real-world, compiled application, such an *untrapped* error would cause the application to end abruptly, which is, of course, unacceptable. All errors should be trapped and dealt with in a convenient way. For example, you should show the user where the problem is and how to fix it. The easiest way to achieve this is by setting up an error handler in the *cmdEvaluate_Click* procedure, as follows. (The lines you would add are in boldface.)

```
Private Sub cmdEvaluate_Click()
    ' Declare two floating point variables.
    Dim reWidth As Double, reHeight As Double
    On Error GoTo WrongValues
    ' Extract values from input textbox controls.
    reWidth = CDbl(txtWidth.Text)
    reHeight = CDbl(txtHeight.Text)
    Ensure that they are positive values.
    If reWidth <= 0 Or reHeight <= 0 Then GoTo WrongValues
    ' Evaluate results and assign to output text boxes.
    txtPerimeter.Text = CStr((reWidth + reHeight) * 2)
    txtArea.Text = CStr(reWidth * reHeight)
```

```
    Exit Sub
WrongValues:
    MsgBox "Please enter valid Width and Height values",
vbExclamation
End Sub
```

Note that we have to add an *Exit Sub* statement to prevent the *MsgBox* statement from being erroneously executed during the normal execution flow. To see how the *On Error* statement works, set a breakpoint on the first line of this procedure, run the application, and press the F8 key to see what happens when either of the TextBox controls contains an empty or invalid string.

1.10.Ready, Compile, Run!

Visual Basic is a very productive programming language because it allows you to build and test your applications in a controlled environment, without first producing a compiled executable program. This is possible because Visual Basic converts your source code into *p-code* and then interprets it. P-code is a sort of intermediate language, which, because it's not executed directly by the CPU, is slower than real natively compiled code. On the other hand, the conversion from source code to p-code takes only a fraction of the time needed to deliver a compiled application. This is a great productivity bonus unknown to many other languages. Another benefit of p-code is that you can execute it step-by-step while the program is running in the environment, investigate the values of the variables, and—to some extent—even modify the code itself. This is a capability that many other languages don't have or have acquired only recently; for example, the latest version of Microsoft Visual C++ has it. By comparison, Visual Basic has always offered this feature, which undoubtedly contributed to making it a successful language. At some time during the program development, you might want to create an executable (EXE) program. There are several reasons to do this: compiled programs are often (much) faster than interpreted ones, users don't need to install Visual Basic to run your application, and you usually don't want to let other people peek at your source code. Visual Basic makes the compilation process a breeze: when you're sure that your application is completed, you just have to run the Make *projectname* command from the File menu.

It takes a few seconds to create the Rectangle.exe file. This executable file is independent of the Visual Basic environment and can be executed in the same way as any other Windows application—for example, from the Run command of the Start menu. But this doesn't mean that you can pass this EXE file to another user and expect that it works. All Visual Basic programs, in fact, depend on a number of ancillary files—most notably the MSVBVM60.DLL file, a part of the Visual Basic runtime—and won't execute accurately unless all such files are correctly installed on the target system.,

For this reason, you should never assume that a Visual Basic program will execute on every Windows system because it's working on your computer or on other computers in your office. (If your business is software development, it's highly probable that the Visual Basic environment is installed on all the computers around you.) Instead, prepare a standard installation using the Package and Deployment Wizard, and try running your application on a clean system. If you develop software professionally, you should always have such a clean system at hand, if possible with just the operating system installed. If you're an independent developer, you probably won't be inclined to buy a complete system just to test your software. I found a very simple and relatively inexpensive solution to this dilemma: I use one computer with removable hard disks, so I can easily test my applications under different system configurations. And since a clean system requires only hundreds of megabytes of disk space, I can recycle all of my old hard disks that aren't large enough for any other use.

Before I conclude this chapter, you should be aware of one more detail. The compilation process doesn't necessarily mean that you aren't using p-code. In the Visual Basic jargon, *compiling* merely means *creating an executable file*. In fact, you can compile to p-code, even if this sounds like an oxymoron to a developer coming from another language. (See Figure 1-11.) In this case, Visual Basic creates an EXE

file that embeds the same p-code that was used inside the development environment. That's why you can often hear Visual Basic developers talking about *p-code* and *native-code* compilations to better specify which type of compilation they're referring to.
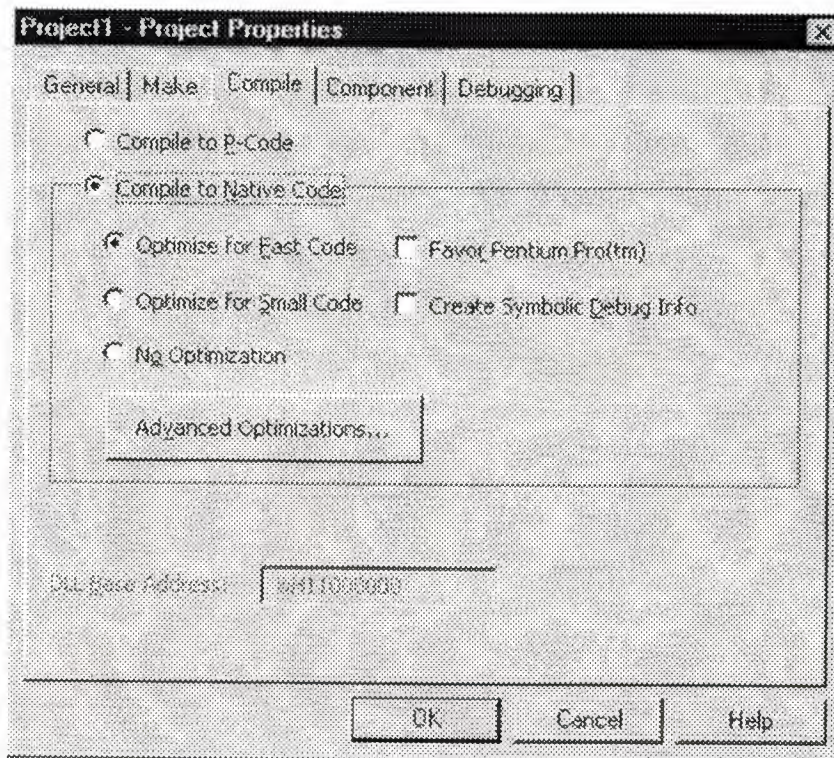
Figure 1-11. You can opt to compile to p-code or native code in the Compile tab of the Project Properties dialog.

In general, such p-code-compiled programs run at the same speed as interpreted programs within the IDE, so you're missing one of the biggest benefits of the compilation process. But here are a few reasons why you might decide to create a p-code executable:

- P-code-compiled executables are often smaller than programs compiled to native code. This point can be important if you're going to distribute your application over the Internet or when you're creating ActiveX controls that are embedded in an HTML page.

- P-code compilation is often faster than native code compilation, so you might prefer to stick to p-code when you compile the program in the test phase. (A few types of applications can't be tested within the IDE, most notably multithreaded components.)

- If your application spends most of its time accessing databases or redrawing windows, compilation to native code doesn't significantly improve its performance because the time spent executing Visual Basic code is only a fraction of the total execution time.

We've come to the end of this *tour de force* in the Visual Basic IDE. In this chapter, I've illustrated the basics of Visual Basic development, and I hope I've given you a taste of how productive this language can be. Now you're ready to move to the next chapters, where you can learn more about forms and controls and about how to make the best of their properties, methods, and events.

CHAPTER 2

DATABASE AND ACCESS

2.1.Why is the computer necessary in our life

Computer software has become a driving force; it is a powerful force that set Decision-making and serves as a basis for modern investigation and problem solving.Computers have become a key factor that gives products and services that modern   look   ,its   embedded   in   systems   of   all kinds;medical,industrial,military,entertainment,even office-based products.

A Computer system in a service management record can promise better speed and efficiency with almost no change of effors.

2.2.How to develop a database application

The steps involved in database application development any relational data base application there are always the same basic steps to follow.Microsoft Access is a relational data base management system because all data is stored in an Access data base in the form of simple tables.Another name for a table is relation.

The steps of Access database design like this

- Database design
- Tables design
- Forms design
- Query design
- Report design
- Macro design
- Modüles design

2.3.Relational database

DBMS has established themselves as one of the primary means for data storage for information based systems ranging from large business applications to simple pc based programs.However a relational database management system (RDBMS) is the system used to work with data management operations more than 15 years,and still improving,providing more sophisticated storage,retriaval systems.Relational database

management systems provides organisations with ability to handle huge ammount of data and changing it into meaningful information.

2.4. The facilities of access

Microsoft Access is relational DBMS(Database Management System) with all the features necessary to develop and use a data base application. The facilities it offers can be found on most modern relational DBMSs and all versions of Access.

- Tables are where all the data is stored. They are usually linked by relationships.
- Queries are the way you extract data from the database
- Forms are the method used for input and display of database data.
- Reports are used to display nicely formatted data on paper.
- Macros are sets of simple commands that execute sequences of database operations.
- Modules are used to stare general-purpose VB database program code.

2.5. Visual basic and Access

Microsoft Acess is the DBMS(Database Management System) VB and Access in developing data base applications is that for non-trivial database applications, VB offers more flexibility to the developper then the VB comes with Access. Access database using VB program code and setting properties.

First method of linking VB forms to Access databases called the data control. The data control is a simple VB control that you drag on to a VB form to link it to your choosen database. The data can be displayed and updated using tiedtext boxes, list boxes, combo boxes, and grids.

2.5.1.DAO(Data Access objects)

The DAO approach to database programming often requires more code ,but like SQL compared to the Qury Design View,offers greater control to the database programmer over what's going on his/her application.

Data Access Objects are things like databases,recordsets,table and query definitions, and fields,Rather than tying a recor set to a data control when we use DAO we shall allow our programs to create and manipulate recordsets.

2.5.2.ADO(Active X Data Objects)

The ADO programming is in principle very similar to DAO programming but cointains some new commands.ADO is Microsoft's new approach to database programming which aims to give the programmer a more consistent way of connecting to a broad range of different types of data source.

2.6.The application of Access

MS Access is begin used as the development tool,and the application is going to be a single user application,which means its going to be instolled on one machine,this application however may be used by more than one user on many computers sharing the same tables by using simple advancements.

For a new database,after having specified the database name and path as above,you will be confronted with the following window.
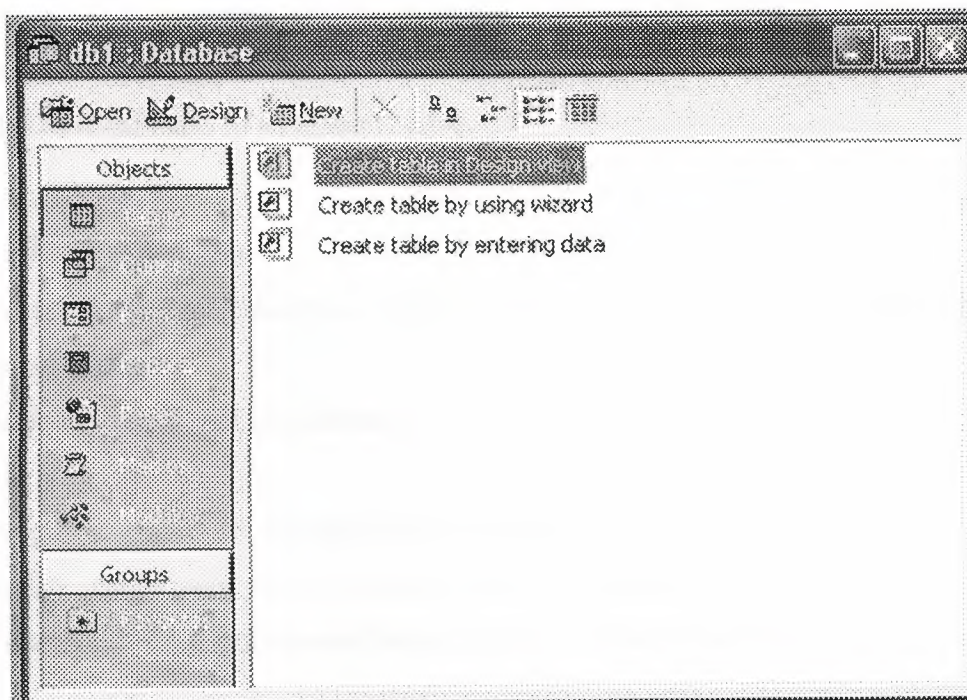
Figure1.1. The window of database

This window shows that there are notables in database yet.Click new button.

24
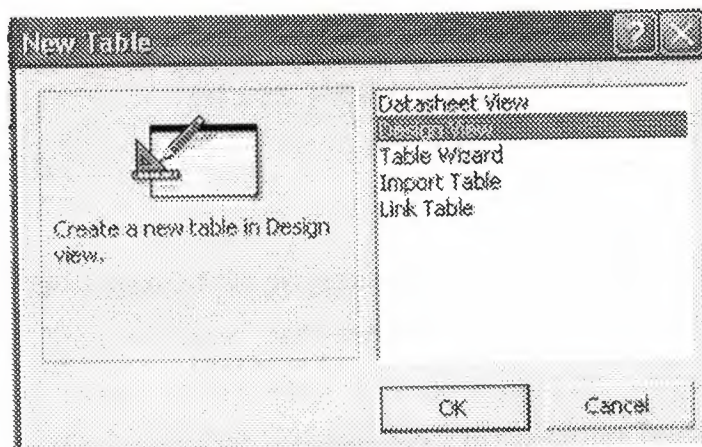
Figure 1.2. The window is type of table design

sellect the Design View by clicking on the listbox and then the OK button.Design View gives us more control over the design of our database than either the Table Wizard or the Datasheet view.Import Table is used to bring in data from an existing database and Link Table is database to an external table.

2.6.1. Tables Design

In my project's table designing with primary key. Guide Lines for making a database project.

The database consists of one tables;

Table1

Please pay attention on the naming conventions of objects, you are required to use appropriate names using these conventions for your objects.

TheTable one have got eleven fields one is its unique name

Figure 1.3 The Table1

MAİN PROGRAM

3.1. MAIN MENU

This is the main menu of the program. There is also some sub menus on the top of the main menu. From the main menu we can go sub programs by using this sub menus. There are also some buttons. They are used to go to the sub programs. They are providing facilities for users of the program. We can see all sub programs on the main menu.

Record button is used to go record part of the program. In the part we enter madicine rrecord information.

Information button is used to go to Iformation part. Here we make rapor of stock and son in the stock .

Seel button is used to show information such as number of madicine and piece of madicine and code of madicine

Esdeger button is used to keep information about the madicine and their accompaniment .

The form and codes of the main menu is following down.
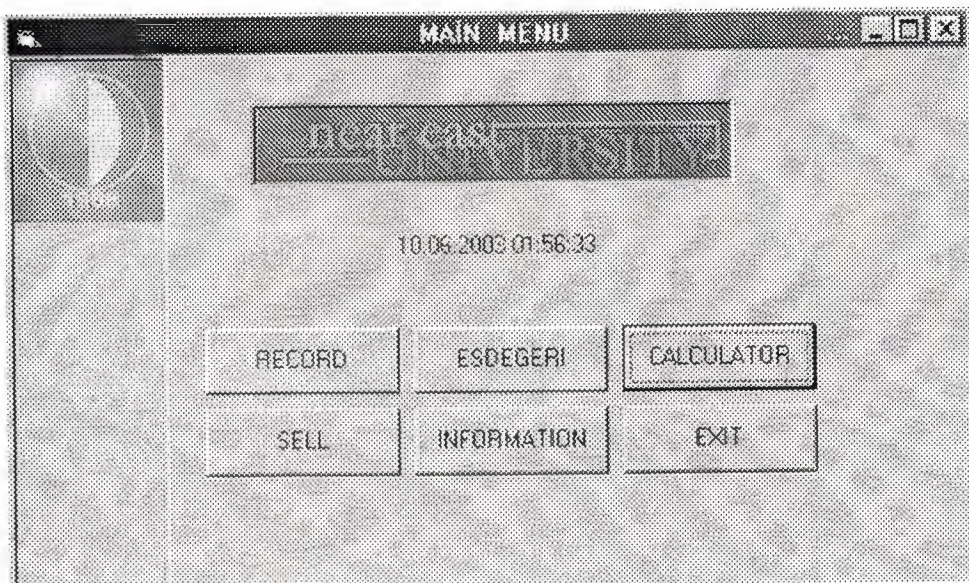
Figure 3.1. Main Menu

Private Sub Command1_Click()

```vb
Load Form2
Form2.Show
End Sub
Private Sub Command2_Click()
Load Form4
Form4.Show
Form1.Visible = False
End Sub
Private Sub Command3_Click()
Load Form3
Form3.Show
End Sub
Private Sub Command4_Click()
DataReport1.Show
Form1.Visible = True
End Sub
Private Sub Command5_Click(Index As Integer)
End
End Sub
Private Sub Command6_Click()
Load Form5
Form5.Show
Form1.Visible = False
End Sub
Private Sub Form_Initialize()
Form8.Show
Form9.Visible = True
Form1.Visible = False
Load Form8
Form8.Show
End Sub
Private Sub Form_Load()
Picture1.Align = center
End Sub
```

```
Private Sub Label1_Click()
Label1.Caption = Now()
End Sub
Private Sub Timer1_Timer()
Form1.Caption   =   Right(Form1.Caption,   (Len(Form1.Caption)   -   1))   +
Left(Form1.Caption, 1)
Label1.Caption = Now()
End Sub
```

3.2. THE PASSWORD SECREEN

I built In this secreen Who can enter the program. The progaram user have a psword and user

name. With pasword and user name the user can use the program. The pasword secreen is on
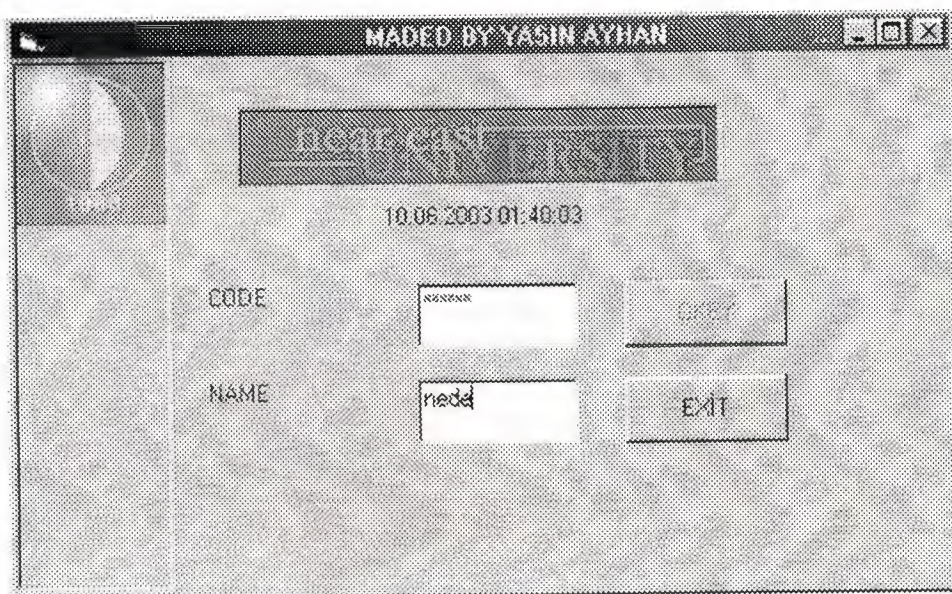
active when the program start to run.

Figure 3.2 Pasword Writing

Private Sub Command1_Click()

Form9.Visible = False

Form1.Visible = True

End Sub

Private Sub Command2_Click()

End

End Sub

Private Sub Text1_Change()

If Text1.Text <> "" And Text1.Text = "yasin" And Text2.Text <> "" Then

Command1.Enabled = True

28
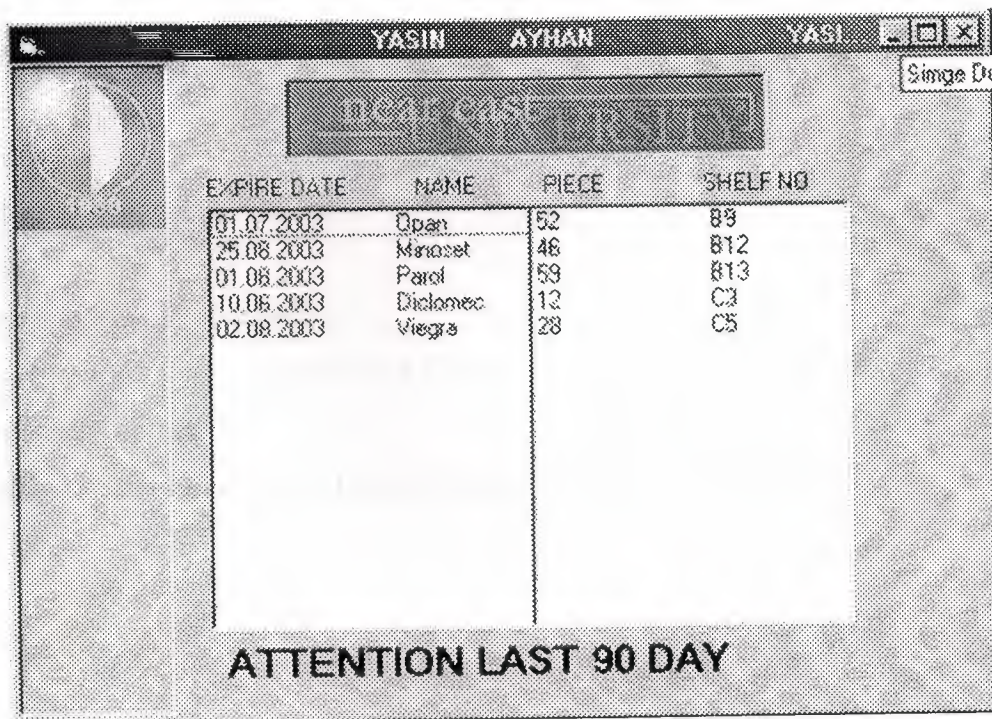
```
Command1.SetFocus

End If

End Sub

Private Sub Text2_Change()

If Text1.Text <> "" And Text2.Text <> "" And Text1.Text = "991267" Then

Command1.Enabled = True

Command1.SetFocus

End If

End Sub

Private Sub Timer1_Timer()

Label3.Caption = Now()

Form9.Caption    =    Right(Form9.Caption,    (Len(Form9.Caption)    -    1))    +
Left(Form9.Caption, 1)

End Sub
```

3.3. UNIT EXPIRE DATE

The secreen About Product which is expire date. When main program run expire date secreen

will be on active

directly. Al the information about madicine are provided with this form and there are 90

day

for exipire date which are shown product.

Figure 3.2. Product Selection

Private Sub Form_Initialize()

Dim db As Database

Dim tb As Recordset

Set db = OpenDatabase("C:\WINDOWS\Desktop\vt1.mdb")

```
Set tb = db.OpenRecordset("tablo1")
While Not tb.EOF
x = tb.Fields("l_date")
m1 = Val(Month(x))
m2 = Val(Month(Date))
Y1 = Val(Year(x))
Y2 = Val(Year(Date))
If Y1 = Y2 And m1 >= m2 Then
m = (m1 - m2) * 31
If m < 90 Then
List1.AddItem Str(tb.Fields("l_date")) + "            " + tb.Fields("name")
List2.AddItem tb.Fields("piece") + "                " + tb.Fields("shelf_no")
End If
End If
tb.MoveNext
Wend
End Sub
Private Sub ta_Click()
End Sub
Private Sub Timer1_Timer()
Form8.Caption    =    Right(Form8.Caption,    (Len(Form8.Caption)    -    1))    +
Left(Form8.Caption, 1)
ta.Caption = Right(ta.Caption, (Len(ta.Caption) - 1)) + Left(ta.Caption, 1)
End Sub
```

3.4. UNIT RECORD OF MADICINE

Section of showing the type of record. You can select to type of record with using record of

madicine secreen :the type of record are searching, deleting, adding, finding, editining and
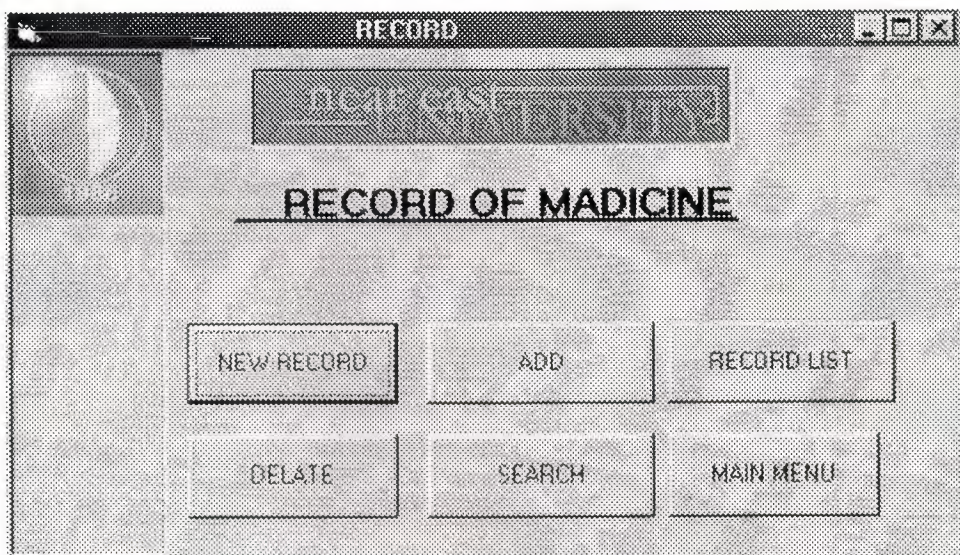
also you can see the report of stock with this screen.

Figure 3.4. Record Of Madicine

Private Sub Command1_Click(Index As Integer)

Load Form6

Form6.Show

Form2.Visible = False

End Sub

Private Sub Command2_Click()

```
Load Form6

Form6.Show

Form2.Visible = False

End Sub

Private Sub Command3_Click()

Load Form6

Form6.Show

Form2.Visible = False

End Sub

Private Sub Command4_Click()

Load Form6

Form6.Show

Form2.Visible = False

End Sub

Private Sub Command5_Click()

DataReport1.Show

End Sub

Private Sub Command6_Click()

Form2.Visible = False

Form1.Visible = True

Form1.Show

End Sub

Private Sub Form_Load()

Form1.Visible = False

End Sub

Private Sub RE_Click()

End Sub

Private Sub Timer1_Timer()

Form2.Caption    =    Right(Form2.Caption,    (Len(Form2.Caption)    -    1))    +
Left(Form2.Caption, 1)

End Sub
```

3.5. UNITE RECORDE

The recording will accour with code. If oyu want to write data to database You must enter the

code of product. May be you forget to enter product code the software will situmulate to user

with message box. Also when you record some data to database some of the recording name

is vary important. Forexample code and expire date this are vary important object of the program. Because of searching accoure with code and expire date. Also the name of accompaniment is vary important because of if the product there is not on the stock the accompaniment will came to screen. That is the reson you must enter the accompaniment of

product.

Figure 3.5. Record

```
Private Sub Command1_Click()

Dim db As Database

Dim tb As Recordset

Set db = OpenDatabase("C:\WINDOWS\Desktop\vt1.mdb")

Set tb = db.OpenRecordset("tablo1")

yasin:

MsgBox "YOU MUST ENTER ALL COMPONENT"

a = MsgBox("DO YOU WANT TO RECORD", vbYesNoCancel, "READ
CREFULLY")

If a = 6 Then GoTo kayit

If a = 7 Then GoTo fin

If a = 2 Then GoTo fin2

kayit:

tb.AddNew

If Text5.Text = "" Then GoTo yasin
```

```
tb.Fields("no") = Text1.Text
tb.Fields("explanation") = Text2.Text
tb.Fields("piece") = Text3.Text
tb.Fields("a_item") = Text4.Text
tb.Fields("l_date") = Text5.Text
tb.Fields("shelf_no") = Text6.Text
tb.Fields("p_explenation") = Text11.Text
tb.Fields("name") = Text12.Text
tb.Fields("f_tel") = Text8.Text
tb.Fields("f_name") = Text9.Text
tb.Fields("b_date") = Text10.Text
tb.Fields("price") = Text7.Text
Move Last
If Text1.Text = "" Then GoTo yasin
tb.Update
tb.Close
db.Close
fin:
GoTo son
fin2:
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
Text6.Text = ""
Text7.Text = ""
Text8.Text = ""
Text9.Text = ""
Text10.Text = ""
Text11.Text = ""
Text12.Text = ""
son:
End Sub
```

```vb
Private Sub Command2_Click()

Dim db As Database

Dim tb As Recordset

Dim s  As String

Dim c As Integer

Set db = OpenDatabase("C:\WINDOWS\Desktop\vt1.mdb")

Set tb = db.OpenRecordset("tablo1")

s = Text1.Text

tb.Index = "primarykey"

tb.Seek "=", s

While Not tb.EOF

If s = tb.Fields("no") Then

a = MsgBox("do you want to delate", vbYesNo, "delate screen")

If a = vbYes Then GoTo sil

If a = vbNo Then GoTo atla

sil:

tb.Delete

Text1.Text = ""

Text2.Text = ""

Text3.Text = ""

Text4.Text = ""

Text5.Text = ""

Text6.Text = ""

Text7.Text = ""

Text8.Text = ""

Text9.Text = ""

Text10.Text = ""

Text11.Text = ""

Text12.Text = ""

c = c + 1

atla:

End If

tb.MoveNext

Wend
```

```
If c <> 0 Then
MsgBox "is deleted by you"
End If
End Sub
Private Sub Command3_Click()
Form6.Visible = False
Form2.Visible = True
Form2.Show
End Sub
Private Sub Command4_Click()
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Text5.Text = ""
Text6.Text = ""
Text7.Text = ""
Text8.Text = ""
Text9.Text = ""
Text10.Text = ""
Text11.Text = ""
Text12.Text = ""
End Sub
Private Sub Command5_Click()
Dim db As Database
Dim tb As Recordset
Dim s  As String
Dim c As Integer
Set db = OpenDatabase("C:\WINDOWS\Desktop\vt1.mdb")
Set tb = db.OpenRecordset("tablo1")
s = Text1.Text
tb.Index = "primarykey"
tb.Seek "=", s
While Not tb.EOF
```

```
If s = tb.Fields("no") Then
a = MsgBox("is it this record", vbYesNo, "search screen")
If a = vbYes Then GoTo sil
If a = vbNo Then GoTo atla
sil:
Text1.Text = tb.Fields("no")
 Text2.Text = tb.Fields("explanation")
 Text3.Text = tb.Fields("piece")
 Text4.Text = tb.Fields("a_item")
 Text5.Text = tb.Fields("l_date")
 Text6.Text = tb.Fields("shelf_no")
 Text11.Text = tb.Fields("p_explenation")
 Text12.Text = tb.Fields("name")
 Text8.Text = tb.Fields("f_tel")
 Text9.Text = tb.Fields("f_name")
 Text10.Text = tb.Fields("b_date")
 Text7.Text = tb.Fields("price")
c = c + 1
atla:
End If
tb.MoveNext
 Wend
 If c <> 0 Then
 MsgBox "is found by searcher"
 End If
 End Sub
 Private Sub Command6_Click()
 Dim db As Database
 Dim tb As Recordset
 Dim s  As String
 Dim c As Integer
 Set db = OpenDatabase("C:\WINDOWS\Desktop\vt1.mdb")
 Set tb = db.OpenRecordset("tablo1")
 s = Text1.Text
```

```
tb.Index = "primarykey"

tb.Seek "=", s

While Not tb.EOF

If s = tb.Fields("no") Then

a = MsgBox("do you want to change", vbYesNo, "changing screen")

If a = vbYes Then GoTo sil

If a = vbNo Then GoTo atla

sil:

tb.Edit

tb.Fields("no") = Text1.Text

tb.Fields("explanation") = Text2.Text

tb.Fields("piece") = Text3.Text

tb.Fields("a_item") = Text4.Text

tb.Fields("l_date") = Text5.Text

tb.Fields("shelf_no") = Text6.Text

tb.Fields("p_explenation") = Text11.Text

tb.Fields("name") = Text12.Text

tb.Fields("f_tel") = Text8.Text

tb.Fields("f_name") = Text9.Text

tb.Fields("b_date") = Text10.Text

tb.Fields("price") = Text7.Text

tb.Update

c = c + 1

atla:

End If

tb.MoveNext

Wend

If c <> 0 Then

MsgBox "is changed by you"

End If

End Sub

Private Sub Form_Load()

End Sub
```

3.6. UNITE ACCOMPANIMENT

In this screen you can learn any madicine that you want thay are name, shelf no, accompaniment, how many piece so on . If your madicine is there the software will stimulateto you like there is your searching madicine in the database with messagebox. İf there is notany pruduct that you want also the soft ware will stimulate to you like, there is not any item

like aspirin please buy on the list. Also the madicine on the list are the same madicine that you

want but just the company name is different. İf you interest the chamical line the pruduct are the same product on list and you want.
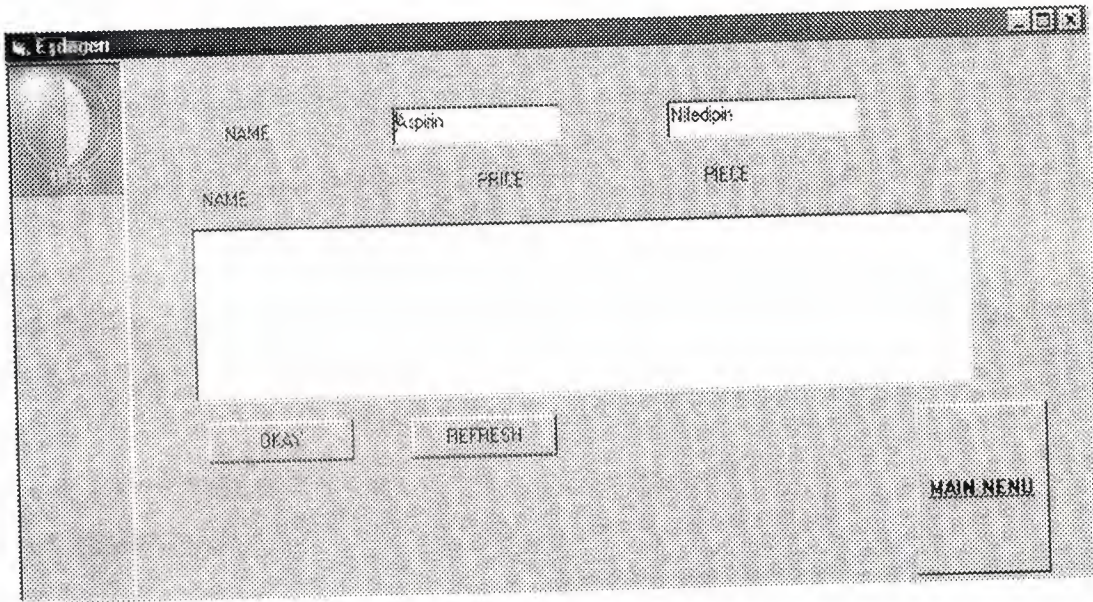
Figure 3.6. Eşdegeri

Private Sub Command1_Click()

Form3.Visible = False

Form1.Visible = True

Form1.Show

End Sub

Private Sub Command2_Click()

Dim db As Database

Dim tb As Recordset

Dim s, sd, a1, a2, a3, a4, a5 As String

Dim a11, a21, a31, a41, a51 As String

Dim c As Integer

Set db = OpenDatabase("C:\WINDOWS\Desktop\vt1.mdb")

Set tb = db.OpenRecordset("Tablo1")

s = Text1.Text

While Not tb.EOF

If s = tb.Fields("name") Then

a5 = tb.Fields("no")

a3 = tb.Fields("piece")

```vb
a4 = tb.Fields("a_item")
a1 = tb.Fields("name")
a2 = tb.Fields("price")
take = a4
List1.AddItem tb.Fields("name") + "                         " + "                         " + a2 +
"
" + a3 + "                         " + a4 + "            " + a5
c = c + 1
End If
tb.MoveNext
Wend
If Val(a3) > 0 Then GoTo atla
List1.Clear

MsgBox "THERE IS NOT " + s
tb.MoveFirst
While Not tb.EOF
If take = tb.Fields("a_item") Then
 If s = tb.Fields("name") Then GoTo unwrite
 a51 = tb.Fields("no")
 a31 = tb.Fields("piece")
 a41 = tb.Fields("a_item")
 a11 = tb.Fields("name")
 a21 = tb.Fields("price")
List1.AddItem tb.Fields("name") + "   " + a21 + "   " + a31 + "    " + a41 + "     " + a51 +
unwrite:
End If
tb.MoveNext
Wend
atla:
If c = 0 Then MsgBox "aaaaaaa"
End Sub
Private Sub Command3_Click()
List1.Clear
```

```vb
Text1.SetFocus
Text1.Text = ""
Text2.Text = ""
End Sub
Private Sub Form_Load()
Form1.Visible = False
End Sub
Private Sub Command1_Click()
Form3.Visible = False
Form1.Visible = True
Form1.Show
End Sub
Private Sub Command2_Click()
Dim db As Database
Dim tb As Recordset
Dim s, sd, a1, a2, a3, a4, a5 As String
Dim a11, a21, a31, a41, a51 As String
Dim c As Integer
Set db = OpenDatabase("C:\WINDOWS\Desktop\vt1.mdb")
Set tb = db.OpenRecordset("Tablo1")
s = Text1.Text
While Not tb.EOF
If s = tb.Fields("name") Then
a5 = tb.Fields("no")
a3 = tb.Fields("piece")
a4 = tb.Fields("a_item")
a1 = tb.Fields("name")
a2 = tb.Fields("price")
take = a4
List1.AddItem tb.Fields("name") + "                    " + "                    " + a2 +
"
" + a3 + "                    " + a4 + "            " + a5
c = c + 1
End If
```

```vb
tb.MoveNext
Wend
If Val(a3) > 0 Then GoTo atla
List1.Clear

MsgBox "THERE IS NOT " + s
tb.MoveFirst
While Not tb.EOF
If take = tb.Fields("a_item") Then
 If s = tb.Fields("name") Then GoTo unwrite
  a51 = tb.Fields("no")
 a31 = tb.Fields("piece")
 a41 = tb.Fields("a_item")
 a11 = tb.Fields("name")
 a21 = tb.Fields("price")
List1.AddItem tb.Fields("name") + "   " + a21 + "   " + a31 + "    " + a41 + "     " + a51
unwrite:
End If
tb.MoveNext
Wend
atla:
If c = 0 Then MsgBox "aaaaaaa"
End Sub
Private Sub Command3_Click()
List1.Clear
Text1.SetFocus
Text1.Text = ""
Text2.Text = ""
End Sub
Private Sub Form_Load()
Form1.Visible = False
End Sub
```

3.7. UNIT CALCULATOR

In this screen there is clasic calculator and sciectific calculator. I made a map for scientific

calculator to windows.
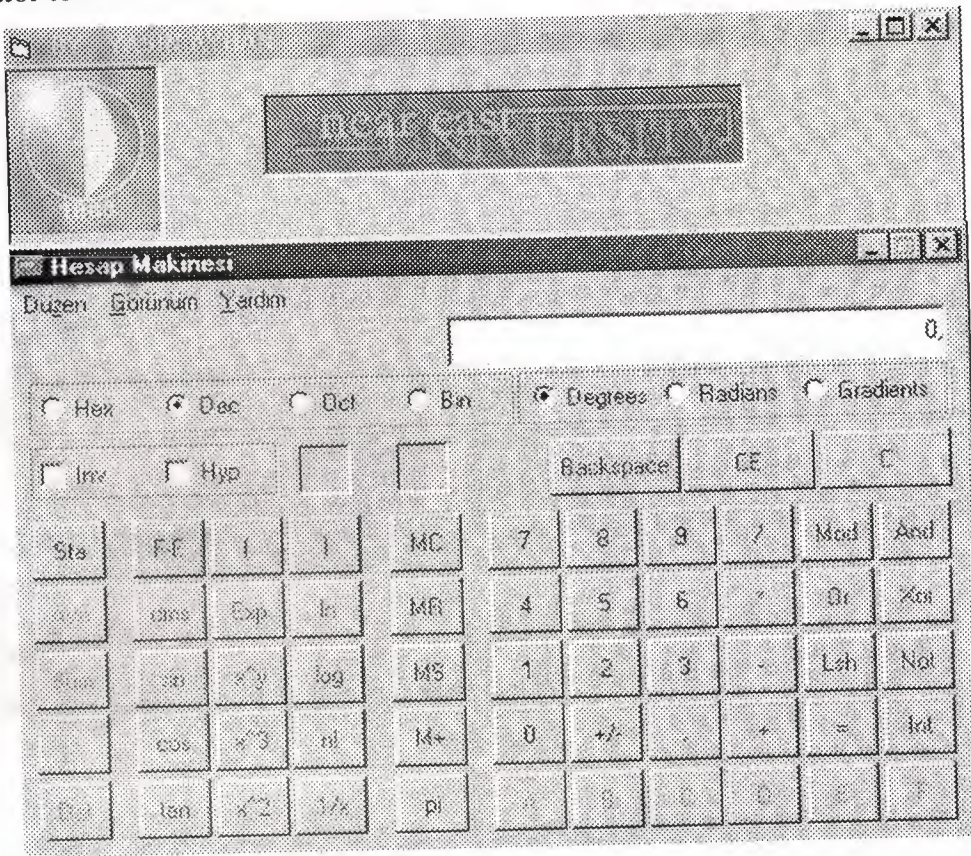
Figure 3.7. Calculator

Dim a As Integer

Dim c As Integer

Private Sub Command1_Click()

c = 1

a = Val(Text1.Text)

Text1.Text = ""

Text1.SetFocus

End Sub

Private Sub Command2_Click()

c = 2

a = Val(Text1.Text)

Text1.Text = ""

Text1.SetFocus

End Sub

Private Sub Command3_Click()

b = Val(Text1.Text)

```
If c = 1 Then

Text1.Text = Str(a + b)

End If

If c = 2 Then

Text1.Text = Str(a - b)

End If

If c = 3 Then

Text1.Text = Str(a * b)

End If

If c = 4 Then

Text1.Text = Str(a / b)

End If

End Sub

Private Sub Command4_Click()

c = 4

a = Val(Text1.Text)

Text1.Text = ""

Text1.SetFocus

End Sub

Private Sub Command5_Click()

c = 3

a = Val(Text1.Text)

Text1.Text = ""

Text1.SetFocus

End Sub

Private Sub Command7_Click()

Load Form1

Form1.Show

Form5.Visible = False

End Sub

Private Sub Form_Load()

Form1.Visible = False

Text1.Text = ""

End Sub
```

```
Private Sub Timer1_Timer()
Form5.Caption    =    Right(Form5.Caption,    (Len(Form5.Caption)    -    1))    +
Left(Form5.Caption, 1)
End Sub
```

3.8. UNIT SALLING

I think this screen is so important screen because of you sale your madicine from your stock.

Esy to use this scren. After salling the madicine the madicine will dicrease from the stock.

Figure 3.8. Salling
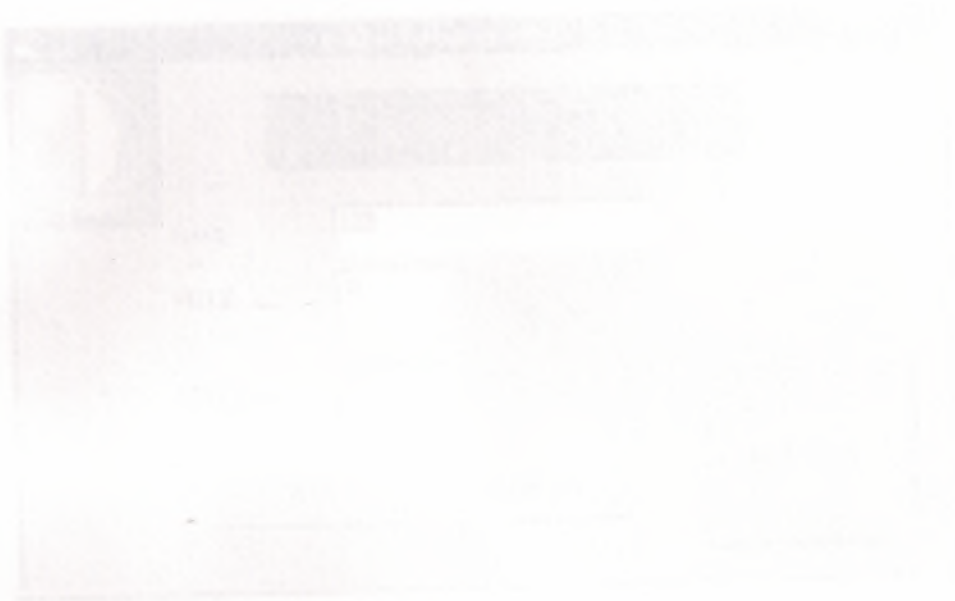
```
Private Sub Command1_Click()
Form4.Visible = False
Form1.Visible = True
Form1.Show
End Sub
Private Sub Command2_Click()
Dim db As Database
Dim tb As Recordset
Dim s  As String
Dim c As Integer
Set db = OpenDatabase("C:\WINDOWS\Desktop\vt1.mdb")
Set tb = db.OpenRecordset("tablo1")
s = Text1.Text
tb.Index = "primarykey"
```

50

```
tb.Seek "=", s

If Val(tb.Fields("piece")) = 0 Or Val(tb.Fields("piece")) < Val(Text2.Text) Then GoTo

atla

While Not tb.EOF

If s = tb.Fields("no") Then

k = Int(tb.Fields("piece")) - Int(Text2.Text)

a = MsgBox("did you sell", vbYesNo, "changing screen")

If a = vbYes Then GoTo sil

If a = vbNo Then GoTo atla

sil:

tb.Edit

tb.Fields("piece") = Str$(k)

tb.Update

c = c + 1

End If

tb.MoveNext

Wend

If c <> 0 Then

MsgBox "It is salled "

Else

atla:

MsgBox "DEMAND IS GRATER THEN STOCK "

MsgBox "THERE ARE" + "" + tb.Fields("piece")

Text3.Text = tb.Fields("shelf_no")

End If

End Sub

Private Sub Command3_Click()

Text1.Text = ""

Text2.Text = ""

Text3.Text = ""

End Sub

Private Sub Form_Load()

Form1.Visible = False

End Sub
```

```
Private Sub Timer1_Timer()
Form4.Caption    =    Right(Form4.Caption,    (Len(Form4.Caption)    -    1))    +
Left(Form4.Caption, 1)
End Sub
```

3.9. UNIT REPORT

The report of all stock in the firm. We can get any extra information obout madicine

Figure 3.8. Information of Stock

3.10. UNIT DATA ENVIRONMENT

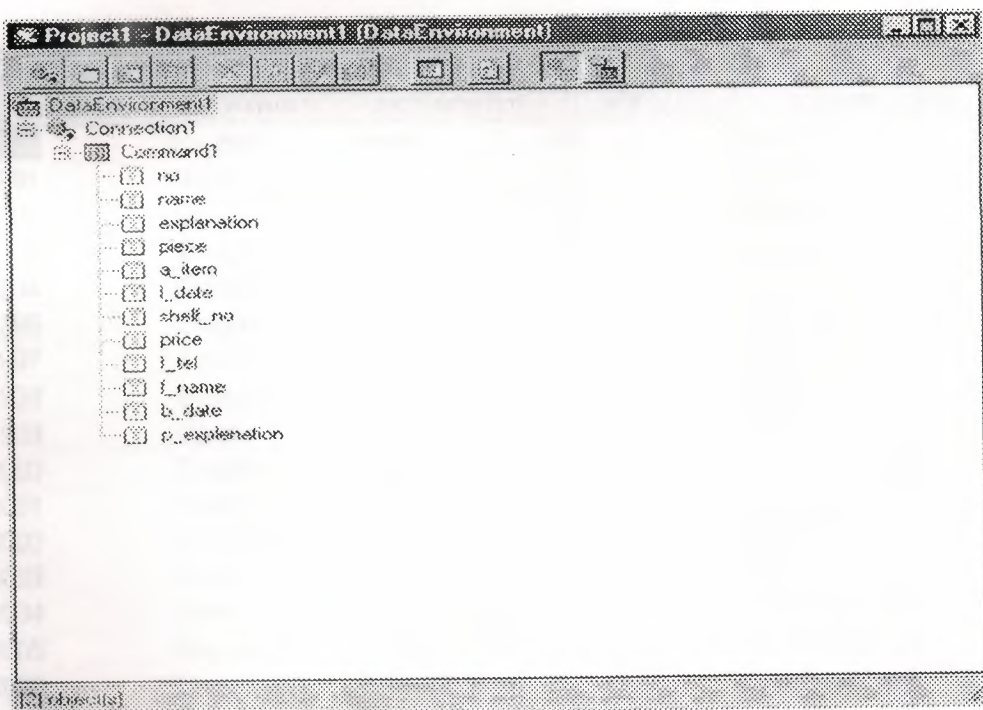Also we use data environment for creating the report

Figure 3.9. Data environment

3.11.Creating Tables

For a new database, after having specified the database name and path as above, you will be confronted with the following window.

3.11.1.Tables

The table of contents in our data base is on table if you look at the table you will see the primary key the searching will accrue with primary key. Also another fields of table is name of the contents.

no	name	explanation	piece	a item	l date
1990	Kardilol	heart	20	Nifedipin	26.05.2003
1991	Nidilat	heart	15	Nifedipin	26.09.2003
1992	Nidicard	heart	30	Nifedipin	21.09.2003
1993	Omepral	stomach	20	Omeprazol	02.02.2004
1994	Prosek	stomach	50	Omeprazol	02.02.2004
1995	Erbolin	stomach	25	Omeprazol	26.05.2003
1997	Asprin	Headeck	80	Acetylsalicit as	27.11.2003
1998	Baybprin	Headeck	40	Acetylsalicit as	02.01.2003
1999	Opan	Headeck	52	Acetylsalicit as	01.07.2003
2000	Ecoprin	Headeck	3	Acetylsalicit as	02.04.2003
2001	Panadol	Headeck	45	Paracetamol	25.05.2007
2002	Minoset	Headeck	46	Paracetamol	25.08.2003
2003	Parol	Headeck	59	Paracetamol	01.08.2003
2004	Volteran	Vheumatizm	82	Diclofenac sodi	25.09.2004
2005	Miyadren	Vheumatizm	88	Diclofenac sodi	25.05.2003
2006	Diclomec	Vheumatizm	12	Diclofenac sodi	10.06.2003
2007	Odonok	Vheumatizm	25	Diclofenac sodi	02.08.2004
2008	Viegra	Heart	28	Sildenofil	02.08.2003

Figure 3.10. Database

shelf_no	price	f_tel	f_name	b_date	p_explenation
B1	250000000	05428633125	Bayer	26.05.2001	n
B2	200000000	05428841545	Aysa	26.09.2001	n
B3	196000000	05426565586	Bayer	21.09.2001	n
B5	250000000	05423645254	Omepral	02.02.2002	y
B6	19000000	05422654528	Omepral	02.02.2004	n
B7	250000000	05422532524	Omepral	20.07.2001	n
B8	10000000	08542115021	Bayer	10.05.2002	y
B8	60000000	05425847892	Bayer	10.05.2001	y
B9	90000000	05422645621	Aysa	29.05.2002	y
B10	15000000	05425654222	Bayer	05.03.2001	y
B11	12000000	05332654256	Paracetamol	20.08.2001	y
B12	20000000	03687559549	Paracetamol	01.02.2001	y
B13	19000000	03887214516	Aysa	01.10.2002	y
C1	60000000	08624520145	Bayer	01.02.2002	n
C2	96100000	08624512145	Diclofen	01.05.2002	n
C3	60000000	05451214552	Aysa	01.10.2002	n
C4	92000000	05425863145	Diclofen	01.12.2002	n
C5	250000000	03256645211	Aysa	01.10.2002	n

Figure 3.10.1 Database

55

BALANCES

In this part I prepared code balances accompaniment balances and expire date balance. In the balances there are three section.

First section is search criteria. Here we can search code and madicine that we want to find it. We search madicine by use the mad_ no ,m_name , balance, and balance type. We search madicine by use the code, accompaniment, and expire date

Second section is sort criteria section .we sort madicine by code, accompaniment, and balance.

In the final section there is a table which we can see all balance information about the madicine.

Figure 3.11. Data Balance

CONCLUSSION

Visual Basic is an easy program to grasp. This cause is why we have had decided to use this program.

Visual Basic is a Microsoft Windows programming Language. Visual Basic is a distinctly different language providing powerfull features such as graphical user interfaces, even handling, access to the Win32 API, object-oriented features, error handling, structured programming, and much more.

In this project we have built madicine database program. It is easy to use and It can be use most kind of drugstore. We have used Visual Basic for write this program and I used Microsoft Access database for keep all my databases.

In this study our main aim to put accross is that this program can be operated by some one who has never used it before.

In this program there is also menus to make your writting much simpler, It containing windows menus and also afacility to prepare reports.

REFERANCES

1-) Ihsan Karagülle ; Zeydin Pala(1999). Microsoft Visual Basic 6.0. Istanbul.
Türkmen press.

2-) Prof. Dr. Mithat Uysal (1999). Development Of The Software with Visual
Basic 6.0. Istanbul. Beta Press.

3-) Ihsan Karagülle ; Zeydin Pala (1999). Microsoft Visual Basic
6.0 Pro. Istanbul. Türkmen Press.

4-) Hilal Drugstore, Girne

5-) Macit Pharmacy, Lefkosa