

Teaching Beginner And Advanced Programming With RobotBASIC

One of the many powerful features of RobotBASIC is its ability to *grow progressively* with the requirements of programmers as their skills evolve. This makes RB an ideal educational language for students as well as for teachers. RobotBASIC is also an ideal tool for achieving advanced projects since it has functionalities that provide powerful abilities while remaining nearly effortless to use.

Students nowadays are often frustrated and disheartened, despite having access to programming languages and tools far more powerful than anything in the past. In fact, it is precisely this power that thwarts today's students because it is often inseparable from complexities that make it difficult to progress from simple to advanced topics in gradual easy steps.

It is like being thrown in the deep end of the pool. Some may not be adversely affected by this style for teaching swimming. Most, however, will be irredeemably traumatized by the experience.

Viewed from the perspective of students (or the self-taught) it is easy to see why programming courses (and books) about today's languages are often confusing and discouraging. Due to the nature of these languages, it is not possible to teach them without having to delve into topics such as variable typing and scoping and in many cases even object instantiating and program modularization. You cannot escape having to learn about include libraries and even pointers and principles of what compilers do. All these topics, of course, are introduced before the student has even learned what a program is or why you need variables and modules.

This document will demonstrate that, despite being a powerful programming language, RobotBASIC avoids the aforementioned problems while also providing features that allow the learning process to be filled with gripping, relevant and meaningful projects very early on in the learning process.

1- The Beginner Advantage:

RB is a language with which beginner programmers get immediate feedback on ideas and actions without having to overcome a steep initial hurdle before they can start creating interesting projects.

Because RB has *versatility* and *flexibility* without compromising on *ability*, a teacher will find that it is an ideal programming language for implementing pseudo code with a syntax that, in itself, can be used as pseudo code. To illustrate the fact, let's look at the following scenario.

To introduce students to the concepts of acquiring *input* from the user and *printing* the results of an action on the screen you come up with a simple example and write on the board in normal English the steps in Pseudo Code 1 below.

```
Ask the user to Input his/her Name
Ask the user to Input his/her BirthYear
Calculate the user's Age using the BirthYear
Print on the screen "Hello " then the user's Name
Print on the screen "You are " then the user's Age then " Years Old"
End the program
```

Pseudo Code 1

The above pseudo code is extremely comprehensible. It might be a little too verbose for practical usage, however, students can immediately see the correlation between it and the following RB implementation:

```
Input "What is your name: ",Name
Input "What year were you born: ",BirthYear
Age = Year(Now())-BirthYear
Print "Hello ",Name
Print "You are ",Age," years old"
End
```

Program 1: Notice the line-by-line correlation with Pseudo Code 1.

It is generally not necessary to explain to students the above RB code after having been introduced to Pseudo Code 1. Besides, after a few examples, the teacher can even bypass the more *verbose* style and start using RB's syntax as *the* pseudo code due to the non-cryptic nature of the language.

Students can run RobotBASIC.exe from a USB Flash Drive *without any installation* or dependence on a particular machine. After that they would type the above program in the IDE's editor and then click RUN for an *immediate* result. Students will be able to use the program and observe its actions and will quickly appreciate how an idea has become a concrete running program with very little effort.

Notice the lack of any *extraneous* syntax that would only serve to confound and befuddle the student at this stage of learning. Consider the C++ implementation in Program 2 below. It performs the exact same action as Program 1. **Which code would you, as a teacher, rather present to your students as an implementation of the earlier pseudo code? Which code do you think your students would have a better chance of understanding? Which do you think has better correlation to the pseudo code?**

With the RB implementation, the student does not need (at this stage) to contend with totally meaningless (and only serves to confuse) concept of *variable typing*. With the C++ implementation the student has to struggle with, and the teacher has to try to explain, the *advanced concept of arrays* (for a string). Students will only be bewildered by the amount of *#include* files that were necessary to make the program work, not to mention the confusion the whole concept will cause in the first place.

It is hardly necessary to go on pointing out the reasons Program 2 is not a good introduction for a novice programmer. However, ponder this final point. **Which code do you think your students are more likely to be able to emulate and reproduce by themselves?** This aspect is extremely important for *self-taught programmers*. Many people may take on programming by themselves without official courses or having an instructor. RobotBASIC is an ideal language for the self-taught programmer because its syntax is easy to learn and its facilities allow for achieving powerful projects early on in the learning process.

```
#pragma hdrstop
#include <condefs.h>
#include <iostream.h>
#include <conio.h>
#include <time.h>
#pragma argsused
int main(int argc, char* argv[])
{
    int Age, BirthYear;
    char Name[40];
    cout << "What is your name:";
    cin >> Name;
    cout << "What year were you born:";
    cin >> BirthYear;
    cout << "Hello " << Name;
    Age = (1970+ time(NULL)/3600.0/24.0/365.0)-BirthYear;
    cout << "\nYou are " << Age << " years old";
    getch();
    return 0;
}
```

Program 2: Notice the almost total lack of correlation to Pseudo Code 1.

2- Advancement In Surmountable Stages:

RobotBASIC provides syntax that allows students to accomplish impressive results at any level of complexity along the learning curve while they are gaining more experience and sophistication.

Imagine if you had a vehicle that can be a tricycle while you are learning to ride and a bicycle when you are more capable. However, when the time comes where you need more power the very same vehicle is capable of performing as a motorcycle. RobotBASIC is such a versatile vehicle for the purpose of programming.

2.1- Free And Strict Variable Typing

It is a lot easier for students to appreciate the concept of variables without the restriction of having to declare their types. Notice how RB's flexibility allowed Program 1 to be simpler and better correlated to Pseudo Code 1. This is due to the lack of the restrictive and irrelevant (at this stage) *requirement* for variable type declarations before the variables can be used. The student is able to just use variables and RB takes care of assigning them a *type according to what is being stored in them*.

This is only logical. A variable is a storage space for holding data. It will only serve to confuse beginners to have to declare the type of variables when they are yet becoming familiar with the concept of variables. Since it is a storage place then just use it to store data. Why restrict it by declaring what is to be stored in it before using it.

Consider the following For-loop in RB:

```
For I=1 to 20
  J = I*3
Next
```

This is simple, intuitive, convenient, and much less typing than having to declare the variables I and J as integers before using them. After all, isn't it *obvious* that they need to be integers? Why can't the language just take care of assigning them types as required? Well, that is exactly what RB does.

Nonetheless, strict variable typing can help in reducing runtime errors and in forcing organizational skills. Once students are more experienced with the whole concept of variables and data types and have programmed a few simple programs, they are more able to cope with the restrictions and inflexibility of strict variable typing in return for more control over a program.

Can RB meet this level of programming? Definitely it can. With the command **Declare** RB reverts to becoming a strict variable typing system and will impose the requirement that variables should be declared before they can be used and that they should always be assigned an appropriate value type according to their declaration.

Consider Program 1. What do you think would happen if when running the program the user gave a string instead of a number for the *BirthYear*? Would the program fail? When?

The answer is yes, it will fail, but not when the user enters the inappropriate data type. It will fail on the line that calculates the age since it assumes the value *BirthYear* is a valid numerical value. Why did it not fail upon the entry of the wrong value type? The reason is that because strict variable typing has not been activated the variable *BirthYear* would take on the type of the data entered by the user of the program.

RB provides many ways for creating programs that are more tolerant of user input and therefore avoid errors caused by incorrect data entry. Nevertheless, this is an appropriate juncture for the introduction to variable typing. Add the following line at the *top* of the code in Program 1:

```
Declare Name="", BirthYear=0, Age=0
```

This will force RB to start performing strict variable typing and also will create the variable *Name* as a string type and assign it an initial value of an empty string and also the variables *BirthYear* and *Age* as integers with initial values of 0.

Will the program still fail when a user enters a string instead of a number for the birth year? When? Yes, it still fails, but it will fail when the user tries to enter a string for the *BirthYear*. So what is the advantage? It is a small one. Now the error is generated when the user enters the wrong value and not much later in the program where it may be harder to figure out why there is an error and what caused it.

In fact, for the purposes of Program 1, a more appropriate data type for *BirthYear* would be a string. This allows a user to enter anything but then the program can use one of RB's functions to detect the type of entered data and convert it to a number. This makes the program more immune to wrong data entry. Consider the following modified program:

```
Declare Name "", BirthYear "", Age 0
Input "What is your name: ",Name
Input "What year were you born: ",BirthYear
if IsNumber(BirthYear)
    Age = Year(Now())-ToNumber(BirthYear)
    Print "Hello ",Name
    Print "You are ",Age," years old"
else
    Print "Hello ",Name
    Print "You did not enter a numeric BirthYear"
endif
End
```

Program 3: The Blue lines are improvements to Program 1.

2.2- Various Levels Of User Interfacing Methodologies

Another area of programming where RB can serve at various levels of sophistication is in User Interfacing. As has been observed above the commands **Input** and **Print** are extremely *intuitive* and easy to use. The beginner programmer can use them to be able to write *useful* programs that interact with the user. With these commands there are no extraneous aspects that only serve to complicate the initial learning process.

However, with RB you can create progressively more complex user interfacing programs from the very simple to the visually pleasing but programmatically complex *Graphical User Interfacing* (GUI) style of programs.

With the **InlineInputMode on** directive and the **Print**, **Input** and **Waitkey** commands, RB programs will perform *standard user interfacing* in the style of *console* programs, much like the days of old before graphics were viable. Add this line at the top of Program 3 above and see how it changes the behavior of the user interaction:

```
InlineInputMode On
```

Without the **InlineInputMode** directive RB will perform at an intermediate level of user interfacing where the **Print** command still outputs to the console (old style) but **Input** and **WaitKey** perform their actions in the control panel (bottom of the screen) that utilizes an automated and simple to use GUI style interfacing.

As programming requirements become more demanding the programmer can make use of the more involved, yet not too complicated facilities of RB such as **GetKey**, **GetKeyE**, **KeyDn()**, **ReadMouse**, **JoyStick**, **JoyStickE**, **xyString**, **xyText**, **Sound** and **PlayWav**.

For the ultimate in sophistication RB provides numerous *GUI* elements that are powerful to use but in RB are not too hard to understand and utilize unlike most other languages.

With **FilePrompt()**, **FileSave()**, **DirPrompt()**, **StrInput()**, **MsgBox()**, **TextBox()**, **StrongBox()**, **ErrMsg()** and **xyInput()** the programmer can create with one line of code a myriad of different types of powerful and sophisticated GUI dialog boxes that would require numerous lines of code in other languages not to mention the complexity of setting them up in the first place.

The following two lines program prompts the user for a text file name and then reads the specified file (if any) and creates a dialog box that allows the user to browse the text as shown in Figure 1. The lines of code below are the entire program. Imagine what it would take to do the same with other languages.

```
FN = FilePrompt("*.Txt")
if FN != "" then TextBox(FN,"When finished Push OK or Cancel")
```

Program 4: Two Lines Program that performs a powerful action.

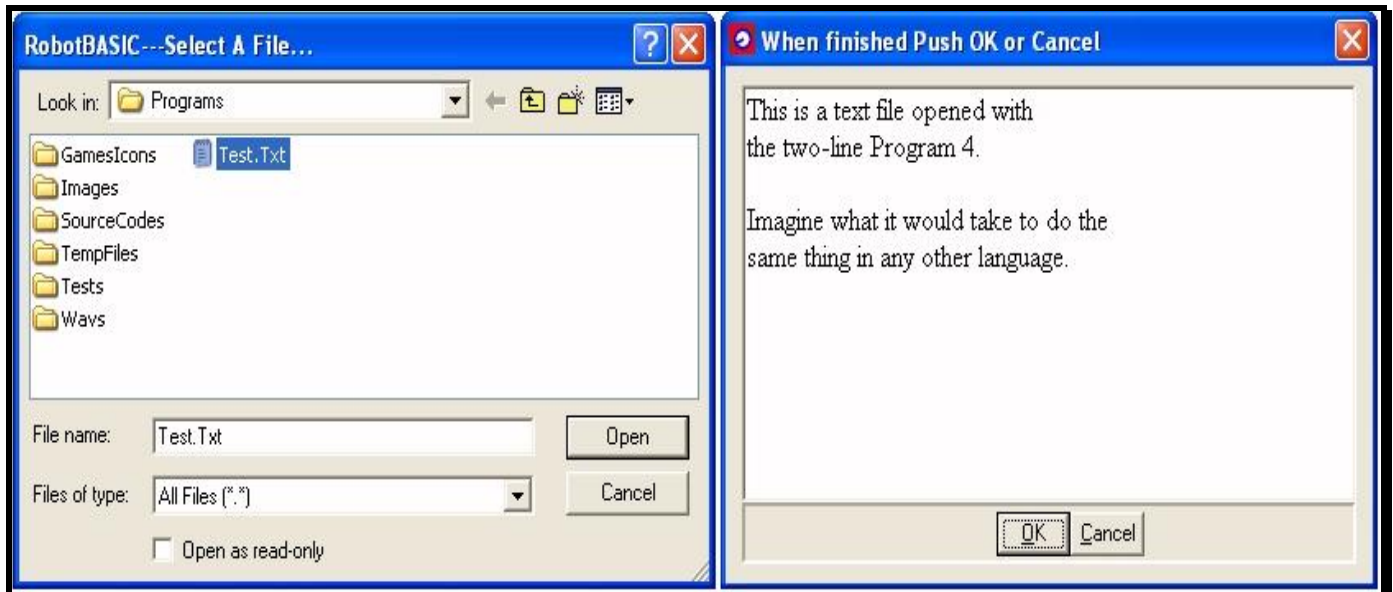


Figure 1: GUI Dialog Boxes created by Program 4.

RobotBASIC also provides extremely easy to program constructs for implementing and utilizing Push Buttons, Edit Boxes, Check Boxes, Radio Buttons, Memo Boxes, Spinners, Sliders and much more. See the [Graphical User Interfacing](#) section in the [RobotBASIC help file](#).

2.3- Examples Of Progressive Sophistication

As an example for how RB can be used in a progressively more sophisticated but yet extremely understandable and easily surmountable manner, we will apply further modifications to Program 3. You have already seen how it is an advance over Program 2 as far as using variables is concerned. Now we will see how the program can be made to be progressively more complex as far as User Interfacing is concerned.

Program 5, utilizes RB's easy to use dialog boxes (bolded text) to obtain input from the user and to display the results. Figure 2 shows the possible resultant dialogs.

```
Name      = StrInput( , "What is your name:")
BirthYear = StrInput( , "What year were you born:")
if IsNumber(BirthYear)
    Age = Year(Now())-ToNumber(BirthYear)
    msg = "Hello "+Name+crlf()+"You are "+Age+" years old"
    ErrMsg(msg, "RobotBASIC", MB_OK|MB_INFORMATION)
else
    msg = "Hello "+Name+crlf()+"You did not enter a correct BirthYear"
    ErrMsg(msg, "RobotBASIC", MB_OK|MB_ERROR)
endif
End
```

Program 5: One level of improvement to Program 3 by using GUI Dialog Boxes.



Figure 2: Dialogs resulting from running Program 5.

Dialog Boxes can be useful, easy and quick to use. Nevertheless this style of user interfacing has limitations. To illustrate yet another step of sophistication we will use Push Buttons and Edit Boxes.

Program 6 is at the intermediate level, utilizing much of RB's power but yet is not overly complicated. Notice how creating Edit Boxes and Push Buttons is not inordinately more complicated than using **Input** or dialog boxes but yet provides a lot more sophistication.

The reason that using such a powerful construct is so simple is because, *again*, RB provides progressive steps of complexity. Notice that there was no need whatsoever to use Event Driven programming to handle the push buttons. Does RB provide for Event Driven programming too? Yes it does. This is a very advanced concept and we will discuss it among other more advanced concepts later. But, consider how RB has made it possible to utilize an advanced tool with a *surmountable step in complexity*.

```

ClearScr gray
xyText 55,10,"What is your name:", ,10,fs_Bold,,gray
xyText 8,40,"What year were you born:", ,10,fs_Bold,,gray
AddEdit "Name",200,8,200 \ AddEdit "BirthYear",200,38,80
AddButton "Calculate",100,80,100
while true
  if LastButton() == "" then continue
  if IsNumber(GetEdit("BirthYear"))
    Age = Year(Now())-ToNumber(GetEdit("BirthYear"))
    msg = "Hello "+GetEdit("Name")+crlf()+"You are "+Age+" years old"
    Icon= MB_INFORMATION
  else
    msg = "Hello "+GetEdit("Name")+crlf() \
      +"You did not enter a correct BirthYear"
    Icon = MB_ERROR
  endif
  ErrMsg(msg,"RobotBASIC...",MB_OK|Icon)
wend

```

Program 6: Another advance over Program 3 by using GUI components. Also the program is shown, as it would appear in the RobotBASIC IDE, with syntax highlighting.

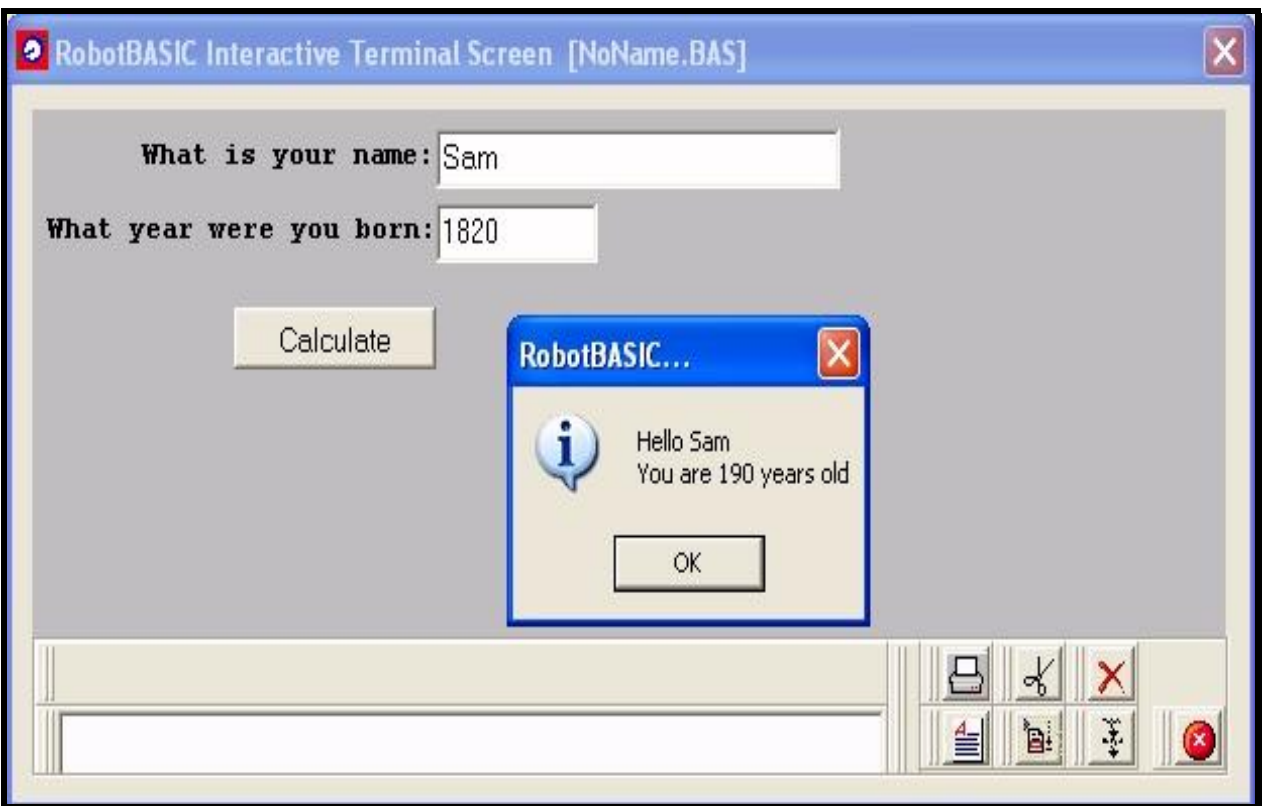


Figure 3: What Program 6 looks like when running.

2.4- Various Levels Of Program Structuring

Another way RB facilitates progressive advancement is with program *modularity*. While students are still at the initial stages, program formatting is not relevant or meaningful. Consider Pseudo Code 1. There is nothing in it that pertains to program structuring. At this stage it is an altogether non-germane concept. Thus, when students are shown Program 2 (C++ code) they would be quite confused as to all the extraneous code that does not have any correlation to the pseudo code.

RobotBASIC allows for a free flowing program structure that enables the creation of code such as in Program 1. Notice how there is no requirement whatsoever to create any type of modularity that would have only served to puzzle the student.

This *freedom* is helpful in many situations. Consider for example the code in Program 4. Programming is about achieving a task quickly and efficiently. With RB you can do so without any *imposed* requirements and restrictions that do not pertain to the issues of algorithmic development.

Nevertheless, program structuring and modularity become necessary as projects grow in complexity. As with user interfacing RB provides various levels of sophistication to serve programmers at any stage of progression. Initially students learn to create lines of code to achieve a single task. Later, they should be introduced to the concept of *modular programming*. It is a lot easier to do so with RB than with other languages.

The **GoSub** statement combined with *Labels* provides an intermediate level for achieving modularity that is easy to understand and does not entail hard to explain (initially) side effects. With **GoSub** subroutines students do not have to contend with the concept of *Local* and *Global variables*. They can create code just as they have been doing all along and then surround it with a Label and a **Return** statement (easy to comprehend). Then they use these *modules* by calling them with the *succinct GoSub* statement.

Notice how the complication of local scoping is dispensed with at this particular stage. It would only have introduced a level of complication that would not have any relevance to students. They do not need to contend with understanding the concepts of *by reference and by value parameter passing*. All these nuances are not something that would initially be relevant.



One of the most important aspects of education is to enable students to *correlate* their knowledge. Correlation of newly introduced concepts to previously acquired knowledge helps cement the new information and due to its relevance it is easier for students to appreciate the reasons for the new levels of sophistication.

When students start creating more complicated programs and experience the issues and complications of having only *global variable scoping* they would quickly and *intuitively* grasp the power that **Call/Sub** subroutines provide. Explaining the concepts involved would be extremely easy since by then they would have a *reason* and a *case study* that comes from their *own experience* rather than from a *contrived meaningless* dry example in a classroom session.

Therefore, again, RobotBASIC provides student with the *flexibility to enable growth in sophistication in surmountable steps* while at every stage only concepts significant to achieving the tasks at the current level are required.

2.5- Examples Of Progressive Program Structuring

Lets consider how progressive program structuring can be applied in practice. Consider the following Pseudo code to teach students how to swap the values of two variables.

```
Assign 10 to A and 40 to B
Store value of A in a temporary variable
Put value of B in A then the value in the temporary variable into B
Print A then B
```

Pseudo code 2: Swapping two variables.

Here is the RB program that implements Pseudo Code 2.

```
A = 10 \ B = 40
Temp = A \ A = B \ B = Temp
Print A;B
```

Program 7: Swapping two variables.

Notice how RB's free structure enables a simple and quick implementation and allows it to be almost a 1-to-1 correlation. Notice how the student can concentrate on the *algorithm* rather than on distracting nuances that serve no purpose while in the process of trying out the *algorithmic logic*.

Later modularity is introduced in a simple and easy step. Students will have no difficulty in appreciating the relationship between Programs 8 and 7. Also they can easily grasp the concept of modularity and the need for it without having to contend with additional concepts before they are required.

```
Main:
  A = 10 \ B = 40      \ GoSub SwapAndPrint
  A = 20 \ B = "test" \ GoSub SwapAndPrint
  A = 4  \ B = 7.5    \ GoSub SwapAndPrint
End
//-----
SwapAndPrint:
  Temp = A \ A = B \ B = Temp
  Print A;B
Return
```

Program 8: Swapping two variables using **GoSub** subroutines.

For many projects using **GoSub** subroutines would suffice. Nevertheless, students would soon run into the limitations of global variable clashing and having to assign variables before the call is made to the subroutine. **Learning how to overcome these limitations is in itself a valuable knowledge that many computing courses fail to teach.** This is why many programmers nowadays are not able to program in Assembly (see later for more about this).

At this point students would have a practical appreciation for the power and convenience of **Call/Sub** subroutines. They would quickly and *naturally grasp* the concepts of local scoping and *by value parameter passing* since they have *experienced the need* for them *through practice in real situations*. So when the students see the code in Program 9 a teacher would have very little difficulty explaining the related concepts.

```
Main:
  Call SwapAndPrint(10, 40)      //notice two integers
  Call SwapAndPrint(20, "test") //string and integer
  Call SwapAndPrint(4, 7.5)     //integer and float
End
//-----
Sub SwapAndPrint(A,B)
  Temp = A \ A = B \ B = Temp
  Print A;B
Return
```

Program 9: Swapping two variables using **Call/Sub** subroutines.



Program 9 demonstrates yet another instance of RB's power and flexibility. The way RB handles variables allows for the subroutine to work with the parameters being of any type and even of different types. *In any other language you would have quite a difficult time writing a program that carries out the actions of Program 9.*

Parameter passing by reference is another concept that is best explained when a real practical need for it arises. Here is the swapping routine implemented with by reference parameter passing just to show how it is done in RB.

```
Main:
  A = 10 \ B = "test"
  Call SwapThem(A, B) \ Print A;B
End
//-----
Sub SwapThem(&VarA, &VarB)
  Temp = VarA \ VarA = VarB \ VarB = Temp
Return
```

Program 10: By Reference Parameters in **Call/Sub** subroutines.



Programs 7 to 10 have all been artificial (yet educational) exercises. In RobotBASIC there is absolutely no need to write a subroutine to swap variables, since there is a command that would do it for you painlessly.

```
A = 10 \ B = "test"
Swap A,B \ Print A;B
```

Program 11: Swapping two variables using RB's **Swap** command.

3- Learning With RobotBASIC Is More Enjoyable:

As you have seen in the previous sections, RB has many advantages for teaching programming in the *traditional* way using normal examples. However, RB also has another advantage that should be a *compelling incentive* for using RB.

In most languages doing anything with graphics or simulations is not possible at the initial learning stages due to the complexity required to achieve them. Thus, a student is obliged to try out boring, dry, and in many ways contrived code examples illustrating the concepts being learned.

Traditionally, when teaching Looping, a teacher is restricted to doing things such as printing out numbers or using simple text input and output. Imagine how much *more entertaining, visually exciting* and practical it would be to teach looping using a *Simulated Robot*, or an *Animated Graphic*.

Which do you think is a more exciting example for using a While-Loop, Program 12 or Program 13? Moreover, consider which is more complicated?

```
x = 0
while x < 20
  x++ \ print x
Wend
```

Program 12: Traditional Looping Example

```
rLocate 400,300
while !RBumper()
  rForward 1
wend
```

Program 13: Looping Example Using RB's inbuilt Robot Simulator



Since RB has an *inbuilt robot simulator*, you can make use of this *motivational* and enjoyable tool to make teaching looping and other programming constructs much *more relevant and meaningful* than with the dry and contrived traditional method of printing out number sequences.

Creating *animated graphics* with RB is amazingly simple that using it at the initial learning stages is very viable besides introducing a level of visual stimulation that serves to *grip* the students' attention and to raise their level of *interest* in the subject being taught.

```
Flip on \ Linewidth 5
for i=10 to 500
  CircleWH i,i,50,50,red
  flip \ clearscr
next
```

Program 14: Looping Using Graphics and Animation.



Program 14 uses RB's powerful, yet simple to use, commands to animate a circle on the screen while teaching the principles of For-Loops. Since using the graphical commands in RB does not in any way impose additional complexity, it is possible to use them to make a For-Loop more exciting but even further, more relevant.

4- Attainable Advanced Programming:

RobotBASIC is replete with commands and functions that make it possible to accomplish projects that would be quite difficult for an average programmer to achieve with most languages. [With few lines of code a novice programmer can create projects that would be extremely difficult to do with other languages even for an accomplished programmer.](#) With most of these facilities the commands are as easy to use as **Print** and **Input**, but yet provide computing power that would require months and hundred of lines of code to achieve otherwise, not to mention the necessary advanced skills.

4.1- Multimedia

With RB you can play video and audio clips in numerous formats (mp3, wav, midi, avi, mpeg, wmv and much more) with programmatic control over the playback. You can also record audio. Additionally, for the purposes of gaming, you can play multiple audio sounds simultaneously. Watch this video to see RB's [multimedia in action](#).

4.2- Bitmap and Graphics Animation

With a comprehensive Bitmap and Screen manipulation set of commands you can create professional looking games and simulations. Some of the bitmap commands allow you to manipulate images in ways that would require highly advanced programming skills. Try out some of the [games on this web page](#). Also watch this video for an [online tutorial](#) on how to do animation with RB.



Figure 4: Sample Games

4.3- Robot Simulator

The inbuilt Robot Simulator allows for experimenting with numerous robotic projects. With instrumentation such as line sensors, infrared sensors, bumpers, GPS, compass and much more you can simulate projects that would cost thousands if they were to be attempted with a real physical robot, not to mention the frustration caused by damages due to lack of experience while learning.

See this [online tutorial](#) for how to use the robot simulator. Also, there are two books that teach how to use the robot simulator in RB, [Robot Programmers Bonanza](#), and [Robots In The Classroom](#). Also [RobotBASIC Projects For Beginners](#) is a very good book for young students and novices.

Additionally watch this video to see the [robot simulator in action](#). See this one for a more [general discussion](#) on the simulator. This [impressive video](#) will be of interest as well.

4.4- TWAIN Device Capture

With RB you can interface with Cameras, Scanners, Web Cams and much more, to acquire images that later can be manipulated to do things such as color recognition, object detection, movement detection and much more. Watch this video for an example project that utilizes a Web Cam to make a [puppet appear to be alive](#).

4.5- Hardware Interfacing

RB makes it extremely simple to achieve projects that involve interfacing a PC with external electronics hardware to design control projects. With Parallel port, Serial Port and USB port commands you can communicate with Microcontrollers, Digital I/O controllers, Servo Motor Controllers, Serial devices, and much more. You can also use Bluetooth and Zigbee wireless devices to create remote control projects. See this [PDF](#) document for more details (also [this](#) and [this](#)). Also watch this video to see how RB can be used to control a [Space Station Model](#).

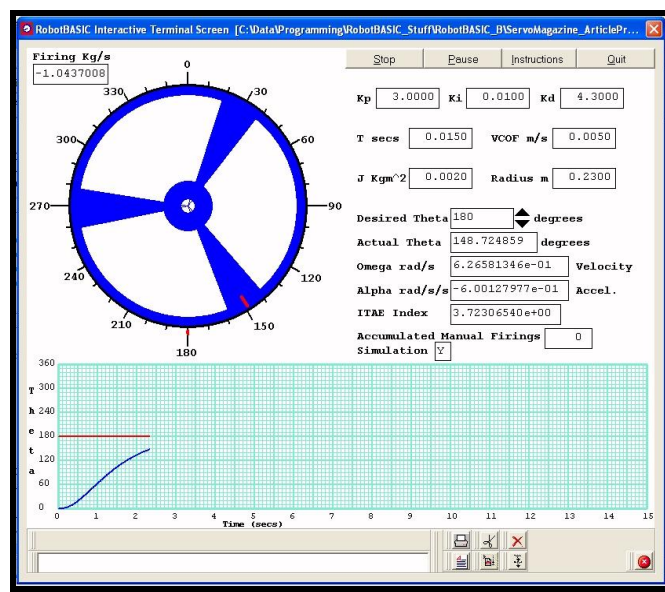


Figure 5: Sample Hardware Control Project

4.6- Internet

Imagine what it would take to write programs that can communicate over the LAN, WAN or the Internet using the SMTP, TCP or UDP protocols. With RB's functions you can do so just as easily as sending bytes over a serial connection. See this [PDF](#) document for more details.

4.7- Matrices

Matrix creation and utilization in RB is most powerful. You can do matrix operations such as inversion, determinant calculation, sorting, and much more. You can read a bitmap or text file into a matrix. With one command you can save an entire matrix to a disk file and read it back and much more. See the [RobotBASIC help file](#) for the myriad of operations you can carry out with matrices. Also see Program 15 for code using matrices.

4.8- 3D Graphics

The Three Dimensional Graphics Engine in RB is simple to learn and use with sufficient power to create impressive 3D-graphics projects. See this video for [some examples](#). Also try playing with this [Rubik's Cube program](#) from our site. Additionally, examine Program 15 below.

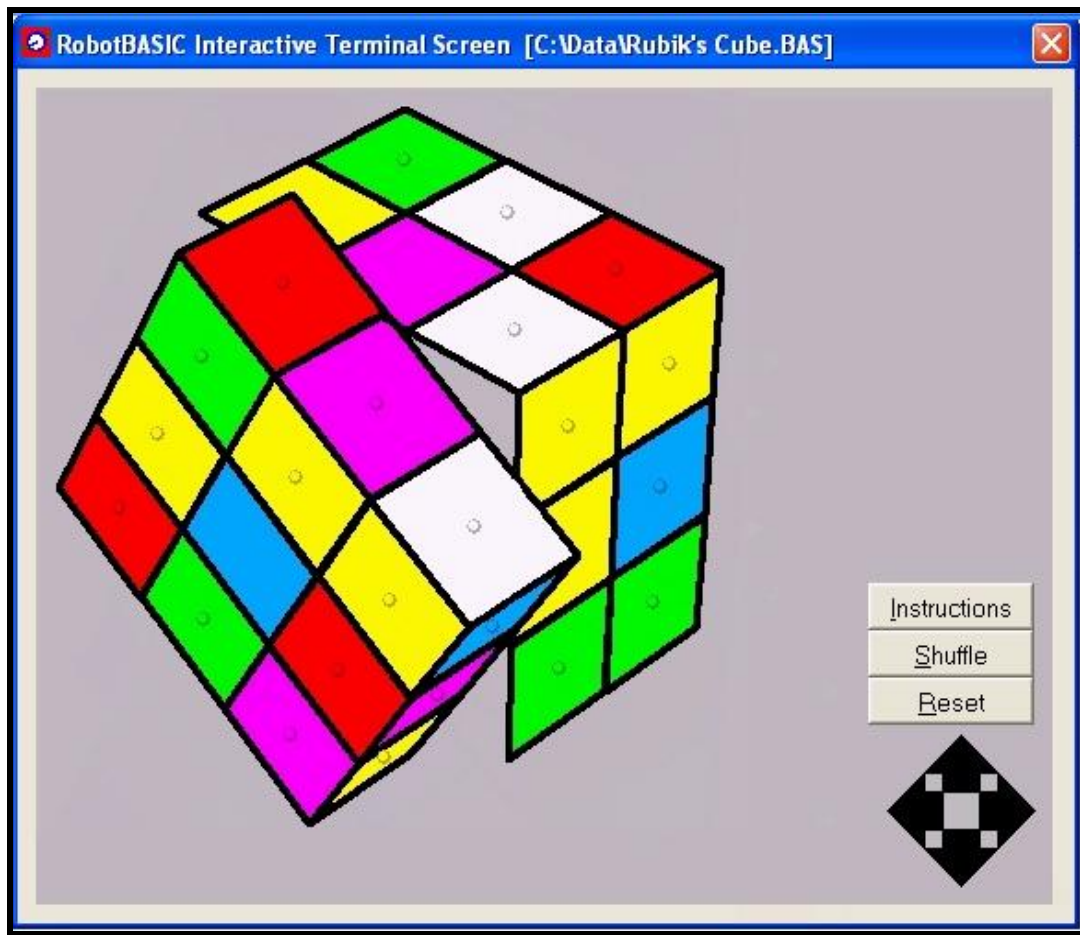


Figure 6: An example of what can be achieved with RB's 3D-Graphics Engine.

Program 15 is an illustration of the power and simplicity of the RobotBASIC 3D-Graphics engine combined with the Matrix functionality. It creates a 3D-Graphics Animation. Notice the Matrix operations and **consider the code size**. See the output screen in Figure 7.

```

data Eye;120,pi(.25),pi(.35),1550,400,300 //rho,theta,phi,d,Cx,Cy
x = 20 \ data points;0,0,0,0,0, x,0,0,0,0, 0,x,0,0,0, 0,0,x,0,0
x = 15 \ data points;x,0,0,0,0, 0,x,0,0,0, x,x,0,0,0, x,x,x,0,0
dim Points[10,5] \ mCopy points,Points //make an array of vertices
flip on
while true
  ge3dto2da Points, Eye //calculated screen coordinates
  for i=1 to 3
    line Points[0,3],Points[0,4],Points[i,3],Points[i,4],2,0
    line Points[6,3],Points[6,4],Points[3+i,3],Points[3+i,4],1,red
    if i<3
      line Points[0,3],Points[0,4],Points[5+i,3],\
        Points[5+i,4],i,Blue+i
    endif
  next
  line Points[7,3],Points[7,4],Points[6,3],Points[6,4],1,green
  flip \ ClearScr \ Eye[1] = Eye[1]+.01
wend

```

Program 15: Matrices and 3D-Graphics Animation Program.

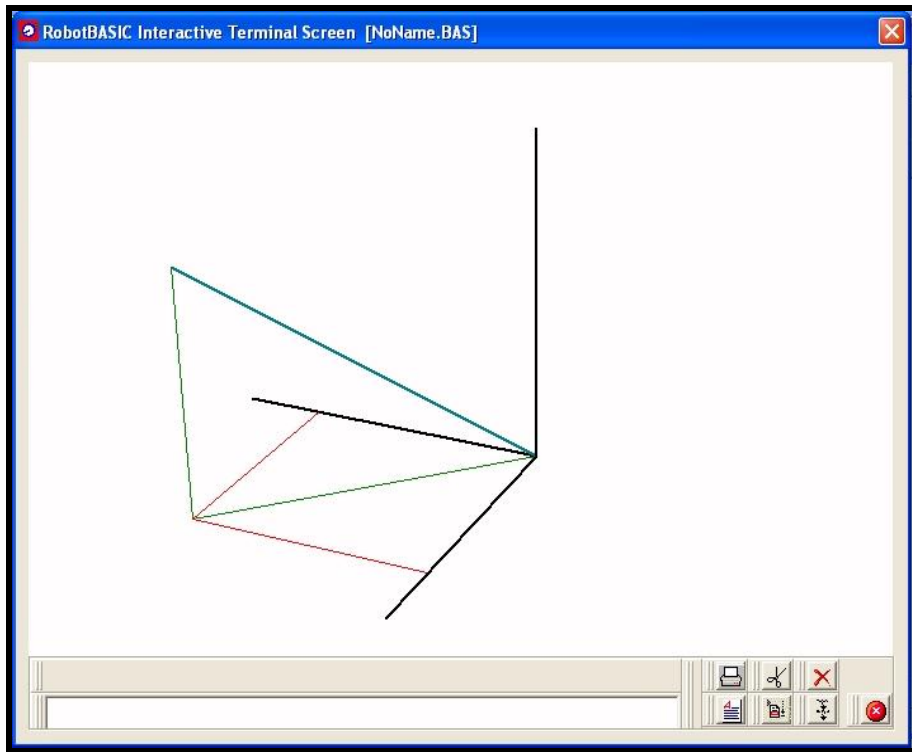


Figure 7: Screen output from Program 15. This is a snapshot; the program is an animation.

5- Going Further:

- Since RobotBASIC is also a compiler, a programmer can create fully standalone executable programs that require no installation and can run without requiring the RB IDE. This allows you to distribute programs to clients and end users with a minimum of hassle.
- The **#Include** facility allows you to create libraries of subroutines to be used by other programmers. This allows a teacher to create code for a classroom as an **#Include** library and the students would be able to use it with minimal fuss. Additionally for the professional these libraries can even be saved as binary files to help hide the code. Once compiled as part of a program the resultant executable will not require the library any longer since it would be “included” in the main program.
- RobotBASIC has an extensive set of functions and commands to do File and Directory manipulation. Along with the Low Level File I/O functions, you can do all sorts of data saving and manipulation. This Low Level File I/O can be Direct or Sequential Access and ASCII or Binary.
- There are numerous commands and functions for manipulating the Clipboard to export and import data to and from other programs. The data can be text or bitmap images.
- You can use the Spawn command to have RB run other programs. For example with one line of code you can write a program that starts Internet Explorer and goes to a particular URL. Try this *program*.

```
spawn("C:\Program Files\Internet Explorer\iexplore.exe", "www.robotbasic.com",p_nowait)
```

- RobotBASIC’s syntax is quite malleable. You can use C++ style syntax instead of a normal BASIC syntax.

<u>C++ style</u>	or you can say	<u>Standard BASIC style</u>
if((a != b)&&(c == t))		if a <> b AND c = t
V++		V = V + 1
X += y		X = X+ y

The C++ style can be helpful when students start transitioning to C++ or Java later in their careers.

- There are commands and function in RB that manipulate variables with indirection. This helps introduce students to *pointer-like* manipulation of variables, in addition to being helpful in doing actions that require some level of variable indirection. See **vType()**,**varType()**, **varValue()**, **VarsList()** and **VarSet**.
- In RB, handling input from users with the GUI components (e.g. Push Buttons, Check Boxes) can be done without having to use Event Driven programming. However, the event driven model of programming can provide power and abilities that are hard to achieve with the normal procedural model.

RB can also be programmed using the event driven model. With the **On[control]** statements (e.g. **OnButton** or **OnEdit**) you can have RB jump to (and later return from) a designated *handler* subroutine (either **Call/Sub** or **GoSub** ones) every time there is a push of button or a change in the text in an edit box and so forth.



Other languages provide only the event driven model for handling GUI components. Since RB provides the ability to process GUI components with both the normal procedural and the event driven models, using GUI components can be introduced to and used by programmers at various levels of experience.

In Program 16, notice how the Main module sits in an empty loop doing nothing. This is because all the action is accomplished with the *Event Handlers* that are invoked when an action is taken with the GUI components.

```

Main:
  GoSub Initialization
  while true
    //do nothing just wait for events to be handled
  wend
end
//~~~~~
Initialization:
  AddEdit "Edit1",60,10,80,0 \ AddCheckBox "Check1",170,10
  AddSlider "Slider1",280,10,200 \
  OnEdit eHandler \ OnCheckBox cHandler \ OnSlider sHandler
  OnAbort aHandler \ OnKey kHandler
Return
//~~~~~
Sub kHandler() //this is a Sub subrotuine with local variable scoping
  n = LastKey()
  if n == kc_Esc then gosub aHandler
  OnKey kHandler
Return
//~~~~~
Sub eHandler()
  n = lastedit()
  xystring 10,230,n,":",getedit(n),spaces(50)
  onedit eHandler
return
//~~~~~
cHandler:
  n = lastcheckbox()
  xystring 10,320,n,"=",getcheckbox(n);getcheckboxcaption(n)\
    ,spaces(50)
  oncheckbox cHandler
return
//~~~~~

```

Program 16: Event Handling Programming Style (continued).

```
sHandler:
  n = lastslider()
  xystring 10,350,n,"=",getsliderpos(n),spaces(50)
  onslider sHandler
return
//~~~~~
aHandler:
  an=ErrMsg("Do you want to abort?","Test",MB_YESNO|MB_QUESTION)
  if an==MB_YES then Exit
  onAbort aHandler
Return
```

Program 16 (continued): Event Handling Programming Style.

6- Wealth Of Resources On The RobotBASIC Web Site:

Visit www.RobotBASIC.Com for numerous resources that will help in learning and teaching RB. There are *six* [Lesson Plans](#) (with [online videos](#)) ready to be used by an instructor. There are links to videos, links to books about RB and many PDF documents (white papers).

In addition to the files needed to run RobotBASIC, the [download Zip File](#) has numerous *demo programs, games and utilities* that you will find extremely useful.

Make sure to browse the many pages on the site and read the [sample chapters](#) (also [this](#) and [this](#)) of the three books that you can use to help teach programming in general and robotics in particular.

7- RobotBASIC As An Assembly Language Trainer:

Using the native machine level language (Assembly) to program microcontrollers and microprocessors is the most effective way of creating the smallest, most efficient and fastest programs for a particular device. However, most programmers nowadays experience difficulty programming in Assembly languages. This is quite fascinating, since in many ways assembly programming should in fact be simpler than using higher-level languages. This dilemma is a puzzling phenomenon, until you consider the reasons.

When you consider what higher-level languages do for the programmer you begin to understand the reasons behind the abovementioned dilemma. Variable scoping, subroutine parameter stacking, loop stacking, and so forth are all constructs that eventually have to be implemented in Assembly languages. However, since higher-level programmers never have to contend with that level of detail they have become too isolated from the entailed nuances.

Many programmers have become *lazy* due to the conveniences that higher-level languages provide. Of course, for the purposes of most projects, the higher-level languages provide a more efficient tool. Nonetheless, they have served to *atrophyed the programming skills* necessary for programming at the machine level.

Programming structures such as For, While etc. are all beautiful and convenient as well as intuitive constructs that help programming be closer to using natural languages. However, they are not available at the machine level. Assembly language programming is limited to simple conditional checks with Jumps (JNE, JZ etc.). There is no such thing as local and global scoping of variables, there are only memory locations. There is no such thing as subroutines with parameter passing, there is only calling a memory location with a return (some assemblers don't even provide that).

In reality Assembly language programming is nothing but a series of simple comparisons with a JMP and a Call/Ret construct. That is it. Some assemblers go a little further and allow for named memory locations, which helps in making programs a little less cryptic and more tolerant to changes.

Therefore, when you consider all these limitations from which the higher-level language programmer is shielded, you can easily understand the difficulty they experience when they have to program at the Assembly level. Even if they were able to understand the concepts with which they've never had to contend, they would still find it hindering to have to do themselves what the compiler has been doing for them all this time.

Many, resort to using higher-level languages to program microcontrollers because they are incapable of making the mental adjustment. Unfortunately, this means that they will never be able to create programs for these devices at the most efficient and fastest possible level.


Furthermore, teaching assembly programming can be quite difficult. Operating systems for PCs nowadays do not even allow it. Additionally, using a microcontroller with all the required connections and so forth (as well as the costs) can be quite discouraging.

RobotBASIC has a solution. Why not use RB as an ***Assembler-Simulator***. With RB it is still possible to program with Goto (JMP) and simple If-Then-Goto statements (JNZ, JGE). Also the GoSub statement and Labels are akin to the assembly Call/Ret constructs.

Therefore, if you limit students to using these constructs, you can utilize RB as if it were an assembly language **of sorts**. Thus RB can act as a very convenient, simple, painless, and cost free ***Assembler-Trainer***.

Think about this. RB can be used in a manner that would be a ***progressive step towards*** preparing the student to programming microcontrollers without the frustrations and cost involved.

This is a concept that may elude many people and ***may even be controversial***. However, with a little consideration and analysis, you will find that in fact using Goto and Gosub with simple If-Then-Goto constructs is really all that Assembly languages are.

 ***RobotBASIC allows progressive advancements in surmountable steps from the very simple to the quite sophisticated.*** This is a major advantage while teaching programming.