PROGRAMMING WITH VB : PROCEDURES

PRESENTATION OVERVIEW

- Introduction
- Procedure and function
- Menu designing in VB
- Data programming
- Simple activex controls

PROCEDURE AND FUNCTIONS

We use procedures and functions to create modular programs. Visual basic statements are grouped in a block enclosed by sub, function and matching end statements. The difference between the two is that functions return values, procedures do not.
A procedure and function is a piece of code in a larger program. They perform a specific task.

The advantages of using procedures and functions are:

- Reducing duplication of code
- Decomposing complex problems into simpler pieces
- Improving clarity of the code
- Reuse of code
- Information hiding

PROCEDURE

A procedure is a block of Visual Basic statements inside Sub, End Sub statements. Procedures do not return values.

```
Option Strict On

Module Example
        Sub Main()
         SimpleProcedure()

  End Sub
        Sub SimpleProcedure()
          Console.WriteLine("Simple procedure")
  End Sub

End Module
```

This example shows basic usage of procedures. In our program, we have two procedures. The Main() procedure and the user defined SimpleProcedure(). As we already know, the Main() procedure is the entry point of a Visual Basic program

```
SimpleProcedure()
```

Each procedure has a name. Inside the `Main()` procedure, we *call* our user defined `SimpleProcedure()` procedure.

```
Sub SimpleProcedure()
    Console.WriteLine("Simple Procedure")
End Sub
```

Procedures are defined outside the main() procedure. Procedure name follows the sub statement. When we call a procedure inside the visual basic program, the control is given to that procedure. Statements inside the block of the procedure are executed.

Procedures can take optional parameters.

FUNCTIONS

A function is a block of Visual Basic statements inside Function, End Function statements. Functions return values.

There are two basic types of functions. Built-in functions and user defined ones. The built-in functions are part of the Visual Basic language. There are various mathematical, string or conversion functions.

```
Option Strict On
 Module Example
 Sub Main()
   Console.WriteLine(Math.Abs(-23))
   Console.WriteLine(Math.Round(34.56))
   Console.WriteLine("ZetCode has {0} characters", _
Len("ZetCode"))
End Sub End Module
```

FUNCTION IN VB
VERSUS
PROCEDURE IN VB

FUNCTION IN VB	PROCEDURE IN VB
A procedure that enclosed by the Function and End Function statements	A block of Visual Basic statements enclosed by a declaration statement and a matching End declaration
Used to perform a contain task	Helps to make the code readable, easy to modify and debug
A function is a specific type of procedure	A procedure is a generalized type of function

CALLING PROCEDURES

- In openroad, the frame or procedure that includes the callproc statement is referred to as the *calling frame* or *procedure*. The procedure you specify in the callproc statement is referred to as the *called procedure*.

- You usually call a procedure with one or more parameters. A parameter is the name of a local variable in the called procedure that is visible to code outside the procedure. You can use parameters to pass values between the procedure and the code that called it. Each time you call the procedure, you can pass different parameters, enabling one procedure to operate on different data. Passing parameters differs for the three types of procedures.

PASSING ARGUMENTS MECHANISM

- **Protection**. In choosing between the two passing mechanisms, the most important criterion is the exposure of calling variables to change. The advantage of passing an argument ByRef is that the procedure can return a value to the calling code through that argument. The advantage of passing an argument ByVal is that it protects a variable from being changed by the procedure.

OPTIONAL PARAMETERS

You can specify that a procedure parameter is optional and no argument has to be supplied for it when the procedure is called. *Optional parameters* are indicated by the Optional keyword in the procedure definition. The following rules apply:

- Every optional parameter in the procedure definition must specify a default value.
- The default value for an optional parameter must be a constant expression.
- Every parameter following an optional parameter in the procedure definition must also be optional.

NAMED ARGUMENTS:

- You learned how to write procedures with optional arguments and how to pass a variable number of arguments to the procedure. The main limitation of the argument-passing mechanism, though, is the order of the arguments. By default, visual basic matches the values passed to a procedure to the declared arguments by their order (which is why the arguments you've seen so far are called positional arguments
- This limitation is lifted by visual basic's capability to specify named arguments. With named arguments, you can supply arguments in any order because they are recognized by name and not by their order in the list of the procedure's arguments. Suppose you've written a function that expects three arguments: a name, an address, and an email address:

FUNCTIONS RETURNING CUSTOM DATA TYPES :

- The custom data type returned by a function must be declared in a Module. Suppose you need a function that returns a customer's savings and checking balance. The custom data type must be defined as follows:
- Type CustBalance
  BalSavings As Currency
  BalChecking As Currency
  End Type
- Then, we can define a function that returns a CustBalance data type as:
- Function GetCustBalance(custID) As CustBalance {statements}
  End Function
- The GetCustBalance() function must be defined in the same Module as the declaration of the custom data type it returns. If you place the function's definition in a Form's code, then you'll get a syntax error during the compilation of the project.

- When you call this function, you must assign its result to a variable of the same type. First declare the variable, then use it as shown here:
- Private Balance As CustBalance
  Balance = GetCustBalance(custID)
- Here, custID is a customer's ID (a number or-string, depending on the application). Of course, you must assign the proper values to the CustBalance variable's fields.
- Here's the simplest example of a function that returns a custom data type. This example outlines the steps you must repeat every time you want to create functions that return custom data types:

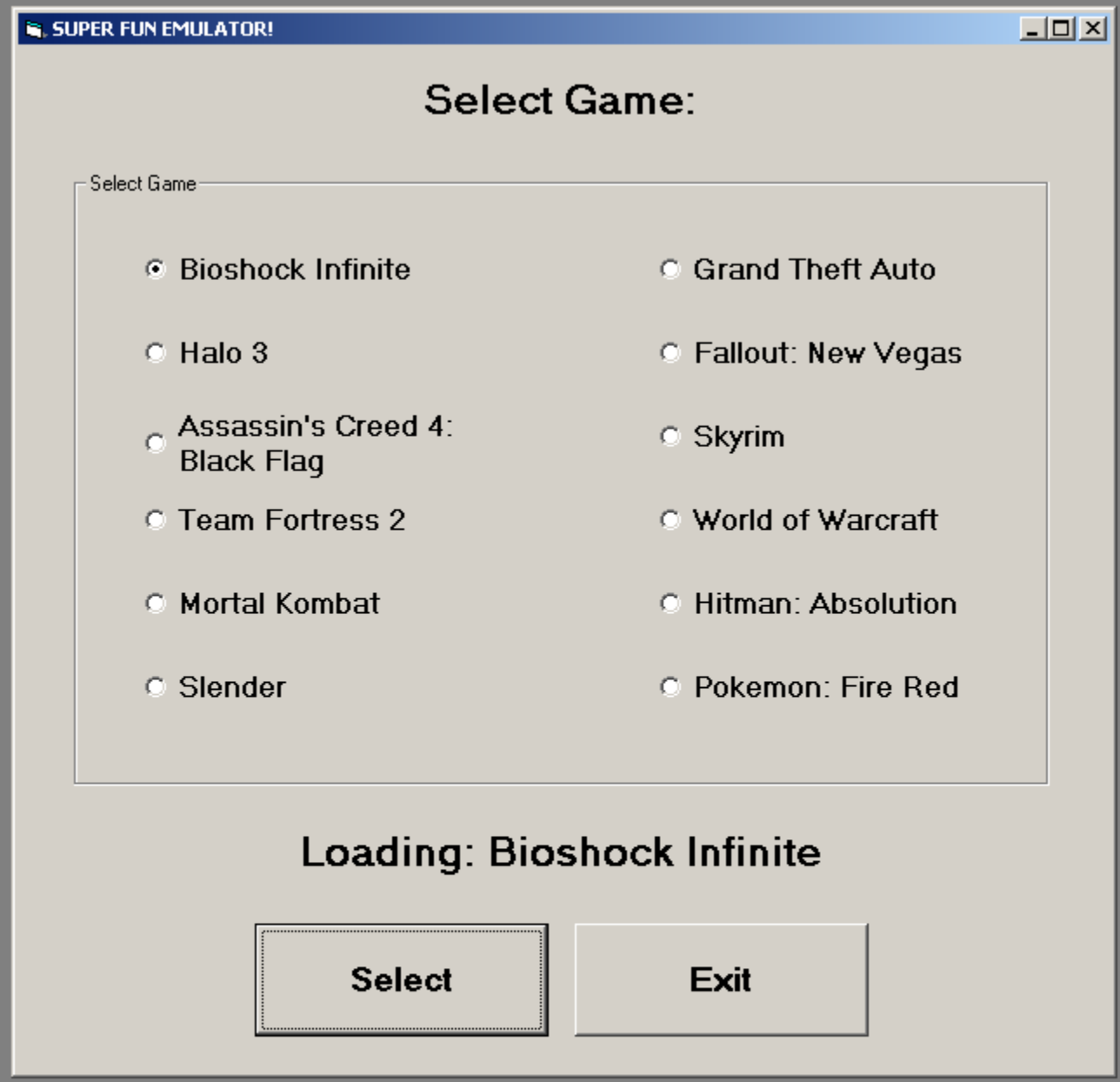
- 1. Add a new Module to the project (or open the application Module, if it exists). Insert the declarations of the custom data type. For example:
- Type CustBalance
  BalSavings As Currency
  BalCheckin9 As Currency
  End Type
- 2. Then implement the function. you must declare a variable of the type returned by the function and assign the proper values to its fields. The following function assigns random values to the fields BalChecking and BalSavings. Then, assign the variable to the function name, as shown next:

- Switch to the Code window of the Form from which you want to call the function and declare a variable of the same type. Assign the function's " return value to this variable and use it in your code. The example that follows prints the savings and checking balances on the Immediate window:
- For this example, I created a project with a Form and a Module. The Form contains a single Command button whose Click event handler is shown here. Create this project from scratch, perhaps using your own custom data type, to explore its structure and experiment with functions that return custom data types.
- In the following section I'll describe a more complicated (and practical) example of a custom data type function.

FUNCTIONS RETURNING ARRAYS:

- In addition to returning custom data types, Visual Basic 6 functions can also return arrays. This is an interesting possibility that allows you to write functions that return more than a single value. With previous versions of Visual Basic, functions could return multiple values by setting the values of their arguments. Let's say you need a function that calculates the basic statistics of a data set. The basic statistics are the-mean value (average), the standard deviation, and the minimum and maximum values in the data set. One way to declare a function that calculates all the statistics is the following:
- Function ArrayStats(DataArray()· As Double, Average As Double_ StdDeviation As Double, MinValue As Double, MaxValue As Double)

- To implement a function that returns an array, you must do the following:
- 1. Specify a type for the function's return value and add a pair of parentheses after the type's name. Don't specify the dimensions of the array to be returned.
- 2. In the function's code, declare an array of the same type and specify its dimensions. If the function should return four values, use a declaration like the following one:
  Dim Results(3) As Double
  The Results array will be used to store the results and must be of the same type as the function. Its name can be anything.
- 3. Before exiting the function, assign the array to the function name:
  ArrayStats = Results()
- 4. In the calling procedure, you must declare an array of the same type without dimensions:
  Dim Stats() As Double
- 5. Finally, you must call the function and assign its return value to this array:
  Stats() = ArrayStats(DataSet())


SUPER FUN EMULATOR!
Select Game:
Select Game
Bioshock Infinite
Halo 3
Assassin's Creed 4: Black Flag
Team Fortress 2
Mortal Kombat
Slender
Grand Theft Auto
Fallout: New Vegas
Skyrim
World of Warcraft
Hitman: Absolution
Pokemon: Fire Red
Loading: Bioshock Infinite
Select
Exit

ACTIVEX CONTROLS (VISUAL BASIC)

ActiveX controls are COM components or objects you can insert into a Web page or other application to reuse packaged functionality someone else has programmed. You can use ActiveX controls developed for Visual Basic 6.0 and earlier versions to add features to the **Toolbox** of Visual Studio.

To add activex controls to the toolbox

1. On the **tools** menu, click **choose toolbox items**.
2. The **choose toolbox** dialog box appears.
3. Click the **com components** tab.
4. Select the check box next to the activex control you want to use, and then click **ok**.

The new control appears with the other tools in the **toolbox**.