Del Siegle, Ph.D.

Developing Student Programming and Problem-Solving Skills With Visual Basic

Three decades have passed since computers were first introduced into American classrooms. Early technophiles will recognize names such as the Commodore PET 2001, Tandy's Radio Shack TRS-80, and the Apple II. During the genesis of computers in the classroom, educators were unsure how to teach students with these strange machines. Farsighted educators knew microcomputers represented a significant technological advancement that would revolutionize how people worked, but they were unsure what role computers would play in everyday classrooms. Would computers be used as a teaching and learning tool, or was their importance their future role in the workplace?

Mathematics teachers were among the first to embrace this new technology. Frustrated with the paucity of educational software that was available, they began teaching students to write computer programs. Many of the early computer manuals included tutorials on writing computer code with the BASIC (Beginner's All-purpose Symbolic Instruction Code) programming language. Educators taught programming for two reasons. First, they believed that computer users would also need to be computer programmers. The "new math" movement of the 1960s centered on this belief and included lessons on binary numbers (Base 2), which were designed to help students better understand computer programming. The second reason for teaching programming was that programming promoted problem solving, a topic dear to mathematics (and gifted program) educators' hearts. Today, the idea that one would need to write a computer program in order to use the computer seems ludicrous. However, the importance of problem solving has not diminished.

Three decades ago, COBOL, FORTRAN, and BASIC were the computer languages of the day with PASCAL debuting slightly later. Seymour Papert (1980) at MIT believed that students at a very young age should be solving problems and programming computers, rather than computers programming students. He developed the LOGO language to introduce very young children to programming. LOGO provided premathematical children with problem-solving and reasoning experiences as they created shapes with the now familiar "turtle." All of this occurred before the advent of Java, C++, and other computer languages that are popular today.

Although most computer users will never need to write a computer program, many students enjoy the challenge of creating one. Computer programming enhances students' problem solving by forcing students to break a problem into its component pieces and reassemble it in a generic format that can be understood by a nonsentient entity. It promotes planning and organization skills, and it requires precision and self-discipline. Computer programming is one of the three proposed types of technological giftedness (O'Brien & Friedman-Nimz, 2006; Siegle, 2008). In addition to programmers, interfacers (those who excel at using software) and fixers (those who enjoy working with technology equipment) often are classified as technologically gifted.

For several years, students have been able to program in Visual Basic through the macro feature that is built into the Microsoft Office Suite. This has been an awkward, albeit free, way to teach Visual Basic programming to students without having the expense of purchasing the programming software. Today, Microsoft offers Visual Basic 2008 Express Edition as a free download from its Web site at http://www.microsoft.com/express/vb/Default.aspx. Visual Basic allows students to create actual programs that can be installed and run using the Windows operating system. It also has options for Web development.

Computer programming is not for everyone, but those who are attracted to it immensely enjoy it. In the past, I have successfully taught BASIC to students

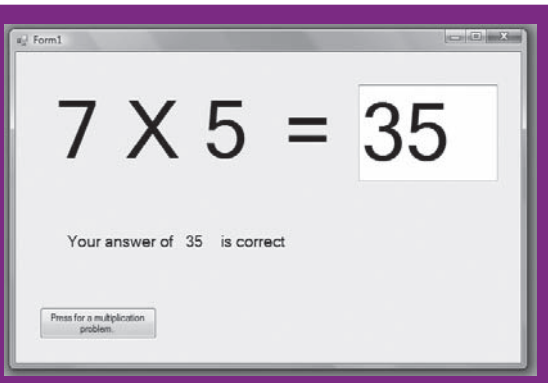

Figure 1. A simple program to test user's multiplication facts.

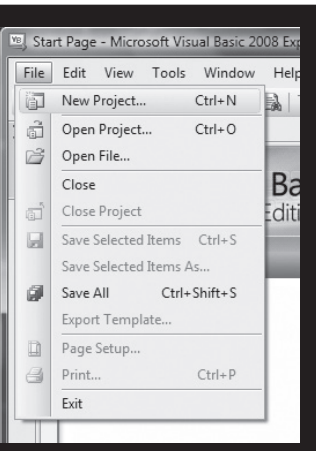

Figure 2. Start a project by selecting New Project from the File menu.

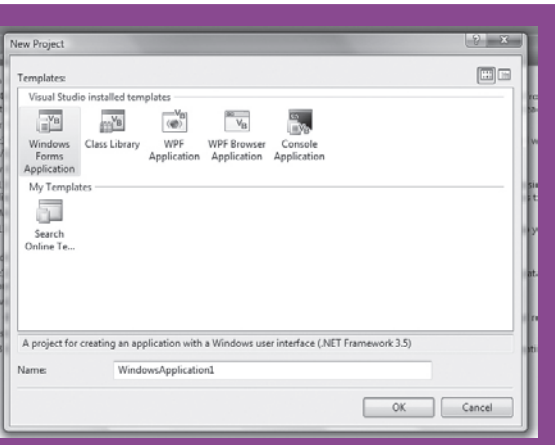

Figure 3. Visual Basic can create a number of different products. For this project, select Windows Forms Application.

as young as fourth grade, some of whom actually went on to careers as computer programmers. The following is a step-by-step description of how to create a simple program using Visual Basic. This activity may be just what a talented student in your classroom is seeking.

Once the Visual Basic authoring software is downloaded and installed, the programming can begin. In this column we will create a simple drill and practice program for multiplication (see Figure 1). Although the Visual Basic options that are featured in this project represent only the tip of the programming "iceberg," the activity shows how easily a simple program can be created and compiled. Those who are familiar with the earlier BASIC language will find the logic and some of the syntax familiar.

The idea behind the Visual Basic system is that the programmer creates screens (**Forms**) that comprise the computer program. A simple program might contain only one form or multiple forms that appear and disappear as the program runs. The programmer places **controls** (from a **toolbox**) that serve different purposes on each form. A control is an object that appears on the computer screen: It might be text to read, a graphic image to view, a pull-down menu to use, or a radio button, checkbox, or command button to click. For example, the **Label** displays text. The **TextBox** allows users to input information by typing. The **Button** control might be programmed to make an object appear or disappear or a mathematical formula to be calculated when the button is clicked. Visual Basic programming is known as *object-oriented programming* because each object can be set to perform a variety of functions. In other words, each object has a life of its own within the program. The simple drill and practice program that we are making can be created with a single Form and three tools: a Label, a TextBox, and a Button control.

Once the program has been installed, the programmer selects **New Project** from the **File** menu (see Figure 2) and **Windows Forms Application** from the dialog box that appears (see Figure 3). A blank form will appear (see Figure 4). The form can be resized by clicking on it and dragging the corners. Each object used in a Visual Basic program has a name. By default, this blank form is called Form1. Each object also has a set of **Properties** that can be changed. The Properties list for an object appears on the right of the screen when the mouse is clicked once on an object. When we click on the Form1 object, its Properties list appears on the right. In the example shown in Figure 5, we have clicked on the **Background** property and a palette of available colors for the form background is shown. In addition to the background color, a number of the form's properties can be changed. For this project, we will not change any of the Form1 properties, although users can easily change the background color as shown in Figure 5.

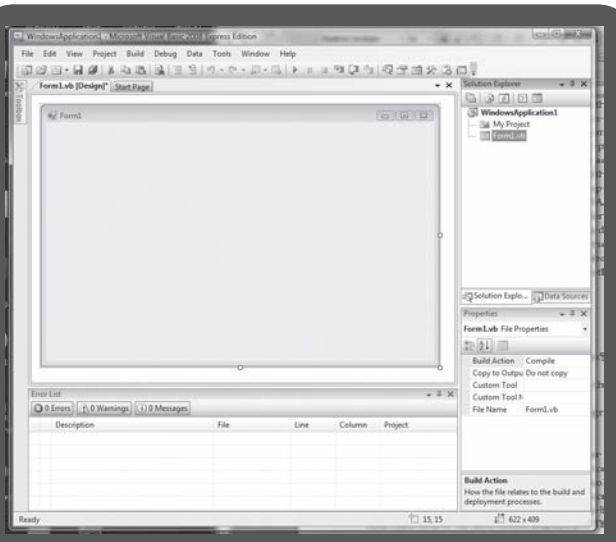

Figure 4. A blank form is displayed in the center. The Properties window appears on the lower right and the Toolbox icon to display tool options appears in the upper left.

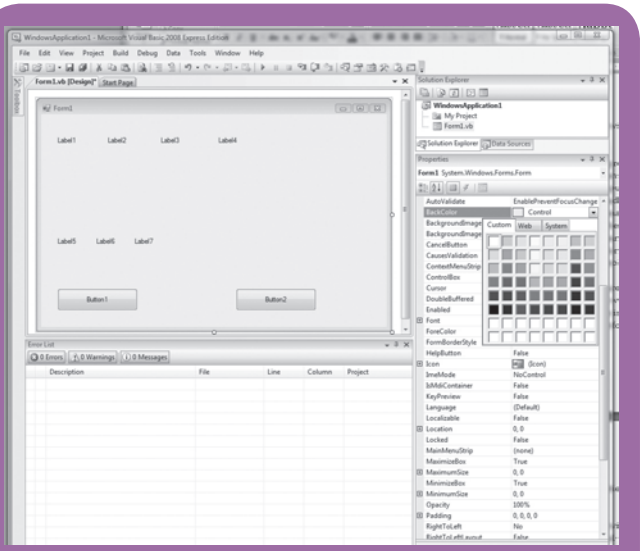

Figure 5. A form's background color can be easily changed by clicking on **BackColor** in the form **Properties** and selecting a **Custom** color.

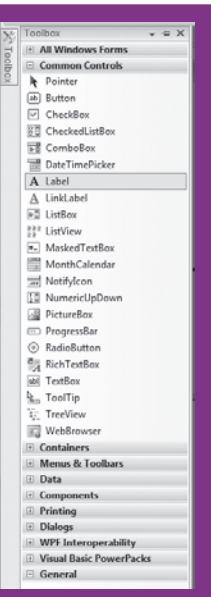

Figure 6. The Toolbox contains different possible Objects on a form and is displayed by clicking on the Toolbox icon located near the top left areas of the screen.

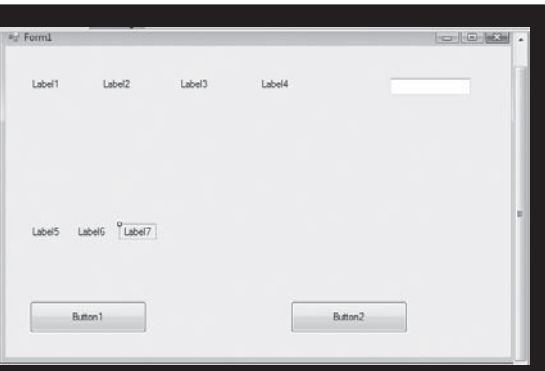

Figure 7. Form1 with seven Labels, one TextBox, and two Buttons.

It is now time to add some **objects** to the form. These objects will display information for the user and will change as the user runs our program. We click on the word **Toolbox** at the top left of the screen to see a list of possible objects (see Figure 6). First, we will select the **Label** command by selecting the word **Label** (seventh on the list of tools). We move the mouse to the form and draw a box where we wish to place the label. For this project, we will use seven labels, so we click and draw seven different label boxes. We also need one **TextBox** (19th on the list of tools) and two **Buttons** (2nd on the list of tools). A form displaying where these objects should be placed is shown in Figure 7.

We have two tasks. First, we simply need to set the starting properties for each object (how the objects will appear when the program opens). Second, we need to write code for the two buttons, which will have our users request multiplication problems and examine their answers. To do this, we will need to set the **Font** properties for Label1, Label2, Label3, and Label4. We click once on Label1; its **Properties** window appears on the right. To change the size of the font, we double click on the word **Font** in the **Properties** window (see Figure 8). The **Font** options will

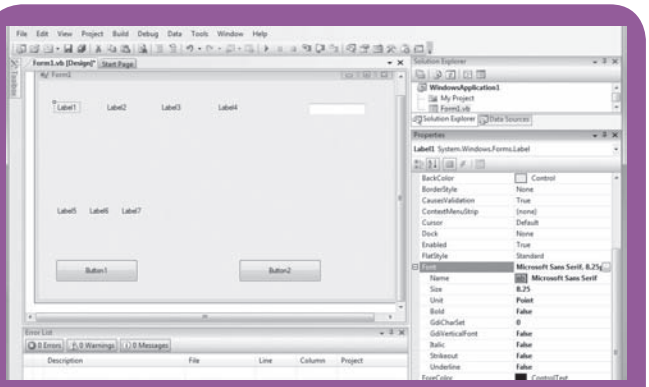

Figure 8. By clicking once on Label1, Label1's Property window is displaced. Double clicking on Font within the Property window displays the options for changing Label1's font.

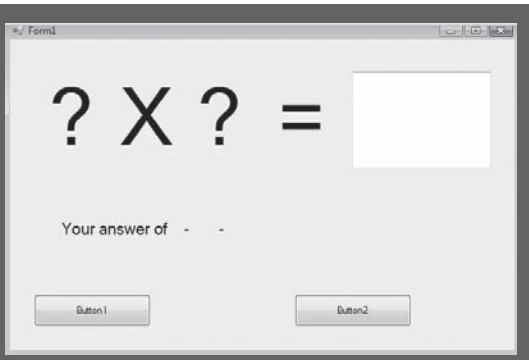

Figure 9. **Font Size** and **Text** changed for Labels1–7 and TextBox1.

show, and we can change the font **Size** to whatever we wish. For our example, we will change the font size from 8.25 to 72. We also want to change the **Text** property for Label1 to a question mark by moving down the properties list to **Text** and replacing the word **Label1** with **?**. Next we click on Label2 and change the **Font Size** property to 72 and the **Text** property to **X**. We will change Label3's font to 72 points and its text to **?**. We also will change the **Font Size** of Label4 to 72 and the **Text** to **=**. Finally, we will click once on TextBox1 and change the **Font Size** in its **Property** window to 72. Our first row of labels should now resemble Figure 9. The labels can be moved to match the spacing shown in Figure 9.

We now need to change the **Text** property of Label5. We click once on Label5 and change the **Text** from **Label5** to **Your answer of**. We do this the same way as we did to change the **Text** property of the Label1–4. We also will change the **Text** of Label6 and Label7 to hyphens by simply entering a **-** in the **Text** property for those two labels. In

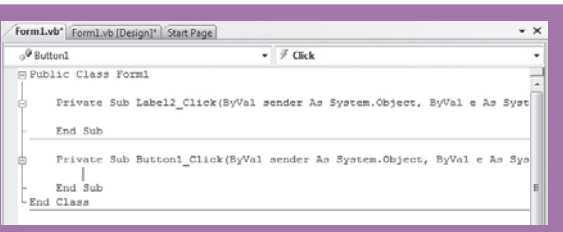

Figure 10. The **Code Editor** after double clicking on Button1 and before entering programming code. By default, the code is set to react to a mouse click. The action can be changed by selecting other mouse options under the **Click** menu at the top right.

the sample shown in Figure 9, we see that the font size for Label5–7 has been changed to 14 and the label displays the change we made in the **Text** property. Although we have changed the **Text** property of these objects, they still retain their names (e.g., Label1).

Once the initial property values of our objects have been set, we are ready to write code for Button1 that will present a multiplication problem and code for Button2 that will check the user's answer to the problem. To program (write code for) an object (such as Button1), we simply double click on Button1 on our form. If we had accidentally double clicked (instead of single clicked) when we were setting the **Properties**, we could have left the **Code Editor** and returned to the **Form** view by closing the **Code Editor** (click on the X on the top right of the **Code Editor**). For now, we want to be in the **Code Editor**. We want to instruct our program to randomly create numbers from 0 to 9 and display them as **Text** for Label1 and Label3. To do this, we program Button1 to change the **Text** of Label1 and Label2 to a random number from 0 to 9. The code to make this happen is:

```
Label1.Text = Int(Rnd() * 10)
Label3.Text = Int(Rnd() * 10)
```

We type the above code in the **Code Editor** at the location of the cursor (see Figure 10).

Computer random number generators produce numbers between 0 and 1 that contain several digits. Because we wish to have our multiplication drill and practice present numbers from 0 to 9, we need to devise a way to have the random number created by the computer with Rnd() changed to a larger whole number. We do this by multiplying the random number by 10. This converts a random number such as .453960 to 4.53960. Because we only want whole numbers, we use the Int command (which stands for Integer) to take the integer (whole number) value of that number. In other

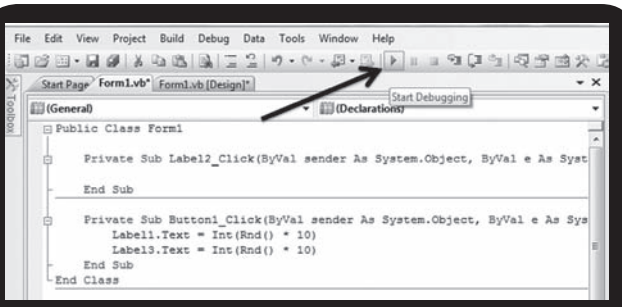

Figure 11. Code can be tested by clicking on the green arrow near the top of the screen.

words, the command will drop the digits after the decimal, which leaves us with 4. If we wanted to restrict the numbers to 0 to 5, we would have multiplied by 6 instead of 10. Using this system, the lowest number is 0 and the number we multiply the random number by is the number of different numbers from 0 up that we will produce.

We can check whether our code works by clicking on the green arrow shown in Figure 11. Once the program starts, we click on Button1. Label1 and Label2 should display random numbers (instead of the question mark we set in the **Properties** window). This occurs because our code instructs each label to set its text to the integer value of a random number. To return to writing code, simply close the window in which the sample program is running. This is achieved by clicking on the X in the top right of the sample program window. To return to the form, close the **Code Editor** in a similar fashion.

Now we need to write code for our second button. We will create code to have Label6 display the answer that our user will be entering in TextBox1 and have Label7 display whether that value is the correct product for the multiplication of the two numbers we randomly generated. On the form, we will double click on Button2 and type the following in the **Code Editor**:

```
Label6.Text = TextBox1.Text
If Label6.Text = Label1.Text * Label3.Text Then
Label7.Text = "is correct."
Else
Label7.Text = "is incorrect."
End If
```

The first line has Label6 display the **Text** that the user entered into TextBox1. The next five lines are an IF-THEN-ELSE statement. When the statement following the IF code is correct, the program executes the code that follows THEN. When the statement following the IF code is not correct, the program executes the code that follows the ELSE. In this case, the IF statement checks whether the user's answer (as displayed as **Text** in Label7) equals the value of the **Text** displayed in Label1 multiplied (computers use * to denote multiply and / to denote divide) by the **Text** in Label3. If they are equal, the **Text** of Label7 displays **is correct**. If they are not equal, the **Text** of Label7 displays **is incorrect**.

We can test our program by clicking on the small green arrow near the top of the screen. Once our program starts, we press Button1 for a multiplication problem. We enter an answer in TextBox1 and press Button2 to judge whether the answer is correct. We should have a functioning program (provided we did not type something incorrectly).

Assuming everything went well up to this point (we typed everything correctly in the right places), we need to modify our program to be more user-friendly. We stop our sample program and return to the form and **Code Editor**. First, we should change the **Text** property of Button1 to **Press for a multiplication problem** so our user knows to click on it to receive his or her multiplication problem. To do this, we click once on Button1 to display the **Property** window (if we are in the **Code Editor**, we will close it to return to our form). We scroll down to the **Text** property and enter **Press for a multiplication problem**. Second, we should change the **Text** of **Button2** to be **Enter your answer in the box above and press here to check it**. We make this change by clicking on Button2 to display the **Properties** window and entering the above message in the **Text** property. If the buttons are too small to display the text, they can be resized larger by clicking on them and expanding the corners. At this point, Form1 should resemble Figure 12.

By now, we have demonstrated that the properties of an object can be set for the start of a program, but can change as the program runs. For example, we set the **Text** of Label7 to be a hyphen and then changed it to display whether the answer was correct or incorrect once our user pressed Button2. As our program now runs, there are a couple of problems. Our user can see Button2 (Check answer) before he or she presses Button1 (Press for problem). We should make Button2 invisible until the problem is presented and we should make Button1 invisible after the problem is presented so the user cannot receive another problem until he or she has correctly answered the current problem. These are simple additions to our program.

Click on Button2 (Check answer) and change the **Visible** property to **False**. This causes it not to appear when the program starts.

Double click on Button1 (Present Problem) and add the following lines of code: **Button1.Visible = False** (this stops the user from requesting a new problem before answering the current one by making the button invisible) and

Button2.Visible = True (this makes Button2 visible so the user can check his or her answer). Close the **Code Editor**.

Double click on Button2 and add the following lines of code just above **Else**: **Button1.Visible = True** (when the answer is correct this will make the Press for problem button visible so a new problem can be requested) and **Button2. Visible = False** (this will make the Check answer option invisible until the next problem is presented).

Type the following after the **Else**: **Button1.Visible = False** (when the answer is wrong, this will keep the user from asking for another problem without getting this one correct). Close the **Code Editor**.

The final code should be as follows:

```
Public Class Form1
Private Sub Label2_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Label2.Click
End Sub

Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
Button1.Visible = False
Button2.Visible = True
Label1.Text = Int(Rnd() * 10)
Label3.Text = Int(Rnd() * 10)
End Sub

Private Sub Button2_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button2.Click
Label6.Text = TextBox1.Text
If Label6.Text = Label1.Text * Label3.Text Then
Label7.Text = "is correct"
Button1.Visible = True
Button2.Visible = False
Else
Button1.Visible = False
Label7.Text = "is incorrect"
End If
End Sub
End Class
```

This simple program can be modified further to keep track of how many problems were answered correctly the first time or to display attractive graphics when the correct answer is given. Although a course in programming is beyond the scope of this column, Microsoft offers free tutorials and videos at http://msdn.microsoft.com/en-us/vbasic/default.aspx. The instructional videos are well worth viewing and range from beginner topics to more advanced topics.

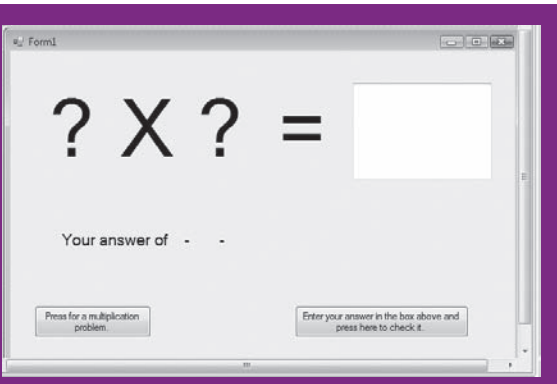

Figure 12. Final view of completed Form1.

As with any technology project, the work should be saved frequently. In the case of Visual Basic, the entire project needs to be saved. This is accomplished with **Save All** from the **File** menu. Once the program is finished and running correctly, it needs to be exported as a program. Select **Publish** from the **Build** menu to achieve this. There will be options for where to publish it. Because this is a Windows program that needs to be installed, the programmer has the option to place it on the Web for download and installation or on a disk for disk installation.

As stated earlier, writing computer code is not for everyone, but those who enjoy it often find it addictive. Time stands still and hours can pass for someone who enjoys the process—whereas for others it can be frustrating because a single incorrect character can keep a program from running correctly. With patience, these instructions should provide an introduction to the exciting world of computer programming.

The skills necessary for writing computer code closely match common characteristics associated with gifted students. Computer programming can be a valuable tool in gifted and talented educators' arsenal of learning activities. GCT

References

O'Brien, B., & Friedman-Nimz, R. (2006, November). *Timing is everything: The emergence of technology talent.* Paper presented at the annual meeting of the National Association for Gifted Children, Charlotte, NC.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.

Siegle, D. (2008). Identifying and developing technological giftedness: Exploring another way to be gifted in the 21st century. In M. W. Gosfield (Ed.), *Expert approaches to support gifted learners: Professional perspectives, best practices, and positive solutions* (pp. 141–150). Minneapolis, MN: Free Spirit.