A Rigorous Course Sequence Based on VB.NET

Qi Yang
Department of Computer Science and Software Engineering
University of Wisconsin - Platteville
Email: YangQ@uwplatt.edu

Abstract

We offer three courses on VB.NET for our computer science students. The first course focuses on GUI controls and event procedures. The second course covers the OO features of VB.NET and other advanced topics. The last course uses VB.NET with ASP.NET to create Web applications and Web services. VB.NET is also used in our Database Design and Implementation course.

In all the three VB.NET courses, hands-on testing is used for the tests. Although it's open computer and open book, the students must program by themselves to create running programs.

Programming in Visual Basic

Our first course on VB.NET is CS2340 – Programming in Visual Basic. The course requires CS I and we use C++ in our CS I course. Although the students understand the basic programming concepts, programming GUI controls, especially the event procedures are new to the students. The students also learn how to connect to MS Access databases at design time.

The concept of Object Oriented Programming is not covered in the course, but the students gain significant experience in OO programming, since VB forms and all controls are classes. This should help the students to take our CS II course, which covers the concept of Object Oriented Programming. One example is multi-form applications. The students are required to use code module with main sub and create an object for each form class.

One difficult issue for the students is to write the form resize event procedure to place controls on a form at run time. It requires some mathematical skill, and some students have trouble to apply what they learn in mathematics to GUI programming. Many students have some difficulties to figure out exactly where to controls at the beginning, but most of them make progress and can do a reasonable job on resize at the end of the course.

Windows Programming

The second course on VB.NET we offer is CS3340 – Windows Programming. The course requires CS II and the students should understand the Object Oriented Programming concept. The course assignments cover the OO features of VB.NET, especially inheritance and polymorphism. They are also required to connect to MS Access databases at run time, using connection, command and adapter classes. Other VB features covered in the course are user controls, threads, and Windows server and client programming using Remoting.

Most course assignments require the students to create solution with multiple projects. One project is normally a class library project, containing the classes. Another project is a GUI project as the user interface and accesses the classes defined in the class library project. Some assignments have a third project, also a class library project, but it has a form and becomes a class library by Visual Inheritance.

Threads programming is the most interesting and also most difficult part for the course. The same .NET thread classes are accessed in VB.NET as in C# and other .NET programming languages. Some interesting assignments on threads are Mutual Exclusion, Barber Shop and Reader and Writer with and without the FIFO rule.

Threads in VB.NET

Threads can be created easily in VB.NET. Assume that System.Threading is imported. The following statement declares a reference to a thread:

```
Dim p1 As Thread
```

In order for the thread to do something, a Sub should be defined. Assume Sub ProcessOne has been defined, the following two statements create a thread object and start the thread:

```
p1 = New Thread(AddressOf ProcessOne)
p1.Start()
```

One assignment on threads is Mutual Exclusion with two threads. Both threads generate a value and use the value to update a global variable Total. The following variables are used to implement mutual exclusion and control the threads:

```
Dim theSem As New Mutex()
Dim p1_paused, p1_cancelled As Boolean
Dim p1_Wait As New AutoResetEvent(True)
Dim p2_paused, p2_cancelled As Boolean
Dim p2_Wait As New AutoResetEvent(True)
```

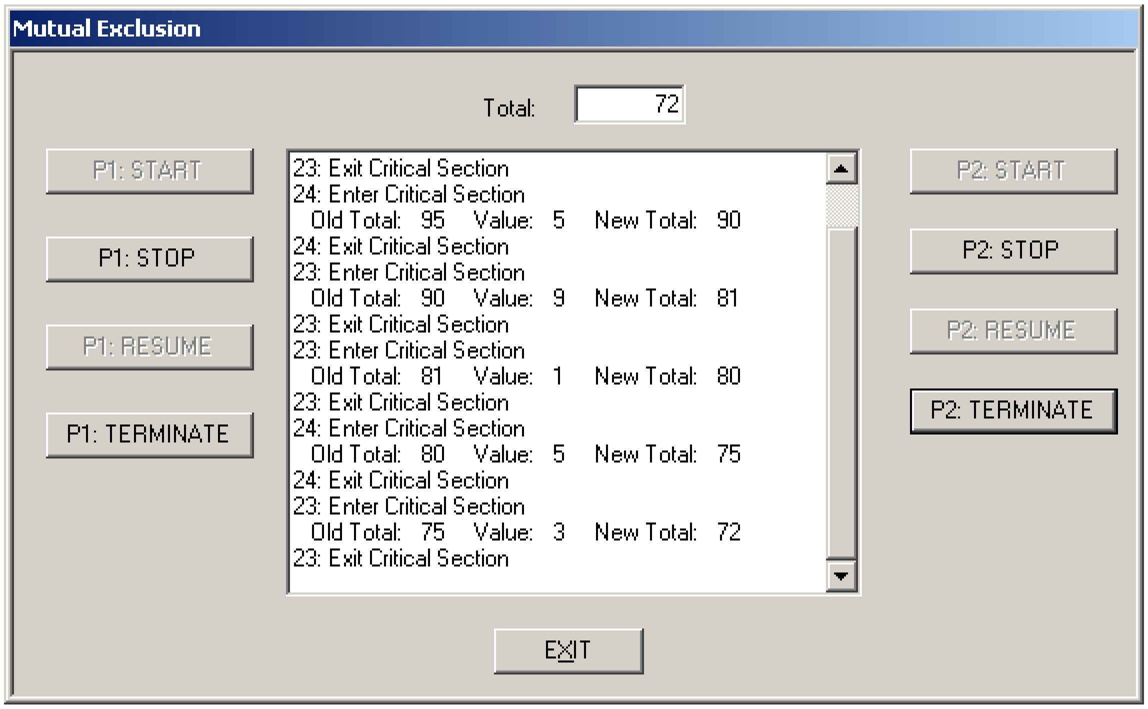
The following is the pseudo code for both threads (i = 1, 2). Method WaitOne() of class Mutex (object theSem) is equivalent to the P operation, and ReleaseMutex() is equivalent to the V operation of a semaphore. Method WaitOne() of object pi_Wait (class AutoResetEvent) blocks the current thread until method Set() is called.

```
While Not pi_cancelled
   If pi_paused
      Wait (pi_Wait.WaitOne())

   If pi_cancelled
      Exit While (terminate the thread)

   Generate a Value
   Set Semaphore (theSem.WaitOne())
   Update Total
   Release Semaphore (theSem.ReleaseMutex())
```

The GUI interface of the program is shown in Figure 1. The event procedures of the buttons are straightforward by using the methods of the thread object and variables defined above:

START	: create a thread object and call the Start method of the object
TERMINATE	: set pi_cancelled to True and the thread will be terminated later

STOP	: set pi_paused to True
RESUME	: set pi_paused to False and call pi_Wait.Set() to wakeup the thread



```
23: Exit Critical Section
24: Enter Critical Section
  Old Total:  95   Value:  5   New Total:  90
24: Exit Critical Section
23: Enter Critical Section
  Old Total:  90   Value:  9   New Total:  81
23: Exit Critical Section
23: Enter Critical Section
  Old Total:  81   Value:  1   New Total:  80
23: Exit Critical Section
24: Enter Critical Section
  Old Total:  80   Value:  5   New Total:  75
24: Exit Critical Section
23: Enter Critical Section
  Old Total:  75   Value:  3   New Total:  72
23: Exit Critical Section
```

Figure 1: Mutual Exclusion

Reader-Writer without FIFO

The Reader and Writer problem is a traditional problem for threads and can be solved with or without the FIFO rule. Two classes are created: Reader and Writer. Each class has a thread object as a private member, some properties to set parameters for the thread, one private Sub that will be executed by an object of the class, and some public methods to start and terminate the thread. A public variable of class ReaderWriterLock is defined in a module and is accessible from both classes:

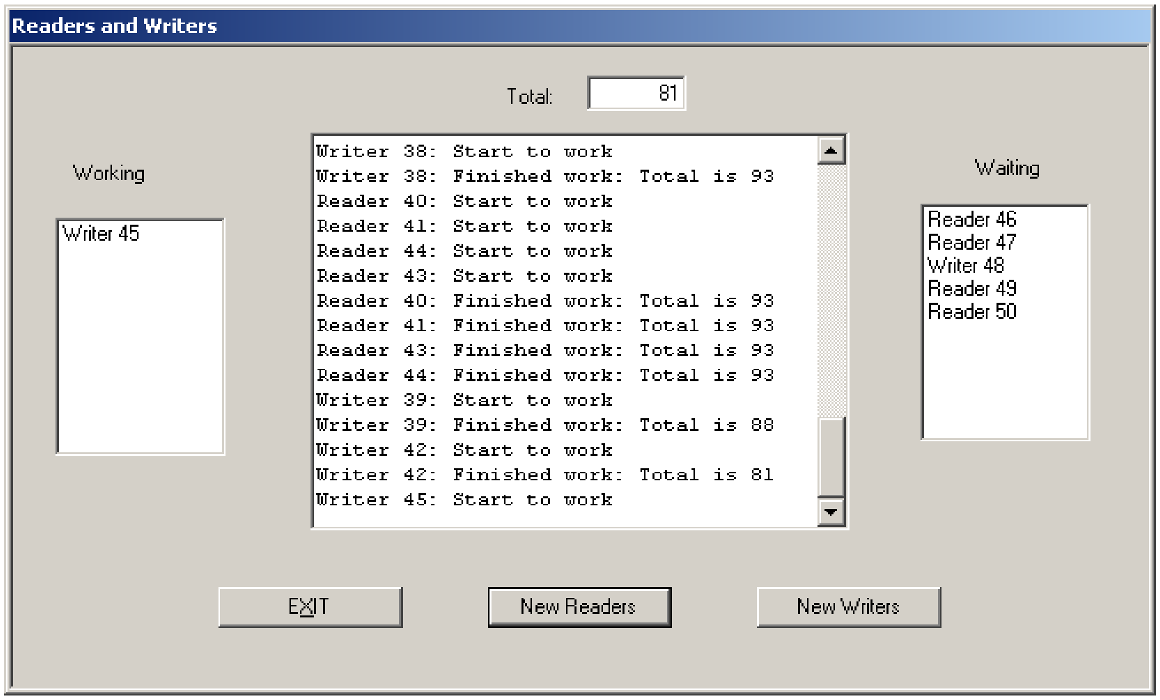
```
Public theLock As New ReaderWriterLock()
```

The pseudo code for Reader and Writer processes is

```
Reader
  Ask for read permission (theLock.AcquireReaderLock())
  Read and do work
```

```
    Release the lock (theLock.ReleaseReaderLock())

  Writer
    Ask for write permission (theLock.AcquireWriterLock())
    Generate value and update Total
    Release the lock (theLock.ReleaseWriterLock())
```

The GUI interface of the program is shown in Figure 2 and 3. Clicking on New Readers or New Writers will generate a new Reader object or Writer object, and the thread of the new object will be started. We can see that writer 48 arrives before readers 49 and 50 but after readers 46 and 47 while Writer 45 is working.



Working:

```
Writer 45
```

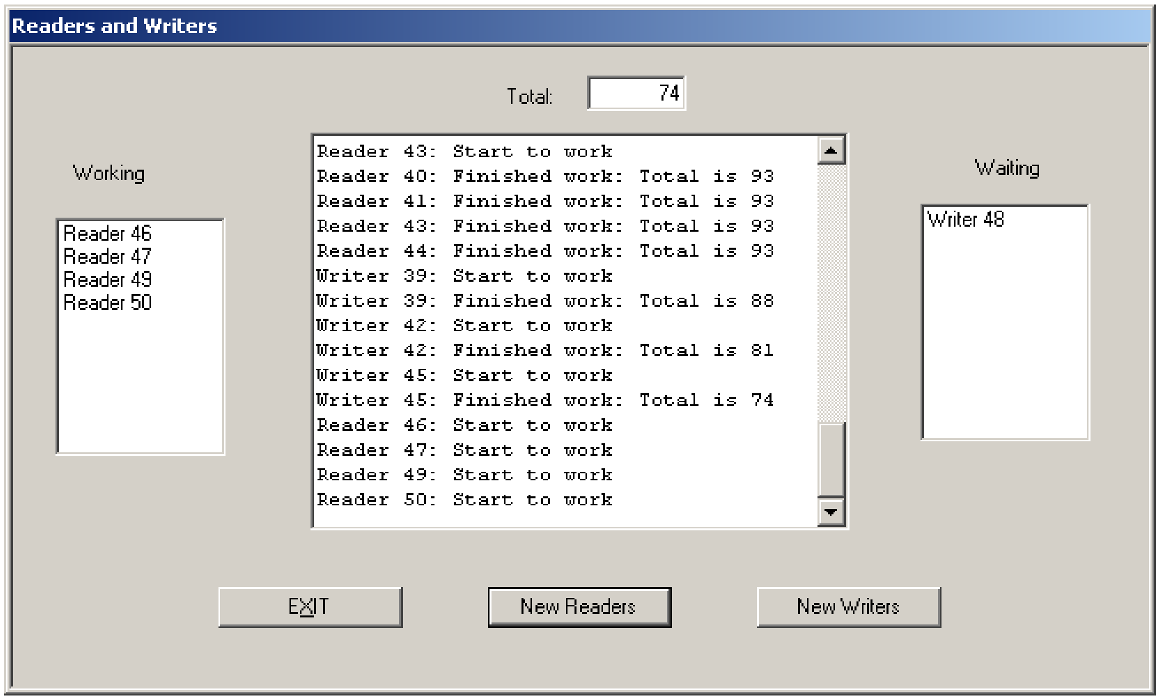
```
Writer 38: Start to work
Writer 38: Finished work: Total is 93
Reader 40: Start to work
Reader 41: Start to work
Reader 44: Start to work
Reader 43: Start to work
Reader 40: Finished work: Total is 93
Reader 41: Finished work: Total is 93
Reader 43: Finished work: Total is 93
Reader 44: Finished work: Total is 93
Writer 39: Start to work
Writer 39: Finished work: Total is 88
Writer 42: Start to work
Writer 42: Finished work: Total is 81
Writer 45: Start to work
```

Waiting:

```
Reader 46
Reader 47
Writer 48
Reader 49
Reader 50
```

Figure 2: Reader and Writer without FIFO (I)

After Writer 45 is done, Reader 46 and 47 can read data, but Writer 48 has to wait. Because FIFO is not enforced, Reader 49 and 50 can also read data while Writer 48 is waiting.



Reader 46
Reader 47
Reader 49
Reader 50

```
Reader 43: Start to work
Reader 40: Finished work: Total is 93
Reader 41: Finished work: Total is 93
Reader 43: Finished work: Total is 93
Reader 44: Finished work: Total is 93
Writer 39: Start to work
Writer 39: Finished work: Total is 88
Writer 42: Start to work
Writer 42: Finished work: Total is 81
Writer 45: Start to work
Writer 45: Finished work: Total is 74
Reader 46: Start to work
Reader 47: Start to work
Reader 49: Start to work
Reader 50: Start to work
```

Writer 48

Figure 3: Reader and Writer without FIFO (II)

Reader-Writer with FIFO

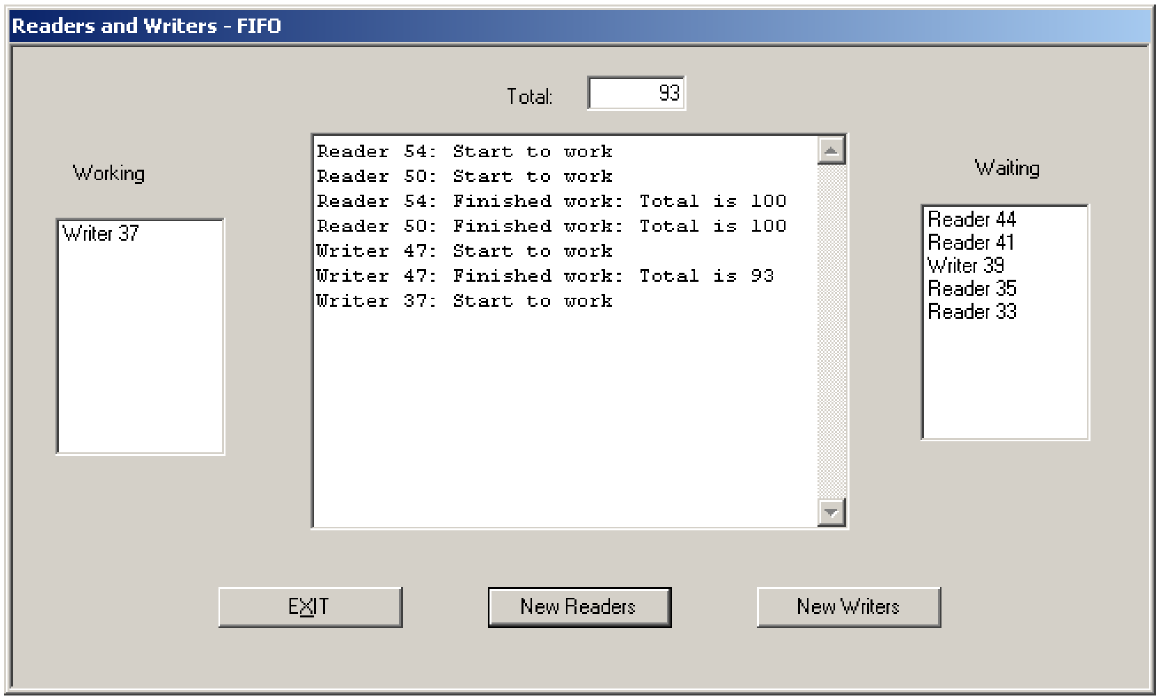
To enforce the FIFO rule, the following variables are added to the module to replace the ReaderWriterLock:

```
    Public MeFIFO As New Mutex()    'Control FIFO Queue
    Public MeData As New Mutex()    'Control Data access
    Public MeRC As New Mutex()      'Control Reader Count
    Public RC As Integer            'Reader Count
```

The Reader and Writer process Subs are modified accordingly:

```
   Reader
      MeFIFO.WaitOne()
      MeRC.WaitOne()
      RC += 1
      If RC = 1
         MeData.WaitOne()
      End If
      MeRC.ReleaseMutex()
      MeFIFO.ReleaseMutex()

      Read data
      MeRC.WaitOne()
      RC -= 1
      If RC = 0
         MeData.ReleaseMutex()
      End If
      MeRC.ReleaseMutex()

   Writer
      MeFIFO.WaitOne()
      MeData.WaitOne()
      MeFIFO.ReleaseMutex()
      Generate a Value and update Total
      MeData.ReleaseMutex()
```

The GUI form is shown in Figure 4 and 5. While Writer 37 is working, Writer 39 arrives after Reader 44 and 41, but before Reader 35 and 33.



Total: 93

Working:

Writer 37

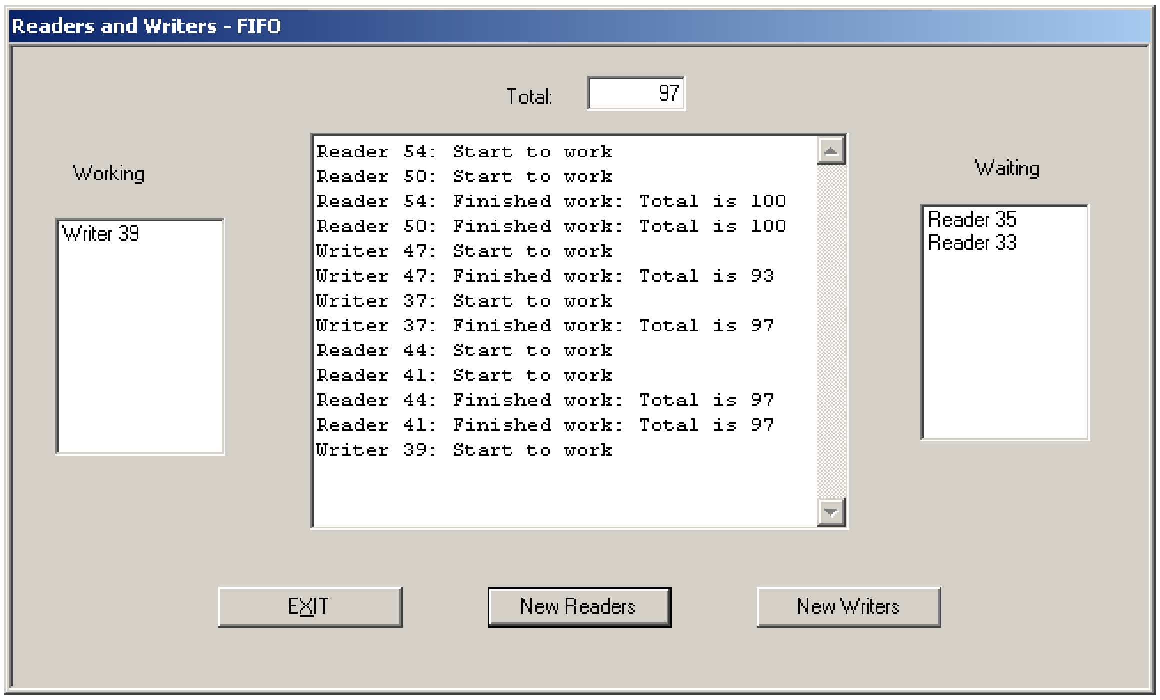
```
Reader 54: Start to work
Reader 50: Start to work
Reader 54: Finished work: Total is 100
Reader 50: Finished work: Total is 100
Writer 47: Start to work
Writer 47: Finished work: Total is 93
Writer 37: Start to work
```

Waiting:

Reader 44
Reader 41
Writer 39
Reader 35
Reader 33

Figure 4: Reader and Writer with FIFO (I)

After Writer 37 is done, Reader 44 and 41 start to work, and Writer 39 and Reader 35 and 33 are all waiting. After Reader 44 and 41 are done, Writer 39 start to work, and Reader 35 and 33 are still waiting.



```
Reader 54: Start to work
Reader 50: Start to work
Reader 54: Finished work: Total is 100
Reader 50: Finished work: Total is 100
Writer 47: Start to work
Writer 47: Finished work: Total is 93
Writer 37: Start to work
Writer 37: Finished work: Total is 97
Reader 44: Start to work
Reader 41: Start to work
Reader 44: Finished work: Total is 97
Reader 41: Finished work: Total is 97
Writer 39: Start to work
```

Figure 5: Reader and Writer with FIFO (II)

Web Protocols, Technologies, and Applications

The last course on VB.NET we offer is CS3870 – Web Protocols, Technologies, and Applications. The course requires CS3340 – Windows Programming and CS3630 – Database Design and Implementation.

The students learn the basic concepts of Web applications such as page post back, session and application variables, authentication and authorization. The students are required to create Web applications using ASP.NET to access databases. They are also required to create a Web service using ASP.NET to provide data from a database to other Web applications.

This 3000 level course requires two other 3000 level courses, and the students have the background to develop some significant Web applications. One such project is the Badger Camp Registration System.

Database Design and Implementation

We also use VB.NET for our CS3630 – Database Design and Implementation. One assignment requires the students to create a GUI program to access an Oracle table. Also for the last phase of the course project, the students will create a GUI program to update multiple related Oracle tables. VB.NET is not required, since no VB.NET course is a prerequisite for the database course, but VB.NET is discussed briefly and most students use VB.NET.

Hands-On Testing

One feature in the three VB.NET courses is that all tests are carried out in Lab. The students are required to create a VB program individually during a test and they have open access to all resources they want to use. The students can also get help from the instructor during a test, but that will result in a reasonable penalty. Help is given only when a student has a problem that prohibits him/her from completing the program.

We started to use hands-on testing in Spring 2002 in the second VB course. Before that, written tests were used, and many students complained that they couldn't remember the exact syntax of VB during tests even they were able to do it using computer. We discussed the issue in our department and brought it to the department's Advisory Board meeting. The Advisory Board members are from different computing industry companies and all of them think that it does not do much good to the students to ask them to memorize the exact syntax and support hands-on testing. With the support from industry community, we decided to try hands-on testing to replace written testing.

During a test, each student uses one computer to create a program individually. The students can bring their textbooks and any other books, notes from class, their programs (even graded), and any other materials they want to use. They have the entire .NET help system to use, and also have open Internet access.

The feedback from students is very positive. Most students like the hands-on testing and enjoy the challenge. They try very hard to prepare for the tests, since they know they will fail if not prepared. This eventually forces the students pay close attention in class and do programming assignments very seriously. Some students also try to go beyond what covered in the class. Overall, the students receive better grades than before. We feel that the students are better VB programmers and deserve their better grades.

The students rarely complain about the hands-on tests, although there are a few students who feel too much pressure during such tests. Some of them failed or dropped the course. Although the number of such students is very small, help them is an important issue in the future.

Students Programs

The students have created many interesting and useful projects in the courses, including different games, a Windows program for our Dean's office to manage records for the study abroad program, a Web registration system for Wisconsin Badger Camp, and a resource registration program for a company that can work locally and remotely.

Summary

We offer three courses on VB.NET at UW-Platteville, and also use VB.NET in the database course. All the courses with the hands-on testing help the students become good VB.NET programmers. In the VB programming contest of the 2004 AITP National Conference, our team won the third place; in 2005, our team won the second place. Our goal is to win the first place in the near future.