RobotBASIC An Effective Educational Programming Platform

Teaching a person who has never ridden a bicycle how to ride one using a motorcycle is foolhardy, even if their aim is to eventually ride a motorcycle. Likewise, teaching programming to beginners using Java/C++ (etc.) is counterproductive.

The amount of time and effort spent in teaching the syntax ***to even begin*** writing a program is tremendous, ***and there is no way to bypass it***. There is no way to start using a simplified version and then evolve in C++ (et al). You have to start in the deep end. There is no way to avoid:

1- Curly brackets, semicolons, unnecessary parenthesis, etc.
2- Variable typing, object instantiating, parameter passing, etc.
3- Counterintuitive or very advanced methods for getting or presenting data to users.
4- Concepts like #include, #import, etc. etc.
5- Header files, libraries, compiling, linking, etc.
6- Private/Public/Global etc. variables/methods etc.

Thus, the beginner student gets bewildered, confused, disheartened and overwhelmed by the amount of work it takes to write a useless and totally unexciting program which necessarily has to be kept so due to the complexity needed for doing anything actually useful.

Even if the students are able to wade through the quagmire of all the seemingly meaningless syntax, they will have to wait for a discouragingly long time through the learning process before they can do anything that is remotely interesting or rewarding.

Look at this standard Java code for a program that draws a red square that follows the mouse as it is dragged around by the user while pressing the left mouse button.

```
package painting;

import javax.swing.SwingUtilities;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.BorderFactory;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseMotionListener;
import java.awt.event.MouseMotionAdapter;

public class SwingPaintDemo3 {
      public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }

    private static void createAndShowGUI() {
        System.out.println("Created GUI on EDT? "+
        SwingUtilities.isEventDispatchThread());
        JFrame f = new JFrame("Swing Paint Demo");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.add(new MyPanel());
        f.pack();
        f.setVisible(true);
    }
}

class MyPanel extends JPanel {
    private int squareX = 50;
    private int squareY = 50;
    private int squareW = 20;
    private int squareH = 20;

    public MyPanel() {
        setBorder(BorderFactory.createLineBorder(Color.black));
        addMouseListener(new MouseAdapter() {
            public void mousePressed(MouseEvent e) {
                moveSquare(e.getX(),e.getY());
            }
        });
        addMouseMotionListener(new MouseAdapter() {
            public void mouseDragged(MouseEvent e) {
                moveSquare(e.getX(),e.getY());
            }
        });
    }

    private void moveSquare(int x, int y) {
        int OFFSET = 1;
        if ((squareX!=x) || (squareY!=y)) {
            repaint(squareX,squareY,squareW+OFFSET,squareH+OFFSET);
            squareX=x;
            squareY=y;
            repaint(squareX,squareY,squareW+OFFSET,squareH+OFFSET);
        }
    }

    public Dimension getPreferredSize() {
        return new Dimension(250,200);
    }

    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawString("This is my custom Panel!",10,20);
        g.setColor(Color.RED);
        g.fillRect(squareX,squareY,squareW,squareH);
        g.setColor(Color.BLACK);
        g.drawRect(squareX,squareY,squareW,squareH);
    }
}
```

There is a lot to programming other than syntax. Algorithm development, logical thinking, software engineering and coding practices are the most important skills in programming. These skills can be taught regardless of the language being used. So why not teach them using a platform that can be used immediately ***without waiting until a person knows how to program before they can learn how to program*** (notice the paradox).

Consider the following code that accomplishes the exact same task as the Java program above.

```
FirstTime = true
ox = 400 \ oy = 300
w = 40 \ h = 40
SetMousePos ox,oy
Flip on
xyText 0,0,"Click the mouse and move it around"
SaveScr
while true
  ReadMouse x,y,mb
  if FirstTime || (mb && (ox != x || oy != y))
     RestoreScr
     RectangleWH x,y,w,h,red,red
     ox = x \ oy = y
     Flip
     FirstTime = false
  endif
wend
```

Which code do you think you can explain to students with any hope of not having them run out of your classroom while screaming and ducking for cover?

The above code would be considered as beginner-intermediate programming in a language called RobotBASIC. RB can be used to teach the above mentioned skills in an easy and fun way. Students can accomplish interesting and exciting tasks early on while they are learning the skills, instead of boring, irrelevant and contrived examples. Students can immediately and with minimal effort start writing useful programs. They do not have to be concerned with advanced principles that, as important as they maybe for the long term, would only serve to confuse and dissuade them from enduring for the long term where the advanced topics become important.

Take for example the topic of variable scoping. Students would not appreciate what the term signifies until they have actually written projects where the difference between global and local variables would impact their code. When students encounter such situations in practice the principle of variable scoping takes on a ***relevant significance*** with a concrete example that they can relate to instead of trying to learn it in a lecture about the topic with contrived examples.

RobotBASIC can also accommodate the student all the way to the advance-intermediate level. All the skills the students acquire in RB, are readily exportable to C++ or Java with some syntax change. This allows the student to move on to more advance topics such as pointers, classes, and OOP with an appreciation and insight for ***how*** and ***why*** these topics are important for an advanced programmer.

Students who have written programs and have already accomplished entire projects would appreciate the encapsulation and code reusability advantages that OOP provides. However students who do not know a subroutine from an if-statement would be extremely confused by an instructor trying to explain the principles of local variables, header files, include libraries, classes, and methods in order to write a “hello world” program.

Using RB, a teacher can introduce students to commands like ***Print, Input*** and ***WaitKey*** to familiarize them with the concepts of user interfacing, but without overloading them with advanced concepts. This way the student is able to

quickly program meaningful projects, but without having to go through hours of confusing instruction on how to setup ***Components*** and how to write their ***Methods*** and customize their ***Properties*** and so on.

As students progress to an intermediate level the teacher can start introducing more advanced concepts as ***xyString, StrInput(), GetKey*** and ***ReadMouse***. Most of these commands and functions provide lots of power in a very convenient programming syntax. So the intermediate programmer can create more powerful programs without too much increase in syntax complexity.

More advanced students may be introduced to things like ***ErrMsg(), MsgBox(), push buttons, edit boxes, JoyStick, Sound, PlayWav, Bitmap commands*** and so forth. Also, RB can be effectively used to introduce the student to GUI and Event Driven programming concepts, yet keeping the jump in complexity at a surmountable level.

Since the syntax of RB can be similar to C++ in many ways, an RB trained programmer can easily move to C++, but with the added advantage of being an already proficient ***Algorithms*** developer, and concentrate on the fine details of the syntax of C++ programming with an appreciation for how they can provide additional capabilities.

For engineering students, the many facilities (e.g. hardware I/O and Matrix Algebra) provided in RB can be extremely hard to achieve with any C++ compiler, even at an advanced level. This means that an engineering student using RB can concentrate on the details of the project at hand ***not*** how to program C++ to develop routines just so as to be able to start programming the ***target*** project.

During a recent upgrade to RobotBASIC a new function called Spell(ExprN) was added. This function spells out a given number in words. The algorithm is not a trivial one and it had to be prototyped. RB was used to do the prototyping work due to its ease of use and nice debugging tools as well as convenient development cycle.

Once the algorithm was worked out using RB, it was converted to C++ code so as to add it to the RB language. The process of converting the RB code to C++ was utterly painless and took only a few minutes.

The code presented below demonstrates three points.

1- By looking at the C++ code and the RB code you will notice the points of syntax resemblance.

2- If you concentrate on the Algorithmic aspect you will notice that it does not matter what language you use to develop the logic of the algorithm. The details of the logic are the same regardless of the syntax of the language.

3- When using RB, there are numerous functions and commands that are available for use ***immeditatly*** and a beginner can use the facilities they provide without having to develop them. However, using a language like C++ students, have to develop their own code, unless there happens to be an available library. If the objective is to develop the algorithms themselves then the inbuilt functions or commands can provide a yardstick for checking the actions of the developed algorithms.

The following code is an RB subroutine called SpellNumber and a C++ function called SpellNumber(double num). Notice that the C++ function uses other functions that are not part of the C++ language and are not listed (they are actually part of the RobotBASIC interpreter). Therefore in reality developing the code in C++ is even harder due to the need for developing these support functions.

I hope the above discussions has convinced you that to teach your students programming using RobotBASIC is a worthwhile alternative and that the time and effort spent in learning RB is a valuable step in the development of students during their journey towards being effective and versed Software Engineers.

```
//RobotBASIC subroutine
SpellNumber: //--input:num   ----output:sn
  if !vType(FirstTime)
    data ones_d;"Zero ","One ","Two ","Three ","Four ","Five ","Six ","Seven "
    data ones_d;"Eight ","Nine ","Ten ","Eleven ","Twelve ","Thirteen "
    data ones_d;"Fourteen ", "Fifteen ","Sixteen ","Seventeen ", "Eighteen "
    data ones_d;"Nineteen ","Twenty "
    data tens_d;"Twenty ","Thirty ","Fourty ","Fifty ","Sixty ","Seventy "
    data tens_d;"Eighty ","Ninety ","hundred "
    data thnds_d;"","Thousand ","Million ","Billion ","Trillion "
    FirstTime = false
  endif

  sn = "" \  n=0 \ tnum = num
  if (tnum > 999999999999) then sn = "Overflow" \ return
  if (tnum < 0) then tnum = -tnum
  while (true)
     hunds = Round(Frac(tnum*1.0/1000)*1000)
     rest = tnum/1000.0 - Frac(tnum/1000.0)
     //====== process hundreds
     tens = hunds#100
     if tens<=20
        hs = ones_d[tens]
     else
        ones = tens#10
        tens = tens/10-2
        hs = tens_d[tens]
        if ones > 0 then hs = hs+ones_d[ones]
     endif
     hunds = hunds/100
     ss = ""
     if hunds > 0 then ss = ones_d[hunds]+"hundred "
     if ss != "" and hs=="zero"
        hs = ss
     else
        hs = ss+hs
     endif
     //===augment result to running string
     if hs == "zero" && rest == 0 && sn==""
        sn = hs
        break
     elseif hs == "zero" && rest == 0 && sn != ""
        break
     elseif hs == "zero" && rest != 0
        //nothing
     else
        sn = hs+thnds_d[n]+sn
     endif
     if rest == 0 then break
     tnum = rest
     n = n+1
  wend
  sn = Trim(sn)
Return
```

```
//C++ function
AnsiString Expression_Class::SpellNumber(double num)
{
  static AnsiString ones_d[]=
    {"Zero ","One ","Two ","Three ","Four ","Five ","Six ","Seven ",
     "Eight ","Nine ","Ten ","Eleven ","Twelve ","Thirteen ",
     "Fourteen ", "Fifteen ","Sixteen ","Seventeen ", "Eighteen ",
     "Nineteen ","Twenty "};
  static AnsiString tens_d[]=
    {"Twenty ","Thirty ","Tourty ","Fifty ","Sixty ","Seventy ",
    "Eighty ","Ninety ","Hundred "};
  static AnsiString thnds_d[]=
    {"","Thousand ","Million ","Billion ","Trillion "};

  AnsiString sn = "", hs, ss;
  int n=0;
  __int64 tnum = num;
  if (tnum > 999999999999999) return "Overflow";
  if (tnum < 0) tnum = -tnum;
  while(true)
  {
     int hunds = Round(Frac(double(tnum)/1000)*1000);
     __int64 rest = tnum/1000;
     //====== process hundreds
     int tens = hunds %100;
     if (tens<=20)
        hs = ones_d[tens];
     else
     {
        int ones = tens % 10;
        tens = tens/10-2;
        if (tens < 0) tens = 0;
        hs = tens_d[tens];
        if (ones > 0) hs = hs+ones_d[ones];
     }
     hunds = hunds/100;
     ss = "";
     if (hunds > 0) ss = ones_d[hunds]+"Hundred ";
     if (ss != "" && hs=="zero")
        hs = ss;
     else
        hs = ss+hs;
     //===augment result to running string
     if (hs == "zero" && rest == 0 && sn=="")
     {
        sn = hs;
        break;
     }
     else if (hs == "zero" && rest == 0 && sn != "")
     {
        break;
     }
     else if (hs == "zero" && rest != 0)
        ;//nothing
     else
        sn = hs+thnds_d[n]+sn;
     if (rest == 0)  break;
     tnum = rest;
     n = n+1;
  }
  return sn.Trim();
}
```