

Microsoft C# Projects for the Classroom

Written by Alfred C Thompson II

Distributed by Mainfunction.com

Introduction	4
Why we have done these books.....	4
Who are we	4
How to Use This Book.....	4
Future Developments	6
Wingdings Instructor Notes	7
Wingdings Student Project.....	9
Thermometer Program Instructor Notes.....	11
Thermometer Program Student project.....	13
Parking Lot Program Instructor Notes	15
Parking Lot Program Student Project.....	16
Dice Display Instructor Notes.....	18
Dice Display Student Project	20
Guessing Game 1 Instructor Notes	21
Guessing Game 1 Student Project.....	23
Guessing Game 2 Instructor Notes	25
Guessing Game 2 Student Project.....	27
Flip Book Program Instructor Notes	29
Flip Book Program Student Project.....	31
Change Making Program Instructor Notes	33
Change Making Program Student Project.....	35
Simple House Drawing Instructor Notes.....	37
Simple House Drawing Student Project	39
Spaceship Invaders Instructor Notes.....	41
Spaceship Invaders Student Project	44
Tic Tac Toe Instructor Notes.....	46
Tic Tac Toe Student Project	48
Picture Display Program Instructor Notes	50
Picture Display Program Student Project	52
Multiple Choice Grading Program Instructor Notes	54
Multiple Choice Grading Program Student Project.....	56
Multiple Choice Grading Program with Structures Instructor Notes.....	58

Multiple Choice Grading Program with Structures Student Project	60
Letter Counter Program Instructor Notes	62
Letter Counter Program Student Project	64
Hangman Program Instructor Notes.....	66
Hangman Program Student Project.....	68
Drawing Program Instructor Notes	70
Drawing Program Student Project.....	72
Roman Numeral Conversion Instructor Notes.....	74
Roman Numeral Conversion Student Project.....	76
Palindromes Instructor Notes.....	78
Palindromes Student Project	80
Dice Class Project Instructor Notes.....	82
Dice Class Project Student Project.....	84
Conway's Game of Life Instructor Notes.....	86
Conway's Game of Life Student Project.....	89
Checkers Game Program Instructor Notes	91
Checkers Game Program Student Project	94

Introduction

Why we have done these books.

C# (pronounced C Sharp) is a new object oriented language from Microsoft. It is a major part of the Visual Studio .NET development environment. Because this is a new language there are not a many projects developed specifically for this language in the classroom. This book has been written to address this need. Specifically, this book is designed to supplement and enhance existing and developing curriculum at the secondary and post secondary level.

In any programming course, there is a need to projects that both develop necessary skills and hold student interest. The projects in this book are designed to supply additional projects for instructors to use. This is a supplement to and not a replacement for a good textbook.

Who are we

Mainfunction offers news, curriculum, grants and resources for secondary computer science, engineering and information technology educators. It's about using technology in innovative ways to further computer science and information technology instruction. Visit us on the World Wide Web at <http://www.mainfunction.com> or direct to the teacher section at <http://educators.mainfunction.com>.

How to Use This Book

Target Audience

This project book has been written to be used in the context of a first programming course using C#. It assumes no previous programming knowledge on the part of a student. It is designed for instructors to use as a supplement to their primary instructional resources. As such, it assumes that the student has available to them an instructor and a textbook for use as reference.

Projects

All projects have two sections. The first section for teacher use and the second for student use. The instructor section includes the following sections:

- ABILITY LEVEL – Required ability level for students attempting project.
- APPROXIMATE COMPLETION TIME – An estimate of how long students will require completing the project.
- OBJECTIVES – What skills and information are being reinforced by the project.
- SKILLS NEEDED – A list of prerequisite knowledge and skills for students who undertake the project.

- **MATERIALS NEEDED** – What resources and materials the project requires supporting student work on the project.
- **TEACHING SUGGESTIONS** – Suggestions on ways to introduce the project, common problems encountered by students, and other information related to the project.
- **RESOURCES** – Any additional resources involved in the project.
- **SUGGESTED EVALUATION** – Indications of what to look for and grade in student projects.
- **SUGGESTED SOLUTION** – A narrative programming solution to the project. Coded and commented sample solutions in C# are available by sending email to editor@mainfunction.com.

The student sections may be reproduced and distributed to students. These sections include:

- **ABILITY LEVEL** – Required ability level for students attempting project.
- **APPROXIMATE COMPLETION TIME** – An estimate of how long students will require completing the project.
- **OBJECTIVES** – What skills and information are being reinforced by the project.
- **OVERVIEW OF PROJECT** – A summary of what the project involves.
- **PROJECT INSTRUCTIONS** – A list of general instructions for completing the project.
- **ADDITIONAL RESOURCES** - Any additional resources involved in the project.
- **SUGGESTED SOLUTION** – A suggestion of one possible solution. This usually comes in the form of a screen capture of a completed solution form. Students should be encouraged to develop alternative solutions. Some instructors may choose not to distribute a solution so as not to limit student creativity. Others may wish to insist on a specific form appearance so that students do not waste time on the appearance of the form over the code solution.
- **PROJECT EXTRAS** – A list of optional additions to the project. These suggestions will be used by students desiring to do more than the minimum requirements of the project.

Note: All screen captures were created with Visual Studio .NET running on the Windows XP operating system. Your screen may look different if you are using other operating systems or if your systems display settings are different.

Ability Levels

All projects in this book have a suggested ability level. The levels, beginner, intermediate and advanced are rather broad. The explanations below are intended to help the instructor select the projects that are appropriate for their students.

Beginner

Assumes little or no previous experience with programming or C# before the instructor's introduction of the project. May understand basic concepts but be unsure of implementation details.

Intermediate

Understands basic concepts including objects, events, properties of objects, and form design. Understands assignment statements, loops, and decision statements. Understands simple text file input and output. Understands variable and object arrays.

Advanced

Understands record types and advanced data structures. Understands file input and output. Uses multiple forms, subroutines and functions. Is able to design solutions to complex problems.

Sample Solutions

The sample solutions, available through editor@mainfunction.com, were written from the beginning using Visual Studio .NET and were created to verify the instructors notes and to provide suggested solutions. These solutions are not intended to be perfect or ideal solutions. Rather they are examples of what students might reasonably be expected to produce.

Future Developments

Mainfunction intends to add projects to the resource database on a regular basis. We encourage you to submit your favorite projects. All projects will be credited to the submitter. Please visit <http://educators.mainfunction.com> and register as a teacher to contribute.

Wingdings Instructor Notes

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 20 Minutes

OBJECTIVES:

- Use an assignment statement to modify object properties

SKILLS NEEDED:

- Basic understanding of object properties

MATERIALS NEEDED:

- C#

TEACHING SUGGESTIONS:

The purpose of this project is to give students a chance to see objects respond to events. It relies on the most basic of object properties (font and text) and the event (click) that students are most used to using.

Explain the difference between design time and run time changes to properties. Tell them that, while the user can not directly change the text or other properties of some objects, a program can be programmed to allow the user to indirectly change many of these properties.

Pay careful attention to the explanation of assignment statements. Some students will have trouble grasping that the copy moves from right to left. They are used to thinking from left to right.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

This program should respond to a click of a command button and copy a string from one box to another. Consider giving that as the C level and ask students to add other features for more credit. See the student section for suggestions of additional features.

SUGGESTED SOLUTION:

Have the students create a label and a text box. Have them change the font of the label box to Wingdings. Wingdings are a standard font included with Windows. If this font is not available, use any font that includes something other than normal English characters.

The command click routine will include a simple assignment statement to copy the contents of the text box into the text property of the label box.

Wingdings Student Project

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 20 Minutes

OBJECTIVES:

- Use an assignment statement to modify object properties

OVERVIEW OF PROJECT:

Create a program that shows a user what characters they enter look like in a different font.

PROJECT INSTRUCTIONS:

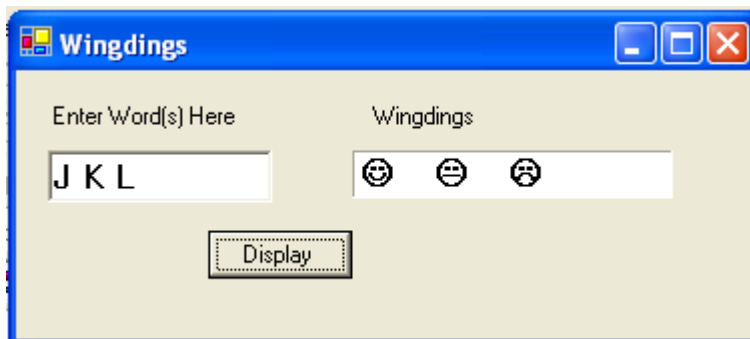
1. On a new form, create a label box, a text box and a command button.
2. Change the font property in the label box to Wingdings, or to a font assigned by your teacher. You may want to increase the size of the font as well.
3. Clear the text property in the label box and the text property in the text box.
4. Change the text property in the command button to something descriptive.
5. Write an assignment statement in the command click routine to copy the contents of the text box into the label box.
6. Test your program.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:

A completed program might look something like this:



PROJECT EXTRAS:

- The `Application.Exit()` method may be used to shutdown a program. Add an exit command button.
- Copying nothing in to a text property empties it. Using a pair of double quotes with no space between them indicates nothing. Use that to create a Clear button that empties both the text and label boxes.
- Create a number of label boxes with different fonts and font sizes. Copy the text string in to all the label boxes.
- Copy the text box into the label box in response to some other event such as the mouse moving over the label box.

Thermometer Program Instructor Notes

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Use mathematical formulas in a program
- Use a scroll bar object

SKILLS NEEDED:

- Understanding of object properties
- Understanding of mathematical operations

MATERIALS NEEDED:

- C#
- Temperature conversion formulas

TEACHING SUGGESTIONS:

Most introductory programming texts use temperature conversion as an early project or example. Asking a student to write a simple input a number, convert the value and display the result program is that it does not really use the power of the computer. This project uses scroll bars as an input device to make the example more visually interesting and to introduce a powerful user interface tool.

The student may set the initial value of the bar either in the property box at design time or in the form load routine with an assignment statement. Unless the student includes program code at form load, the label boxes will not display the results of a conversion until the scroll bar is moved. This can be a useful introduction to concepts of initialization in general.

Scroll bars must have their minimum and maximum values set to work properly with this project. The minimum value is set at the top of the bar. The maximum is at the bottom. This is the opposite of what one sees with a thermometer. C# does not allow a programmer to set the minimum value larger than the maximum value. This means that a user will see lower temperatures at the top of the scrollbar and higher temperatures at the bottom of the scrollbar unless the program changes the value of the scrollbar before it displays the temperature it represents.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

A complete thermometer program should:

- Change the values in the Fahrenheit and centigrade labels with the movement of a scroll bar
- Exit the program cleanly

SUGGESTED SOLUTION:

```
private void vScrollBar1_Scroll(object sender,  
    System.Windows.Forms.ScrollEventArgs e)  
{  
    int degF = 200 - vScrollBar1.Value;  
    double degC = (5.0/9.0)*(degF-32.0);  
    lblF.Text = degF.ToString();  
    lblC.Text = Convert.ToInt16(degC).ToString();  
}
```

Thermometer Program Student project

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Use a scroll bar object

OVERVIEW OF PROJECT:

Create a scroll bar to represent a thermometer. As the slider on the scroll bar is moved display temperature in both Fahrenheit and centigrade in boxes on a form.

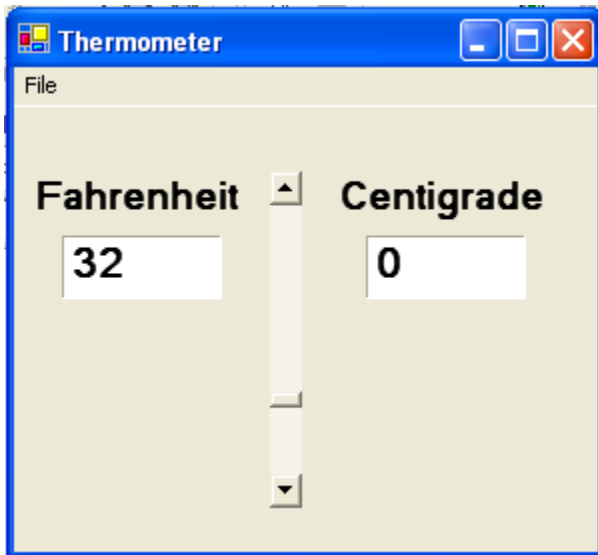
PROJECT INSTRUCTIONS:

1. On a new form, create a scroll bar.
2. Set the minimum and maximum properties for the scroll bar.
3. Create label boxes to display Fahrenheit and centigrade temperatures.
4. Create label boxes to label the temperature boxes.
5. Copy the value of the scroll bar into the Fahrenheit label box in the routine reacting to changes to the scroll bar.
6. Change the value in the centigrade box by using the formula for converting Fahrenheit to centigrade when the Fahrenheit value changes.
7. Create an exit button or menu option with the appropriate code to end the program.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:



The formula for converting Fahrenheit to centigrade is: $C = (F - 32) * (5 / 9)$

PROJECT EXTRAS:

- Add a background to the form. Perhaps a weather related picture.
- Set different background colors for the Fahrenheit and centigrade labels.
- Label the slider bar with degree markings on either side.
- Add buttons to move the slider to “freezing” and/or “boiling.”

Parking Lot Program Instructor Notes

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 90 minutes

OBJECTIVES:

- Use if statements and formulas

SKILLS NEEDED:

- Basic understanding of object properties
- Knowledge of assignment statements

MATERIALS NEEDED:

- C#

TEACHING SUGGESTIONS:

Introduce this project with real life examples of parking lots such as airports or venues of athletic events. Adjust the rates in the description to match actual rates of local parking lots.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

This program should properly calculate parking fees based on the number of hours entered and the rates given to students. Be sure to check times longer than a day and in marginal areas in the formula. The program should format output as currency.

The program should also exit cleanly.

SUGGESTED SOLUTION:

The program should have a number of radio buttons for the user to select the type of vehicle parked. Command buttons can be used to select between standard parking and event rates.

If a standard parking rate is selected, if statements must be used to determine if the truck or car rate is to be used. Additional if statements will handle the boundary cases of less than one hour or hours that total more than the maximum fee for a day.

Parking Lot Program Student Project

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Use a timer
- Understand and use static variables

OVERVIEW OF PROJECT:

Parking garages often have complicated billing procedures. The people who own the garages like to have a computer figure out how much money is owed so that the people working there do not have to waste a lot of time thinking when people are leaving. Your project is to write a computer program that collects some simple pieces of information and tells the cashier how much money to collect.

PROJECT INSTRUCTIONS:

1. Create a form with a text box for entering the amount of time a vehicle was parked. Add option buttons to show if the vehicle is a car or truck. Clearly label each box and button. Add a label or picture box to show the amount to pay.
2. Use the following to calculate how much the customer owes.

<u>Cars</u>	<u>Trucks</u>
\$5 the first hour	\$6.00 the first hour
\$3 for each additional hour	\$3.50 for each additional hour
No more then \$38.00 for 24 hours	No more then \$44.50 for 24 hours

3. Add a button to change an “event rate” of \$19.00.
4. Calculate the fee and display it formatted as money.
5. Add a button or menu option to exit the program.
6. Use good fonts, background colors, labels and captions to make the program as clear to the user as possible.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:

A completed program might look something like this:



The screenshot shows a Windows-style application window titled "Parking Lot Cashier". The window has a menu bar with "File". The main area contains a form with the following elements:

- A label "Hours Parked" above a text input field.
- A radio button labeled "Car" which is selected.
- A radio button labeled "Truck".
- A label "Fee" above a text input field displaying "\$26.00".
- A button labeled "Standard".
- A button labeled "Event Rate".

PROJECT EXTRAS:

- Add bus or van rates.
- Add options for special discounts.
- Rather than asking the cashier to enter the number of hours the vehicle was parked, ask what time the vehicle entered the lot. Use the Time function to find out what the current time is and have the program figure out how long the vehicle was parked.

Dice Display Instructor Notes

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Understand and use string functions
- Understand and use mathematical functions

SKILLS NEEDED:

- Understanding of basic variable types – double, integer and string

MATERIALS NEEDED:

- C#
- Paint Program

TEACHING SUGGESTIONS:

The two critical parts of this project are picking the random number and loading the appropriate picture into each picture box.

The Random data type is used to return random numbers in C#. A random number variable is declared and the Next method is used to return the next random integer in a series. For example:

```
Random rNum = new Random();  
newDie = rNum.Next();
```

The Next method returns an integer that is greater than zero but less than the number specified in the parameter. If no parameter is given then the number will be between 0 and the maximum integer value. To produce the random integers required by this project, students may use this formula:

```
int newDie = rNum.Next(6) + 1;
```

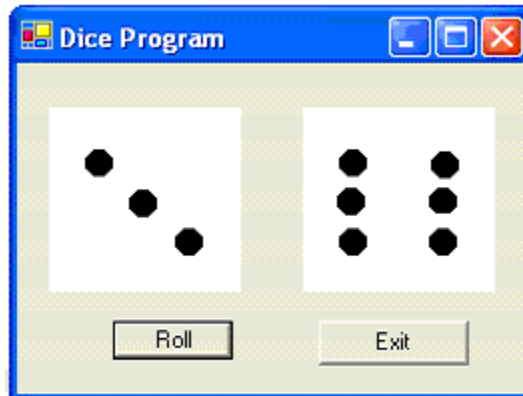
Since the range we need is from 1 to 6 and the number returned is between 0 and 5, the programmer adds 1. Explain to the class that using variables in place of the numeric constants 1 and 6 creates a more general formula.

SUGGESTED SOLUTION:

Six 1 inch by 1 inch die face images should be prepared for students to use. C# objects are measured in pixels by default. There are 96 pixels in an inch. Black and white images will take up the least room and load quickest. Place the images where students can either use them directly or copy them to their own

workspaces. Optionally, students can create their own dice images but they must be careful to create them same size as the picture.

The easiest way to load these images is to create the images with names that are identical except for an identifying number. For example, DIE1.BMP, DIE2.BMP... DIE6.BMP. Use the ToString method and a random number in range to concatenate a file name into a string variable for the Image.FromFile method.



The image above is one possible form solution. The code below will select image files from the folder with the executable file.

```
int newDie = rNum.Next(6) + 1;
string newImage = Application.StartupPath + "\\Die" +
    newDie.ToString() + ".bmp";
picDice1.Image = Image.FromFile(newImage);
newDie = rNum.Next(6) + 1;
newImage = Application.StartupPath + "\\Die" +
    newDie.ToString() + ".bmp";
picDice2.Image = Image.FromFile(newImage);
```

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

A complete dice display program will:

- Display at least two die images
- Die images will change randomly in response to button clicks
- Die images will generally represent different values
- The program will exit in response to a button click or menu option

Dice Display Student Project

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Understand and use string functions
- Understand and use mathematical functions

OVERVIEW OF PROJECT:

Many games depend on the roll of a pair of dice. The object of this program is to simulate the roll of dice. The program will display dice images showing the value of dice values. The project assumes standard six sided dice but can be expanded to other sizes if desired.

PROJECT INSTRUCTIONS:

1. Create a form with two picture boxes and two command buttons.
2. Set the height and width of both picture boxes to 96 pixels by 96 pixels.
3. Set the text property of one button to Exit and write code so that the program terminates when the button is pushed.
4. Set the text property of the second button to "Roll" and write code so that pictures of dice in each picture box

ADDITIONAL RESOURCES:

- Textbook
- Paint program

SUGGESTED SOLUTION:

Use a Random class variable and the Next method with a formula to pick a random number between one and six. Use that number to build the name of an image file and use Image.FromFile method to display that image in a picture box. Do this for each picture box in the form when the display dice button is clicked.

PROJECT EXTRAS:

- Display more than two dice.
- Display the total of the dice in a label box.
- Create and use your own die images.

Guessing Game 1 Instructor Notes

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Understand and use variables with non-local scope

SKILLS NEEDED:

- Understanding of basic events and object properties

MATERIALS NEEDED:

- C#

TEACHING SUGGESTIONS:

Scope of variables is one of the more confusing computer science concepts to be explained in an introductory programming course. The values stored in local variables are lost whenever a subroutine is executed. Variables used by several routines or that must be saved for different calls to the same routine must be created differently.

C# has several options for extending the scope of variables. Declaring a variable at the top of the class definition makes a variable and its value available to all methods in the class. We can also say that the scope of the variable is global to all routines in the class. A name used this way should only be used for one unique variable. If this name is used as a local variable name then the local variable will hide the class level variable.

SUGGESTED SOLUTION:

The guessing game where a player is told that their guess is too high or too low lends itself to simulating a binary search. Using a binary search, any number in the range of 1 to 100 may be determined in no more than seven guesses. Use class level variables to keep track of the highest and lowest possible numbers, as well as the current guess.

Set a guess that is too high as the new top of range. Set a guess that is too low as the new bottom of range. The next guess should be in the middle of the new range of possibilities.

A subroutine for resetting the game can be a useful addition to this project. A static variable in this routine can be used to count the number of games played.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

Test the full range of the program and insure that it will properly guess both one and one hundred. Also, try 49 and 51. The program should properly reach any number.

A good program will not duplicate guesses or guess numbers higher or lower than have already been rejected.

A new game should reset all settings so that counts and ranges are set the same for all games.

Guessing Game 1 Student Project

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Understand and use class level (global) variables

OVERVIEW OF PROJECT:

The object of this project is to write a computer program that will allow the computer to guess a number that you have selected. The computer will make a guess and you, the player, will tell the computer if it guessed too high, too low, or that it guessed the number.

The program will also allow the player to start a new game or to exit the program.

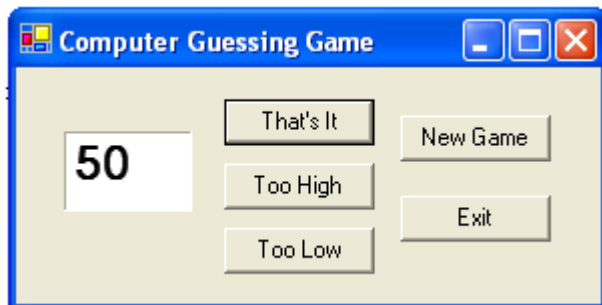
PROJECT INSTRUCTIONS:

1. Create a form with a labeled box to display the computers guess.
2. Create a new game and an exit button.
3. Create objects to indicate if the guess was too high, too low, or right on target.
4. Create program code to respond to the player's indication by selecting and displaying a new computer guess.
5. When the computer guesses the player's number, reset the so that a new game may begin.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:



This solution uses buttons to indicate the success of the computer's guess. The new game button will reset the guess to 50 and set any counters back to zero.

PROJECT EXTRAS:

- Display the number of guesses the computer required to find the players number.
- Declare victory automatically when the last possible guess is made.
- Keep and report counts of how many guesses were too high and too low.

Guessing Game 2 Instructor Notes

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Understand and use **if** blocks
- Understand and use nested **if** blocks
- Understand and use data validation

SKILLS NEEDED:

- Understanding of basic events and object properties
- Familiarity with the Random data type and generating random numbers in a range.
- Optionally: Understanding of exception handling

MATERIALS NEEDED:

- C#

TEACHING SUGGESTIONS:

Nested **if** statements blocks allow a programmer to evaluate a number of possibilities. In this program, there are three possibilities for the program to consider. The number the player guesses may be too high, too low or the number the computer is looking for. The programmer has several options available to evaluate these possibilities. One is to have three totally independent **if** blocks, one for each case.

Three independent **if** blocks are easy to set up but are not as efficient as a program could be. The program must evaluate all three statements even if the first one satisfies the problem. Using nested blocks is more efficient because comparisons are only made until one statement evaluates to true.

If the programmer knows the likely probability of the various options, they may order the **if** checks to have the most probably check made first and the least likely check performed last.

Block IF statements require curly braces to delimit the beginning and ending of the block. Mismatched curly braces are one of the most common errors made with **if** blocks.

Suggest that students use indents to identify the sections of block if statements. This will make it easier to match curly braces. Students often just add braces at the end of a section to silence error messages. This more often results in logic errors than in a correctly working program.

This project is a good one to use when introducing exception handling with try/catch/finally. A good program must validate the guess. A try clause can be opened before the text is converted to an integer to catch conversion problems. Then an if statement can check that the number entered is between 1 and 100. An exception will be thrown if the number is out of range.

If the number is out of range or not an integer (a double or a non number) an exception is thrown and handled by the catch clause. Regardless of an exception or a valid guess the text box should be cleared and returned to focus. This is handled in the finally clause.

SUGGESTED SOLUTION:

Use the **Next** method of the Random data type to select a random number, for the computer. For example:

```
compNumber = myRand.Next(100) + 1
```

The following section of code compares the player's guess (in *myGuess*) to the computer's number (in *compNumber*) with the results reported to the user via message boxes.

```
if (myGuess == compNumber)
{
    MessageBox.Show("You guessed the computer's  
number in " + noGuesses.ToString() +  
" guesses.");
    NewGame();
}
else if (myGuess > compNumber)
{
    MessageBox.Show("Your guess is too high.");
}
else
{
    MessageBox.Show("Your guess is too low.");
}
```

This example assumes a counter variable, called *noGuesses*, which is incremented at each guess.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

The program must correctly identify a numbers relationship to the number “guessed” by the computer. It must not give several conflicting messages for the same guess.

Guessing Game 2 Student Project

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Understand and use **if** blocks
- Understand and use nested **if** blocks
- Understand and use data validation

OVERVIEW OF PROJECT:

Create a program to have the computer pick a random number and allow a player to guess the number. The program will tell the user if their guess is correct or, if incorrect, if the guess is too high or too low.

The program should allow the player to start new games or exit the game completely.

PROJECT INSTRUCTIONS:

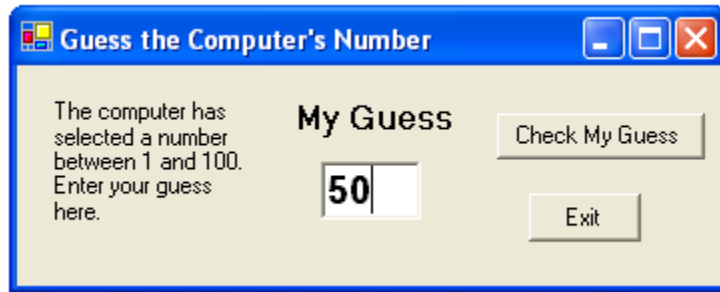
1. Use a method in the class constructor (following the call to `InitializeComponent();`), that uses a Random data variable and the Next method to pick a random number between one and a hundred.
2. Create a text box for the player to enter their guess and a button for the user to tell the computer the guess is ready to evaluate.
3. Evaluate the player's guess and display a message to tell the user if their guess is correct or, if incorrect, if the guess is too high or too low.
4. Display an error message if the user's guess is larger than 100 or less than one.
5. In the subroutine for the "New Game" button, clear the text box and assign a new value to the computer's number. Reset any counters in use.
6. Add an exit button with code to shutdown the program.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:

One possible game board is displayed below. Display messages to the player using label text properties, picture boxes or message boxes.



PROJECT EXTRAS:

- Count and display how many attempts the player takes to guess the computer's number.
- Allow the player a limited number of guesses. After the number of guesses has been exceeded, display the computer's number.

Flip Book Program Instructor Notes

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Use a timer object

SKILLS NEEDED:

- Basic understanding of object properties
- Understanding of static variables

MATERIALS NEEDED:

- C#
- Graphic editor such as Paintbrush

TEACHING SUGGESTIONS:

Animation attracts the interest of most people. This project uses a simple form of animation, the flipbook. Flipbooks use a series of pictures. Each picture is slightly different from its predecessor. The rapid changing of pictures results in the appearance of movement. This is a natural application for a timer.

Two properties are most important in timer objects: enabled and interval. Timer routines must be enabled to execute. Once they are enabled, timer routines execute after the passage of time indicated by the interval time. The time is set in milliseconds.

Students may draw their own animation files or use a set provided by the instructor. If Paint is used to draw images the image attributes should be set to use pixels so that the picture size can be easily determined for the forms picture object.

Note: C# will load and correctly display animated GIF files. An interesting discussion can take place over the relative merits of animated GIF files as opposed to a program controlled animation.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

This program should start a timer that loads or displays a series of pictures. The animation should be easily started and stopped. The program should also exit cleanly.

SUGGESTED SOLUTION:

Create a series of picture files using a graphical editor. The names of these files should have names that are identical except for an identifying number. For example, Dribble1.bmp, Dribble2.bmp... Dribble7.bmp

Initially the timer object enabled property should be set to false. The timer interval should be set to some fraction of a second. Experimentation will help determine the optimum time that will be dependent on the size and loading time of the pictures used. A static counter variable should be incremented in the timer routine. Use the ToString method and the counter to concatenate a file name into a string variable for the Image.FromFile method. Once the counter reaches its maximum value, and the last picture has been displayed, the counter should be reset to zero or one depending on how the first image file has been named.

Use a command button to enable and disable the timer by changing the value of the Enabled property. This will cause the animation to start and stop.

Note: C# picture support includes bitmap (.bmp) files, icon (.ico) files, run-length encoded (.rle) files, metafile (.wmf) files, enhanced metafiles (.emf), GIF files, and JPEG (.jpg) files.

Flip Book Program Student Project

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Use a timer
- Understand and use static variables

OVERVIEW OF PROJECT:

Create a program that displays an animated flipbook.

PROJECT INSTRUCTIONS:

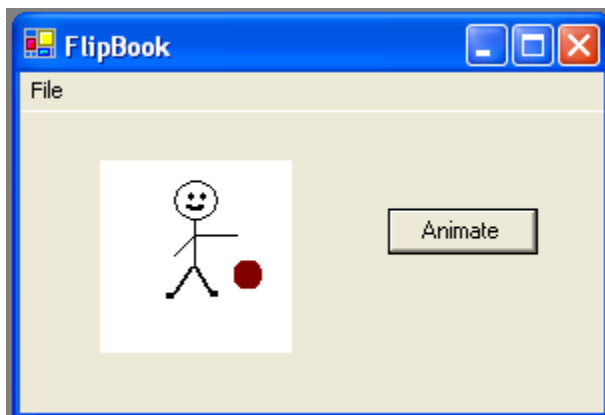
1. Create a picture box on a form
2. Create a series of pictures that vary slightly from each other.
3. Create a command button or buttons to start and stop the timer.
4. Write code in the timer event to load the display the picture series.
5. Program a way to cleanly exit the program.

ADDITIONAL RESOURCES:

- Textbook
- Graphic editor such as Paintbrush

SUGGESTED SOLUTION:

A completed program might look something like this:



PROJECT EXTRAS:

- Add the ability to accept the name of the first file in a picture series to be displayed.
- Add the ability of the user to change the timer interval so that the animation speeds up or slows down.
- Add a second picture box and display two series at the same time. Alternatively, display the same series twice.

Change Making Program Instructor Notes

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 90 minutes

OBJECTIVES:

- Use decision statements
- Use formulas to manipulate money amounts

SKILLS NEEDED:

- Understand mathematical operations including divide and modulus

MATERIALS NEEDED:

- C#

TEACHING SUGGESTIONS:

This application is made for the decimal type. Students who attempt it with normal double variables will have much more trouble because of rounding errors. You may want to explain the difference between these two data types. If you have covered the binary number system you will want to explain that $1/10$ is an infinitely repeating fraction in the binary number system much as $1/3$ is a repeating fraction in the decimal number system. That is what makes accuracy difficult when computers deal with money.

When specifying number constants the compiler assumes integer for number without a decimal point and double for numbers with a decimal point. To avoid conversion problems, the 'm' modifier should be used to tell the compiler that a number constant is a decimal type. For example:

```
if (diff >= .25m)
```

The 'm' is required because diff is a decimal variable and decimal variables cannot be compared to double constants.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

This program should correctly make change for all combinations including those involving pennies. The program should not report that zero of any coin or bill is required.

Test with a number of different values including those that allow for a choice between two nickels and one dime or other similar combinations. The program should not break if the money offered is equal or less than the money required for the item.

The program should also exit cleanly.

SUGGESTED SOLUTION:

The program should start by making sure there is enough money being offered. If the number is enough then the difference in the two values (cost and amount tendered) is calculated. This value is checked against the largest possible bill. The largest currently circulated bills are the 100 dollar bills.

NOTE: Bills of \$500, \$1000, \$5,000 and \$10,000 denominations were once printed but are no longer in general circulation.

The number of 100 dollar bills is determined and that number is added to the list of money given in change. This determination may be done by dividing the difference by 100 and saving the integer value. The amount returned as change in hundreds can be determined by multiplying the number of bills by 100. The new difference is now calculated either by subtracting this value from the current difference or by using the modulus operator.

This same process is followed for each possible item of change.

Change Making Program Student Project

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 90 minutes

OBJECTIVES:

- Use decision statements
- Use formulas to manipulate money amounts

OVERVIEW OF PROJECT:

Create a program to figure change. The program will accept the amount the customer owes. The second value will be the amount of money offered in payment. The program will calculate the amount of change required and tell the cashier exactly how to return the change. The program will report the number of bills and coins that are required. The program will not suggest giving zero of any coin or bill.

PROJECT INSTRUCTIONS:

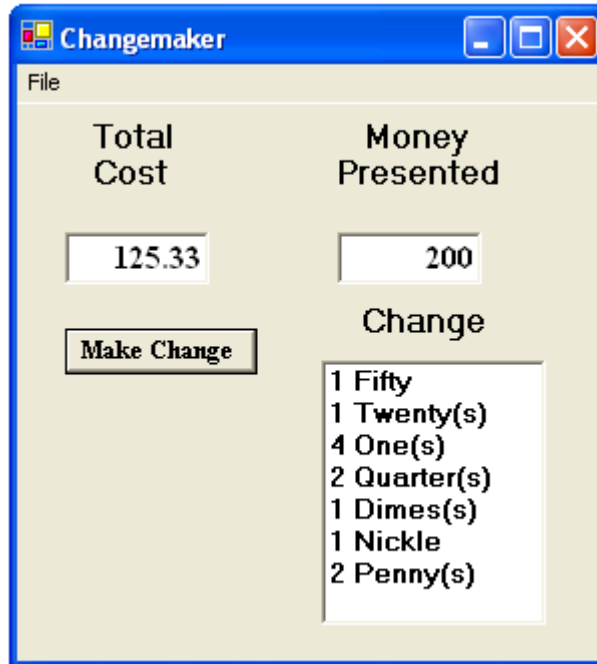
1. Create and label text boxes on a form to accept the amount due and the amount offered.
2. Create a list box to hold the change information.
3. Create a menu option or a command button to tell the computer to calculate the change.
4. In a function, calculate the amount of change due and how many of each bill and coin are required to give the customer their change. Make sure that this function clears previously displayed information to avoid confusion.
5. Program a way to cleanly exit the program.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:

A completed program might look something like this:



PROJECT EXTRAS:

- If insufficient money is offered, tell the user how to give the correct additional money.

Simple House Drawing Instructor Notes

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Understand Drawing objects and properties
- Understand events

SKILLS NEEDED:

- Understanding of a coordinate system

MATERIALS NEEDED:

- C#

TEACHING SUGGESTIONS:

Understanding the coordinate system used by C# is key to this project. The top left-hand corner is 0,0 and the X and Y values of the bottom right corner are positive integers whose value varies based on the size of the form. This means that adding to X moves to the right. Adding to Y moves down and subtracting from Y moves up. It is very common for students to confuse X and Y. Reversing X and Y often results in figures drawn sideways.

The graphics library includes methods for drawing lines, rectangles, ellipses, and polygons as well as pie shapes and filled curves. These methods are used with a graphics object that must be instantiated before the methods can be used. The following statement creates a graphics object named g.

```
Graphics g = this.CreateGraphics();
```

Outlined objects are drawn using a pen whose width and color may be set by the program. Filled objects are drawn using a brush whose color can also be specified. For example:

```
Pen myPen = new Pen(Color.Black);  
Brush myBrush = new SolidBrush(Color.Yellow);
```

The program should respond to a MouseDown event on the form. Select the event options from the Property window and the MouseDown event from the list of events for the object to have the IDE open the appropriate method. The event parameter will return the mouse location. The following code will save this information so that it can be passed to other methods. This example also creates the graphics object that is passed to the DrawHouse method.

```
protected override void OnMouseDown( MouseEventArgs e )
{
    Graphics g = this.CreateGraphics();
    DrawHouse(g, e.X, e.Y);
    base.OnMouseDown( e );
}
```

Encourage students to plan their houses before writing the code. Graph paper can be useful for plotting out the locations of various objects to be drawn.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

A complete house-drawing program should:

- Draw a house with a roof, a door and two windows.
- Draw a circle representing the sun in the sky.
- Add additional objects of the students choosing.
- Clear the form at the user's request.
- Exit cleanly at the user's request.

SUGGESTED SOLUTION:

Clear screen would be implemented by drawing a filled rectangle over the whole form using the form's background color using code similar to that below.

```
Graphics g = CreateGraphics();
g.FillRectangle(new SolidBrush(this.BackColor), 0, 0,
    this.Width, this.Height);
```

Or by using the Clear method of the Graphics object.

```
Graphics g = this.CreateGraphics();
g.Clear(Form1.DefaultBackColor);
```

The code for drawing the house could be implemented as a number of methods. One method could draw the roof, another the main house and others drawing individual windows and the door. These methods would be called by a master method.

Simple House Drawing Student Project

ABILITY LEVEL: Beginner

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Understand objects and properties
- Understand events
- Understand graphic methods and their use

OVERVIEW OF PROJECT:

The object of this program is to draw a simple picture of a house on a form. Draw the house in a location indicated by the user clicking the mouse button.

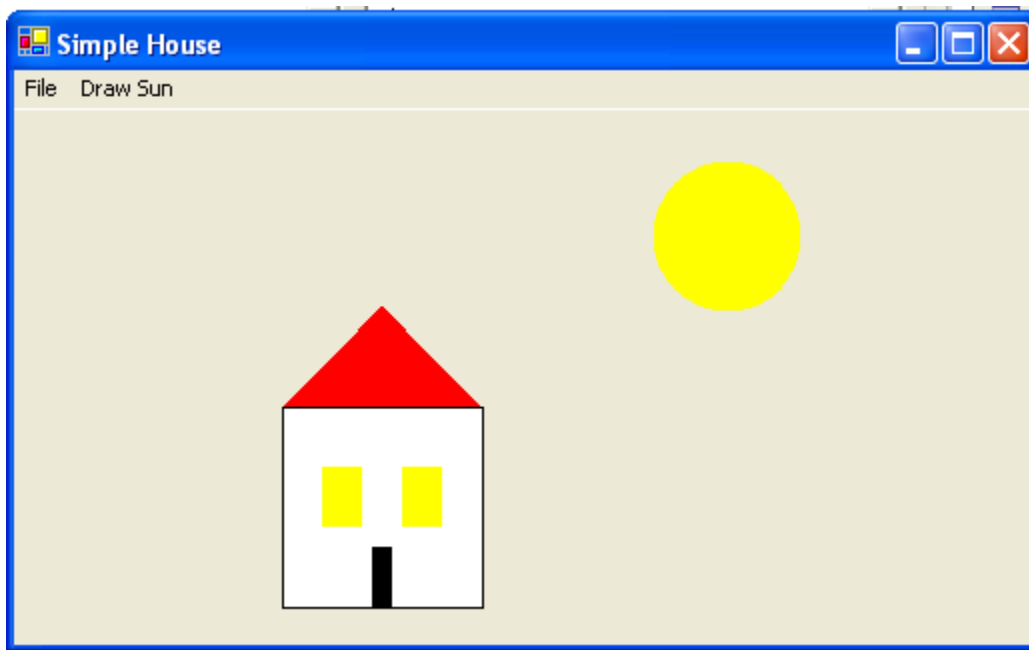
PROJECT INSTRUCTIONS:

1. On a new form, create two command buttons.
2. In the MouseDown routine for the form draw a filled box starting at the location indicated by the X and Y values returned to the routine.
3. Next, draw a triangle as a roof.
4. Draw a tall rectangle in the center or the bottom of the house to represent a door.
5. Draw two white rectangles to represent windows on the house.
6. Somewhere above the house, draw a filled yellow circle for the sun.
7. If the exit button is pressed, shut down the program.

ADDITIONAL RESOURCES:

- Textbook
- Graph paper

SUGGESTED SOLUTION:



PROJECT EXTRAS:

- Draw other objects around the house. For example, a chimney, ornate windows, clouds in the sky, step stones up to the door, or a smile on the sun.
- Change the text property in the forms title bar.
- Make the bottom of the form green and the top part of the form blue.
- Fill in the roof.

Spaceship Invaders Instructor Notes

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 3-4 hours

OBJECTIVES:

- Use timers
- Use the message box method
- Understand and use event handlers
- Create and use arrays of objects

SKILLS NEEDED:

- Understanding of looping constructs
- Understanding of arrays

MATERIALS NEEDED:

- C#

TEACHING SUGGESTIONS:

Two properties are most important in timer objects: enabled and interval. Timer routines must be enabled to execute. Once they are enabled, timer routines execute after the passage of time indicated by the interval time. The time is set in milliseconds. This value is a double which means that very large blocks of time can be used.

This program uses a very simple form of animation. Changing the Top property moves picture boxes up and down the form.

Discuss the difference between moving each object individually and creating an array of object references so that a loop can be used to move the objects. Also explain how the `System.EventHandler` method is used to associate a method with events for multiple objects. Explain how the object reference passed to an event handler is used to identify and manipulate the object.

Once this simple program is completed, encourage students to develop their own original games using the same concepts.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

Picture boxes should move at a uniform rate down the form. Clicking of a box should result in its immediately being placed at the top of the form. The player should have the option of at least three different speeds. Displays should accurately report the number of boxes clicked and missed. The game should have a clear end. Restarting the game should reset all counters and start with the boxes at the top of the screen.

SUGGESTED SOLUTION:

Create a frame for the boxes to move on. While boxes may be created directly on the form, creating the boxes on top of a frame will make defining the range of the box movement easier. Create an initial picture box and define all properties before creating additional objects. Be sure to set the picture property.

Double click on the picture box to create a click response method. This method will cast the object passed to it as sender to a picture box to provide access to the top property. The method should not move the object if the game has been paused by disabling the timer. This code may look like the following.

```
private void picTarget_Click(object sender, System.EventArgs e)
{
    if (timer1.Enabled)
    {
        PictureBox me = (PictureBox) sender;
        me.Top = 10;
        myHits++;
        lblHits.Text = myHits.ToString();
    }
}
```

A timer will be added to the form. The timer tick method will add a value to the top property to move the object down the form.

Two variables will control the movement of the boxes. The first is the timer interval. The distance the box moves each interval is the second variable. The larger the distance moved each interval or the shorter the time between movements the faster the boxes will move. Encourage students to experiment with both settings, once the program works, to find optimum settings. Selecting only one of those variables for use in changing speed during a program run will simplify the program.

The timer routine is the workhorse of this program. In the timer routine, write a loop that incrementally moves each box by adding to its top value. After moving the box, compare it's location to the bottom of the frame. If the top value of the box is larger than the height of the frame added to the top of the frame then that box has dropped below the frame and "escaped." If this is the case, increment the count of escaped boxes and update the display. Also, move the box back to the top of the frame (set the boxes top to the value of the top of the frame).

After each escape, check the escape counter to determine if enough boxes have escaped to end the game. If enough boxes have escaped, display a message box displaying only the yes and no buttons and asking the player if they want to continue with a new game. If the user selects the No button, the `Application.Exit()` method may be used to exit the game. If the player selects the Yes button, call the new game routine to initialize the display and counter variables.

The picture box click routine moves the box to the top of the frame, increments the hit counter and updates the counter display.

Test the program with a single object before moving to the next step.

Copy the original picture box and paste several copies on the form. Add the name of these new objects to the handles clause for the picture click method. This will cause these objects to respond to a mouse click the same way as the original object.

The objects can be moved individually using their name but an array of references to objects can make this process more efficient. Declare an array of picture boxes at the top of the constructor. Following the call to `InitializeComponents`, assign each picture object to an element of the picture box array. The timer tick method can use a loop and the array to move the objects.

Create label boxes for the status display information. Declare form level variables to hold the number of boxes hit and escaped. Another form level variable can control the speed of box movement will facilitate changing that rate through program action.

Allow the user to select different box speeds by using a group of menu options. Label them slow, medium and fast. In the click routine for each option set a specific value to the distance to move each box on each timer event. A more involved and flexible method would be a scroll bar. Use a scroll bar by having the change event place the scroll bar value into the form level variable controlling box speed. Be sure to set reasonable minimum and maximum values for the scroll bar.

Spaceship Invaders Student Project

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 3-4 hours

OBJECTIVES:

- Use timers
- Use the message box method
- Understand and use event handlers
- Create and use arrays of objects

SKILLS NEEDED:

- Understanding of looping constructs
- Understanding of arrays

OVERVIEW OF PROJECT:

Have a number of picture boxes move down a form. If the user clicks on a picture box, move that box back to the top of the screen. Count and display the number of times a box is clicked. When a box moves below the bottom of the screen, count that box as having “gotten away.” After a specific number of boxes getting away, stop the game and allow the user to continue playing or exit the program. Allow the user to quit the game and exit the program.

PROJECT INSTRUCTIONS:

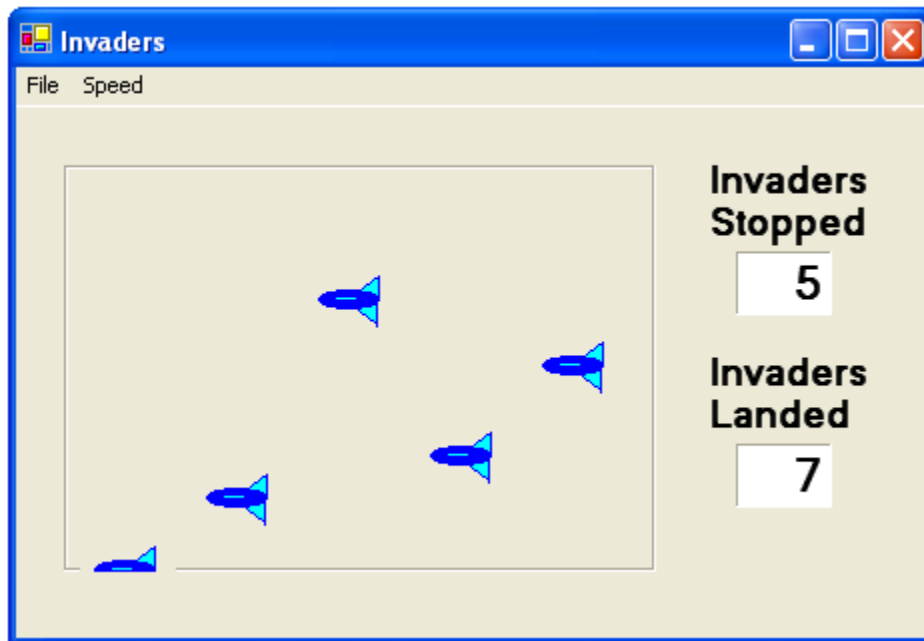
1. Create an object array of picture boxes across the screen.
2. Add a picture or icon file to the picture property of each picture box.
3. Create a timer object. Each time the timer event occurs, move each box down by adding to its Top property value.
4. In the timer event, check to see if a box moves below the bottom of the screen. Increment and display a counter variable if the box escapes. Move escaped boxes back to the top of the screen.
5. If the number of escaped boxes exceeds a specific number, for example ten, call the message box method to ask the user if they wish to play a new game.
6. If the user wishes to play a new game, reset all counters and reposition the boxes at the top of the form. If they do not want to continue, exit the program.
7. Program the picture click routine to move any picture clicked on back to the top of the screen by resetting its Top property value.
8. Add an option, either by buttons, by menu items or by some other way, to allow the user to select the speed the boxes move.
9. Allow the user to exit the program.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:

The image in this example was created using Paint. Either an original image may be drawn or a previously created or acquired image may be used.



PROJECT EXTRAS:

- Add more spaceships.
- Allow the user to select different icons for the picture boxes.
- Create your own characters for the picture boxes.
- Put a different picture in each box.
- Make the pictures move horizontally across the screen rather than down the screen.
- Make the pictures speed up the longer the game runs.
- Use a scroll bar to change the speed of the boxes.
- Add a pause command.

Tic Tac Toe Instructor Notes

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Understand arrays of control references and their use in *if* statements
- Understand and use nested *if* statements
- Use Boolean variables

SKILLS NEEDED:

- Understanding of Boolean operators
- Understanding of control properties and arrays of control references
- Understanding of multi statement and nested *if* blocks

MATERIALS NEEDED:

- C#

TEACHING SUGGESTIONS:

Students can add nine controls to a form at design time or they can have the controls added at run time.

A more advanced project would involve creating the playing board at run time rather than at design time. C# allows for the creation of arrays of controls at runtime. Each element in the array is actually a reference to an object with its own properties. Each object is instantiated using the *new* statement. The newly created object is added to the list of controls for the form using the Add method of the Controls class. For example, the temp object is added to a form with the following statement.

```
this.Controls.Add(temp);
```

The reference is assigned to an element in the array allowing it to be used in a loop. An event handler can be assigned using the EventHandler method of the System object which associates an object and event combination with a method. The following statement adds a method named lblProto_Click to the myBoard[i] object to handle a click event.

```
this.myBoard[i].Click += new System.EventHandler(this.lblProto_Click);
```

Remind students that the name of the event handler does **not** have to include the name of the event.

This project will use a Boolean flag to keep track of whose turn it is. The flag can be easily changed using the assignment *flag* = **NOT** *flag*. Explain that students

do not need to write an *if* statement to first check the current value and that the NOT statement will reverse the value what ever it is.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

A complete Tic Tac Toe program should handle the following cases correctly:

- moves may not be made on occupied squares
- a winning game should be recognized
- no additional moves should be permitted after a game has been won
- the winner (X or O) should be properly identified
- tie games should be recognized
- the board should be cleared either automatically after a game is over or manually by user request

SUGGESTED SOLUTION:

One solution would be to use text properties in label boxes. A second alternative would be to use the Image.FromFile method to load appropriate pictures into picture boxes. A .BMP image of 1 inch by 1 inch created with PaintBrush will fit neatly in a 96 by 96 pixel picture box. Assigning the Image property to Nothing will clear the picture box. Alternatively, a "blank" image may be loaded.

When an object is clicked, first verify that the object is empty. Display a message box or sound the beep if the box is not empty. If the box is empty, display the symbol of the current player and set the objects tag to indicate who has moved into it. Check to see if there is now a winner. Declare a winner when three boxes in a row, column or diagonal are occupied by the same non-zero code. Declare the player who last moved the winner.

If there is no winner, check to see if all the boxes are occupied. Declare a draw if all boxes are occupied and there is no winner.

In case of a draw or a winner being declared, clear the board after the user acknowledges the message. Create a subroutine to clear the board. Call this subroutine from the declare winner code, declare draw code and the clear board or new game button or menu item. Clear the board by resetting the display and setting all the object tags back to zero.

Change the player indicator variable after each move has been made and evaluated.

Students will often assume that any three squares in a row with identical values indicates a winner forgetting that all empty squares have the same value.

Tic Tac Toe Student Project

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Understand arrays of control references and their use in *if* statements
- Understand and use nested *if* statements
- Use Boolean variables

OVERVIEW OF PROJECT:

Create a version of Tic Tac Toe that allows two players to play against each other. The game should report winners, losers, and draws. The program must not allow illegal moves or additional moves after a game has been won.

PROJECT INSTRUCTIONS:

1. Create nine objects to represent the locations for the playing board.
2. Draw lines around the playing objects.
3. Add two buttons or menu options: One for exiting the program and one for clearing the board.
4. In the object click handler, write program code to verify the box is available.
5. If an open box is clicked on, display an X or O depending on which player made the move.
6. After each move, check to see if there is a winner or a drawn game. Report that the game is over if there is a winner or a draw. If there is a winner, no additional moves should be allowed.
7. If the clear game button is pressed reset all playing squares to empty.
8. If the exit game is pressed, shut down the game.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:

Assuming the label box control array is named `myBoard` and the boxes in the array are arranged from 0 in the top left to 8 in the bottom right corners, the following *if* statement will identify a win across the top of the board.

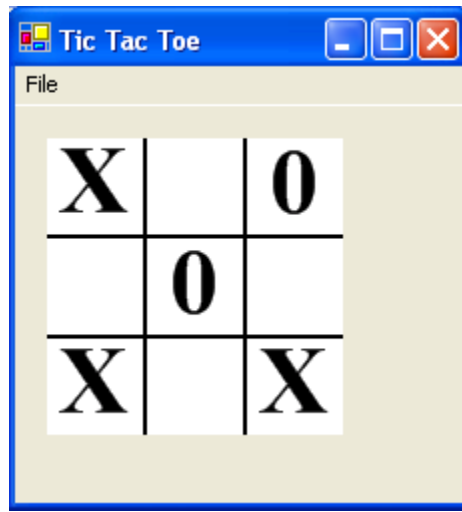
```
if (myBoard[0].Text != "" && myBoard[0].Text == myBoard[1].Text &&  
    myBoard[1].Text == myBoard[2].Text ) return true;
```

This will return the Boolean value True if the top three boxes have the same **non-empty** value. Once all eight possible winning combinations are checked, the program can take appropriate action if no winning combination is found.

The value of a Player flag indicates which player to report as the winner and must be switched after each move.

Checking the text in all nine label boxes can identify a draw. If all the text properties have a value and there is no winner then the game is a draw.

A completed game form may look like the following:



PROJECT EXTRAS:

- Use the Beep statement to notify a user when they click on an occupied box.
- Display a count of the number of draws and the number of times each player wins a game.
- Have the game board clear automatically once the game is over.
- Ask the players if they wish to play a new game at the end of every game.
- Add the playing squares at run time and have the program place them in the correct location.

Picture Display Program Instructor Notes

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Use the open file dialog box

SKILLS NEEDED:

- Basic understanding of object properties

MATERIALS NEEDED:

- C#
- Graphic editor such as Paintbrush
- Picture files

TEACHING SUGGESTIONS:

C# provides a number of useful dialog boxes. These boxes may be used for opening and saving files, setting print options, and selecting colors and fonts. The open file dialog box is typical of these useful tools.

Setting the filter for the open dialog box will probably give students the most trouble. Go over several examples and point them to the help file for reference. Students will have a tendency to define more dialog properties than necessary.

While the picture box has a `SizeMode` property that can be set to `AutoSize`, the form does not automatically resize to hold objects in it. The size of the form may be changed under program control. The objects on a form may also be moved to accommodate changes in the size or number of other objects on a form. Objects on a form should have some distance between them. Explain that students need to make room for borders when they move objects around the form. Also explain how the `Anchor` property is used to keep objects in the same location relative to the sides of the form as its size changes.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

This program should show the open file dialog box with the proper filter for picture files enabled. The file requested by the user should be displayed full size. The form should be resized to include the whole picture. Objects on the form should be relocated as needed so that the picture does not cover them.

SUGGESTED SOLUTION:

The following code sets up an open file dialog box to select graphic image files.

```
openFileDialog1.InitialDirectory = Application.ExecutablePath;  
openFileDialog1.Filter = "Picture  
Files(*.BMP;*.JPG;*.GIF)|*.BMP;*.JPG;*.GIF|All files  
(*.*)|*.*" ;  
openFileDialog1.FilterIndex = 1 ;  
openFileDialog1.RestoreDirectory = true ;
```

Create a picture box in the top left-hand corner of the form. Be sure to set the `SizeMode` property to `AutoSize` so that the box will expand or collapse to fit the picture loaded. Add a common dialog object. Create boxes to display the height and width of the picture with appropriate labels for each box. Create an open file menu option under a file menu. In the open menu click routine, assign a filter string to the filter property of the open file dialog box. Once a file has been selected, load the file into the picture box using the `Image.FromFile` method.

Call a routine to adjust the size of the form to hold the picture. Also, move the display boxes as required so that the picture box does not obscure them. Lastly, copy the pictures height and width into the appropriate boxes.

Note: C# picture support includes bitmaps (.bmp), icons (.ico), run-length encoded files (.rle), and metafile (.wmf) files.

Picture Display Program Student Project

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 1 hour

OBJECTIVES:

- Use the open file dialog box

OVERVIEW OF PROJECT:

Create a program that opens and displays a picture. The program will also display size information about the picture. This program may be used to display the size of pictures in pixels so that picture boxes may be created for them in other programs.

PROJECT INSTRUCTIONS:

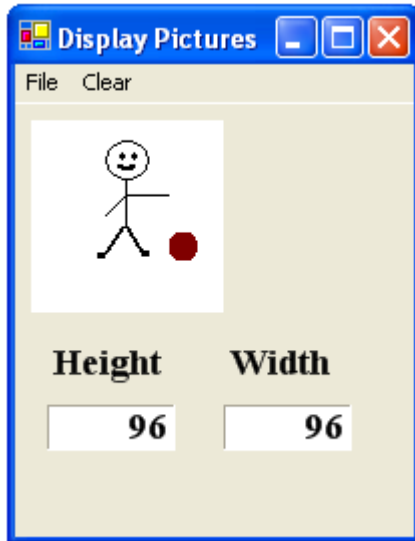
1. Create a picture box with the SizeMode property set to AutoSize.
2. Create a menu option to open a file that will use the open file dialog box.
3. Set the filter and initial directory properties for an open file dialog box.
4. Load the picture file selected by the open dialog box into the picture box.
5. Adjust the size of the form, if necessary, to hold the whole picture. Move any objects covered by the picture.
6. Copy the picture height and width information into properly labeled objects.
7. Program a way to cleanly exit the program.

ADDITIONAL RESOURCES:

- Textbook
- Graphic editor such as Paintbrush
- Picture files

SUGGESTED SOLUTION:

One possible solution would look like this.



PROJECT EXTRAS:

- Display the full name and path of the file opened.
- Convert the size in pixels into inches for a second display.
- Define different filters so that a user can choose to see just one file type at a time.

Multiple Choice Grading Program

Instructor Notes

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Read and parse a sequential data file

SKILLS NEEDED:

- Understanding of loops
- Understanding of string manipulation functions
- Understanding of decision making statements

MATERIALS NEEDED:

- C#
- Data file with names and answers

TEACHING SUGGESTIONS:

Reading data in from a sequential file has many benefits for both students and instructors. Tops among them may be easy reproducibility for testing purposes. Once a set of data is developed and tested, it can be used to prove other programs by comparing results with what is expected. For this reason, as well as to avoid problems caused by faulty data sets, having all students use a common data file is generally desirable.

While loops are a natural tool for reading sequential data files. Introduce the use of exception handling so that indeterminate loops can be used to read through the file without knowing how many records will be read. Make students aware that if they want to know how many records they have read, they have to keep track of record numbers themselves. Some students will be made aware of how many records are in a test set and write programs that use a *for* loop which they may find easier. Their lives will be complicated however if they their program will be evaluated using a second, different, file then the file they use for development. This has the advantage of more accurately simulating real word conditions. In the real world, programmers can rarely count on knowing the size of the data set their programs will use.

Students should have been exposed to the notion that a lower case letter is not equal to an uppercase letter in C# by this time. They may need a reminder at this time. Mixing the case of answers in the data set or using uppercase in the main data set and providing a lower case answer key will show them the need to match cases. The `ToLower` and `ToUpper` string methods will prove useful here.

List boxes are a convenient tool for displaying variable amounts of data. A result string can be created by concatenation different strings and added to a list box. The tab constant (\t) can be used to separate strings when building a string to add to a list box.

Be aware that some students may use parallel list boxes to create columns. If the font attributes are not the same, those boxes may have alignment problems. They must also be sure to clear the right boxes at the right times.

RESOURCES:

- Textbook
- Text editor for creating a data file

SUGGESTED EVALUATION:

A complete program should:

- Read and display all names in the data file
- Properly score each students set of answers
- Display the proper grade for each student
- Calculate and display the average number of correct answers for the whole data file.
- Exit the program cleanly

SUGGESTED SOLUTION:

Prepare an ASCII text data file for students to use. The file should have at least 10 entries.

The program should use a While loop checking using exception handling to check for end of file to read the data file. The program should count each student as it reads its record. A variable should be used to add the total correct answers for each student. Use this variable with the student counter variable to calculate the average number of correct answers. Displaying the average as part of the same routine that counts students and adds totals avoids the need for form level variables.

Care must be taken when determining when to open and close the data file. Trying to open a file that is not closed, or trying to read a file that is not yet open or has been opened and closed, are common errors. For this reason, it is recommended that the open and close be done in the same routine. Alternately, the file should be opened once and not closed until the exit program routine is called.

Multiple Choice Grading Program

Student Project

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Read and parse a sequential data file

OVERVIEW OF PROJECT:

Create a program to correct and grade a set of multiple choice test results. Read a set of correct answers, from a data file or a text box depending on what your instructor has specified, and compare it with the answers in a data file. The data file will have two fields, separated by a comma, for each student. The first field will contain the student's name. The second field will contain a set of answers to a multiple-choice test. For each student, compare the answer key answers to the answers from the student's record. Keep a count of the number of answers from the student's record that match the answers in the answer key.

For each student, display the student's name, the number of correct answers, and a letter grade. Calculate the letter grade scale based on information provided by the instructor.

Calculate the average number of correct answers for the class and report it in a separate picture box.

PROJECT INSTRUCTIONS:

1. Create a listbox or other object large enough to display the names, correct count and letter grade of each student.
2. Create code behind a command button to read the data file and report results
 - a. Open the data file
 - b. Read the answer key
 - c. Read a students record
 - d. Compare student's answers to answer key and count correct answers
 - e. Display student's name, number correct and letter grade
 - f. Count number of students and build running total of correct answers
 - g. Loop until all student records have been read and processed
3. Calculate and report average number of correct answers
4. Create an exit button with the appropriate code to end the program

ADDITIONAL RESOURCES:

- Textbook

- Instructor supplied data file

SUGGESTED SOLUTION:

The screenshot shows a window titled "Grading Program" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, there is a label "Answer Key" followed by a text box containing the string "ABCDABCDAB". Below this is a list box containing a table of student data. At the bottom of the window, there is a "Grade" button, a label "Average Grade" above a text box showing "6.7", and an "Exit" button.

Student Name	Score	Letter Grade
Jean	10	A
Jack	7	C
George	9	A
Paul	5	F
Steven	7	C
Jackie	6	D
Susan	0	F
David	8	B
Dan	8	B

PROJECT EXTRAS:

- Report the average letter grade as well as the average number of correct answers.
- Keep track of the highest and lowest scoring students and report them in a separate box.

Multiple Choice Grading Program with Structures

Instructor Notes

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Use a User Defined type with the Struct statement

SKILLS NEEDED:

- Understanding of sequential files
- Understanding of arrays
- Understanding of string manipulation functions
- Understanding of decision making statements

MATERIALS NEEDED:

- C#
- Data file with names and answers

TEACHING SUGGESTIONS:

User defined types allow the programmer to group related data elements and manipulate them as a single item. They are very often used to define record types for files but may be used in other ways as well. In this case we want students to group names, as read from a data file, and number of correct answers, as calculated by the program.

User defined types should be declared so that they are available throughout the program.

We will also want the program to rate each student in the data file based on information that it will not have until all data records have been read. Creating an array of a user defined type allows us to manipulate the data without having to read the file a second time.

While the rating of students as average, above average, or below average may be done in the same routine as the initial reading of the data file, having students code a separate routine reinforces the use of local as opposed to form level or global variables.

RESOURCES:

- Textbook
- Text editor for creating a data file

SUGGESTED EVALUATION:

A complete program should:

- Read and display all names in the data file
- Properly score each students set of answers
- Display the proper grade for each student
- Calculate and display the average number of correct answers for the whole data file.
- Rate all students as having an average, above average, or below average number of correct answers.
- Exit the program cleanly.
- Make proper use of form level and local variables.

SUGGESTED SOLUTION:

Prepare an ASCII text data file for students to use. The file should have at least 10 entries.

The student record array should be defined at the form level to allow its use across a number of routines. A form level variable should also be defined to hold the count of elements in the array that is being used. An alternative solution would be for the name after the last used to be set to some known flag value.

The program should use a while loop and use exception handling to check for the last data in the file. The program should count each student as it reads its record. This counter variable should also be used as an index into the array of student records. A variable should be used to add the total correct answers for each student. This variable will be used with the student counter variable to calculate the average number of correct answers. This count should be entered in a field of the student record array.

Care must be taken when determining when to open and close the data file. Trying to open a file that is not closed or trying to read a file that is not yet open or has been opened and closed are common errors. For this reason, it is recommended that the open and close be done in the same routine. Alternately, the file should be opened when the program first begins and not closed until the exit program routine is called.

Multiple Choice Grading Program with Structures

Student Project

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 2 hours

OBJECTIVES:

- Use a User Defined type with the Struct statement

OVERVIEW OF PROJECT:

Create a program to correct and grade a set of multiple choice test results. Read a set of correct answers, from a data file or a text box depending on what your instructor has specified, and compare it with the answers in a data file. The data file will have two fields for each student. The first field will contain the student's name. Second field will contain a set of answers to a multiple-choice test. For each student, compare the answer key answers to the answers from the student's record. Keep a count of the number of answers from the student's record that match the answers in the answer key.

For each student, display the student's name, the number of correct answers, and a letter grade. Calculate the letter grade scale based on information provided by the instructor. Store the student's name and the number of correct answers in an array of a structure type.

Calculate the average number of correct answers for the class and report it in a separate box.

Using the array of student records, display a list for all students. Report if a student had an average, above average or below average number of correct answers.

PROJECT INSTRUCTIONS:

1. Create a structure to hold a string name field and a numeric count field.
2. Create a list box large enough to display the names, correct count and letter grade of each student.
3. Create code behind a command button to read the data file and report results
 - a. Open the data file
 - b. Read the answer key
 - c. Read a students record
 - d. Compare student's answers to answer key and count correct answers
 - e. Display student's name, number correct and letter grade

- f. Count number of students and build running total of correct answers
 - g. Loop until all student records have been read and processed
4. Calculate and report average number of correct answers
5. Create code behind a command button to loop through all student records and display if each student has an average, below average or above average number of answers correct.
6. Create an exit button with the appropriate code to end the program.

ADDITIONAL RESOURCES:

- Textbook
- Instructor supplied data file

SUGGESTED SOLUTION:

Name	Number of Correct Answers	Grade	Relative to Average
Jean	10	A	Above Average
Jack	7	C	Average
George	9	A	Above Average
Paul	5	F	Below Average
Steven	7	C	Average
Jackie	6	D	Below Average
Susan	0	F	Below Average
David	8	B	Above Average
Dan	8	B	Above Average

Grade: 6.7

PROJECT EXTRAS:

- Report the average letter grade as well as the average number of correct answers.
- Keep track of the highest and lowest scoring students and report them in a separate box.

Letter Counter Program Instructor Notes

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 2 Hours

OBJECTIVES:

- Use arrays and string manipulation methods

SKILLS NEEDED:

- Understanding of different variable types

MATERIALS NEEDED:

- C#

TEACHING SUGGESTIONS:

Explain that characters are stored internally as numbers. Because characters are numbers they can be used, with limitations, inside formulas. Be sure to discuss the pitfalls of assuming that a character has a specific numeric value. While most systems use the ASCII coding sequence there are others. The .NET Framework uses Unicode internally for example. For this reason it is generally better to use character constants rather than numeric constants to make conversions.

For more information on Unicode, see the Web site of the Unicode consortium (<http://www.unicode.org/>).

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

This program should correctly count the occurrences of each letter in a text message. A text string with known counts should be used to verify the correctness of the program. The program should not report zero occurrences of a letter. The program should also exit cleanly.

SUGGESTED SOLUTION:

An integer array of 26 values should be used to store the counts for each letter. The ASCII value of each letter can be used to compute an offset into this array. For example:

```
myCount[myString[i] - 'A']++;
```

Once the input string has been counted, a simple loop may be used to add the results to a list box. When the results are displayed the array should be checked for non-zero values and those values, along with a character representation of the letter should be added to a list box.

Other display options include a control array of boxes labeled with letters. When the counts are displayed, any box that does not have a value can have its visible property set to false.

These results will appear in alphabetical order because the counter array is in order. If a programmer wanted they could set the Sorted property to True and put the count first in the string item added to the listbox. This would put the results in order by number of occurrences in most cases. Because 10 sorts before 2 in alphabetical order, leading zeros would have to be used to keep this data in order if multiple digits were required.

Words can be counted by counting spaces. The program must take into account that the last word in the box will not generally have a trailing space. Sentences can be counted by counting periods, exclamation marks and question marks.

Letter Counter Program Student Project

ABILITY LEVEL: Intermediate

APPROXIMATE COMPLETION TIME: 2 Hours

OBJECTIVES:

- Use arrays and string manipulation methods

OVERVIEW OF PROJECT:

Letter counting is an interesting application with real world use. Linguists count the letters used in literature as they study changes in language. Cryptographers count letters to help them break ciphers. This project will count the letters in the words and sentences entered in a text box.

PROJECT INSTRUCTIONS:

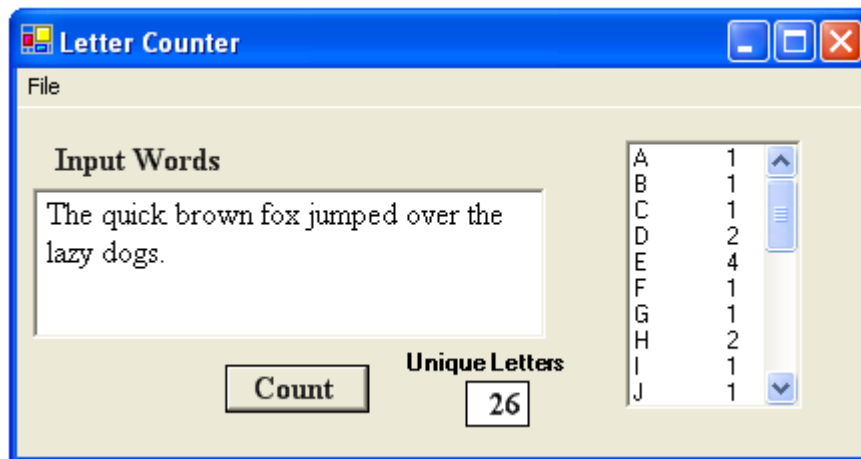
1. Create a new project.
2. Add a text box for the user to enter the text to evaluate. Set the Multiline property to True so the user's text will wrap at the end of the line.
3. Add a list box or other object to use to display the counts of the letters that appear.
4. Add a command button or menu option to start the count. Inside the counting routine:
 - a) Clear all counters and displays so that a new text will not include information from a previous count.
 - b) Clear the area or areas where results will be displayed.
 - c) Create a loop that will examine each individual character.
 - d) If the character is a letter, add to the count for that letter.
 - e) Evaluate the next character until all characters have been evaluated.
 - f) Display the number of times each character appeared. Do not list any character that did not appear in the string.
5. Add a command button or menu option to exit the program.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:

A completed program might look something like this:



PROJECT EXTRAS:

- Count the number of unique letters in the text string.
- Count characters other than letters.
- Count words. Most words will end with a space. The last word will usually not have a space after it.
- Count sentences. Remember that periods are not the only characters that end sentences.
- Report on the number of words in each sentence.
- Report the average number of letters in each word.

Hangman Program Instructor Notes

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 3 hours

OBJECTIVES:

- Use String manipulation methods
- Use looping and decision statements
- Read a sequential file

SKILLS NEEDED:

- Basic understanding of object properties

MATERIALS NEEDED:

- C#

TEACHING SUGGESTIONS:

This game involves a variety of programming concepts. This program will involve reading words from a file, using loops to search through strings and arrays to track and display information. This project requires planning. Students should be encouraged to break the game down into its component parts.

The random word can be selected in any of several ways. The words can be selected from an array. The array can be populated using a series of assignment statements but this is not desirable. It is too limiting. The array should be populated with words read from a sequential file. The program must determine how many words are available and use the Random class to select a number in that range. Once selected, the word is stored for use throughout the program.

The length of the word selected can be determined using the Length property. This will allow the program to display the number of letter positions. This may be done in a number of ways including an array of label boxes that will be set to visible or invisible depending of the number of letters in the answer.

The letter guessed can be accepted from a text box. It may also be selected from a list of letters. There are many options for the programmer to select. The work begins once the letter has been selected.

The letter guessed and the letters in the answer word should all be converted to a common case to simplify string comparisons. Students may use the IndexOf method to search for a letter in the answer word. This method will return the location in the word if the letter is found. This is limited, as it will only find the first occurrence of the letter. Using a loop and indexing through the word to compare the letter guessed to each letter in the target word is a better method to search the answer word. This will allow the program to easily display the letter in the

correct location in the blank word displayed. As a letter is found in the target word the corresponding position on the display array can be updated.

If the letter is not found, the hangman picture must be updated. This picture may be drawn using various Graphics methods including DrawLine and DrawEllipse. The picture may also be drawn in several image files using Paint or other drawing tool. The appropriate image for the number of incorrect guesses can be loaded into a picture box. Allow students some flexibility in how they display the “victim” but make sure that they do not spend too much time on this at the expense of the meat of the program.

The programmer must also decide how to handle the case of a letter being guessed a second or third time. If the letter has been guessed already, should that be a wrong guess? If so, the program will have to check against a list of previously guessed letters. This should be done before the letters are checked against the answer word.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

The program should select a word at random. The program should correctly identify letters as being part of a word regardless of what case letter is entered. The program should identify a win when all the letters in a word have been identified. A letter that appears more than once in a word should only have to be selected once. The program should draw a picture in sections as incorrect letters are guessed. The game should end if the word is not guessed in a set number of letter guesses.

SUGGESTED SOLUTION:

Create an array of label boxes to show the individual letters in the word to be guesses. Use a loop and the Length property of the word to set only the required label boxes visible. If the label text properties are set to a known value at this time then can easily be checked to see if they have been filled by a correct guess.

A loop can scan each label box and if none of them have the empty value in them the game is over and the player has won.

The same counter that is used to track the number of incorrect guesses can be used to define what parts of the hanging image are required. If the hanging man is drawn on a picture box using Graphics methods, the parts of the picture should be drawn using the Paint method of the class. The Refresh method can be used to cause the picture to be redrawn whenever a new piece is to be added. This will also ensure that the picture is redrawn if the picture box is hidden and unhidden.

Hangman Program Student Project

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 3 hours

OBJECTIVES:

- Use String manipulation methods
- Use looping and decision statements
- Read a sequential file

OVERVIEW OF PROJECT:

Write a program to play the game hangman. On program startup, the program must read a list of words from a text file and randomly pick one word for the player to guess. The player guesses one letter at a time until either all the letters in the word have been guessed or some number of wrong guesses has been made without the word being guessed.

The program should keep track of letters guessed and display that list on request. Display a cute little picture to show the player how close they are to losing.

PROJECT INSTRUCTIONS:

1. Create an ASCII text file with some number of words.
2. Create a new project. In the class constructor, open and read the word list. Save the words in an array.
3. Create a text box for the player to enter a letter they guess is in the word. Add a command button to signal that there is a guess to be evaluated.
4. Create a picture box to load or draw the picture of the victim as the player guesses incorrectly.
5. Create a list box to display letters that are used as guesses.
6. Have the program select a word randomly from the word list. Set a label box visible for each letter in the word.
7. Create and format label boxes to display the letters as they are guessed. The boxes should be visible at the start of the game in a way that tells the player how many letters there are in the word.
8. In response to a signal from the user, search the word to determine if the guess is in the word. If it is, show the letter in the correct location in the word display. If the letter is not in the word, display the next picture in the picture box. If the maximum number of guesses has been taken or the word has been completely guessed, end the game. Allow the user to request a new game.

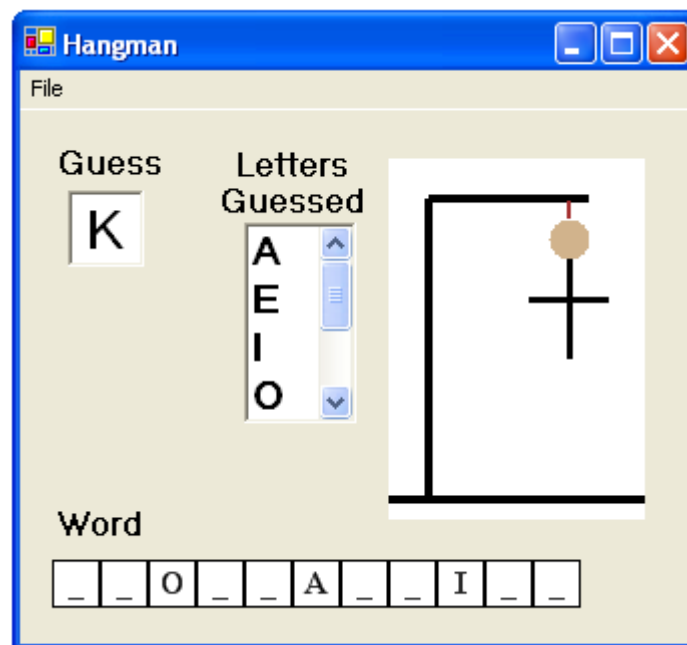
9. Add a menu option to restart the game. All counters must be reset, a new word must be selected, and the label boxes must be cleared and displayed in the correct number.
10. Add a menu option to exit the game.

ADDITIONAL RESOURCES:

- Textbook
- Word list

SUGGESTED SOLUTION:

A completed program might look something like this:



PROJECT EXTRAS:

- Check all guesses and penalize the player if they guess a letter a second time.
- Create a two-person version of the game. Allow a player who guesses correctly to continue to guess but lose their turn on an incorrect guess. Keep a different hangman picture for each player. Or instead of using a victim, award money to a player making a correct guess and subtract money for an incorrect guess.
- Allow a player to add additional words to the program's word list.

Drawing Program Instructor Notes

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 3 hours

OBJECTIVES:

- Understand graphic objects
- Understand handling events including mouse and paint events
- Use the ellipse, rectangle and line methods to draw objects on a form

SKILLS NEEDED:

- Understanding of arrays
- Understanding of the scope of variables
- Understanding of control structures (*if* and/or *switch*)

MATERIALS NEEDED:

- C#

TEACHING SUGGESTIONS:

Several picture boxes will be used to display colors and drawing options in this program. The objects that will have shapes drawn on them should have these objects drawn in response to the Paint event. This will ensure that these shapes are drawn not only when the form is first loaded but redrawn when the form is hidden and returns to the front of the screen. Students may need to have the Paint event explained to them. They may also need to have adding handlers and the concepts around events explained if they have not used them in earlier projects.

Class variables may also be used to keep track of related pieces of information. For example, using class variables to keep the X and Y coordinates registered by a mouse down event handler allows the programmer to use them with the X and Y coordinates registered by a mouse up event. Having both sets of coordinates allows the program to set the beginning and ending of lines or boxes to draw.

The click event handler for color boxes and shape boxes should set variables that are used by the MouseUp handler to set the pen or brush color, set the pen thickness and select what object is drawn.

The easiest way to clear the form is to draw a filled rectangle of the form's background color that covers the whole form. Objects, such as the color selection boxes and any control buttons will not be covered by this filled rectangle.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

A complete drawing program should:

- Allow a user to select one of at least 4 object types to draw
- Include a set of color selection boxes initialized with different background colors that allow a user to select colors to use for drawing objects
- Draw selected objects using the selected color at locations indicated by the user
- Clear the form on a users request
- Exit the program cleanly

SUGGESTED SOLUTION:

Use class variables to track the current color to use, the object to draw and the beginning location selected by a user.

A set of colored picture boxes will allow for selection of drawing colors using the click event and saving the background color associated with that box. Line thickness and shape selections can be accomplished with similar boxes that respond to a click event.

Students may use any number of object types in their programs. Check boxes or label boxes may also be used. Any object that may be selected and which may be used to display a color or an object may be used.

Lines are drawn with the DrawLine method. Circles and ellipses are easily drawn using the DrawEllipse and FillEllipse methods. The DrawRectangle and FillRectangle methods are used for rectangles. Other shapes can be done with the DrawPolygon and FillPolygon methods. Those methods require more effort and several mouse up and down events but allow for more complex object to be drawn. Students who attempt these options should plan for them carefully.

Students should be encouraged to be creative in both the objects supported by their program and the implementation of those objects.

Drawing Program Student Project

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 3 hours

OBJECTIVES:

- Understand graphic objects
- Understand handling events including mouse and paint events
- Use the ellipse, rectangle and line methods to draw objects on a form

OVERVIEW OF PROJECT:

The object of this program is to create a drawing program that allows a user to select colors to use and objects to draw. The user will select a colored object to indicate which color to use for drawing. A second set of objects will display objects the program will draw for the user. After selecting a color and an object, the user will use the mouse to show the program where to draw the object.

The user must have at least four objects to draw.

PROJECT INSTRUCTIONS:

1. Create a number of boxes on the form and set their background colors.
2. Create a second set of objects to indicate shapes that the program will draw.
3. Use the Paint event for the shape objects to draw examples in each box.
4. Create global variables to store the shapes and colors selected by the user. The user will click on the appropriate sample boxes choose a shape or color.
5. Create program code on the form to draw objects of the selected shape and color in response to mouse actions.
6. Add a program option to clear the form.
7. Create an exit button or exit menu item with the appropriate code to end the program.

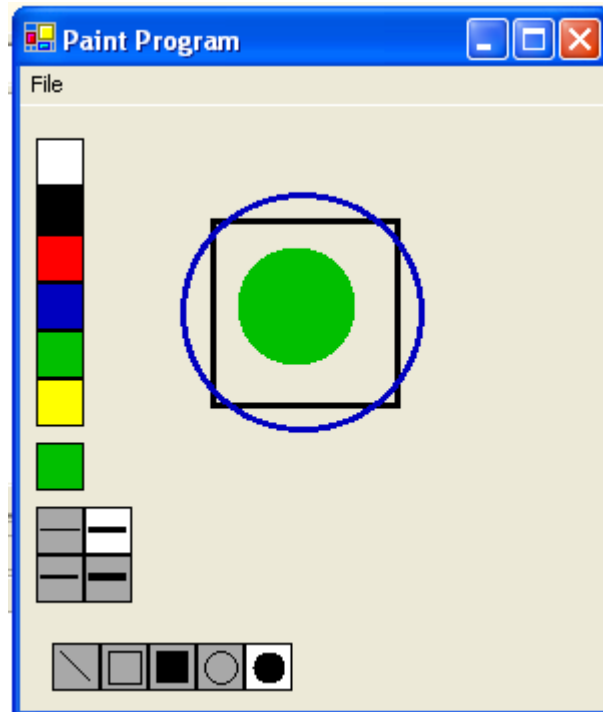
ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:

In the solution below, the square at the bottom of the color array shows the currently selected color. Change the background of that square when a new selection is made.

This sample also includes an extra option that lets the user select the thickness of lines drawn.



PROJECT EXTRAS:

- Create additional shape options for the user. For example, triangles or pentagons.
- Create the color and/or shape selection objects on their own forms.
- Change the color of the shapes to match the currently selected color
- Allow the user to select different pen thicknesses

Roman Numeral Conversion Instructor Notes

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 3-4 Hours

OBJECTIVES:

- Use complex decision making structures

SKILLS NEEDED:

- Understanding of decision making structures
- Understanding of string handling functions

MATERIALS NEEDED:

- C#
- Roman numeral conversion chart

TEACHING SUGGESTIONS:

This is as much logic puzzle as a programming problem. Occasionally a student will suggest a single *if* statement for each number. The prospect of 4,999 statements doesn't always daunt them. Reminding them that the program must convert both ways resulting in 9,998 lines will usually convince them that this is not a practical solution.

The number 4,999 was chosen for a practical reason. Roman numerals at five thousand and above involve characters with lines above them. Those characters are not in the ASCII character set. Four thousand nine hundred ninety nine is a number large enough to provide a challenge to most students.

Students should be reminded to validate the data. What will happen if a real number or a non digit is entered in the Arabic data box? Or if an invalid character is entered in the Roman number box? How is upper and lower case handled? You may want to remind students of the ToUpper method of the string class.

Encourage students to think out how they convert these numbers by hand. When converting from Arabic to Roman, most people use a method involving subtracting the highest possible number associated with a symbol. That symbol is set aside and the next symbol is examined. This continues until no more symbols are needed and a Roman numeral has been built. This suggests several possible algorithms.

Converting from Roman to Arabic is even easier. Evaluate the string from left to right converting each symbol to its numeric value and adding it to a counter. The tricky part is handling cases of leading symbols whose value is subtracted from

its larger neighbor. Compare each symbol with the symbol to its right to determine the appropriate number.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

Verify that the program works for a wide range of numbers. The number 1994 and 4999 provide good tests as they involve a number of leading subtraction symbols. Verify that a number converted in one direction results in the same number when converted back.

SUGGESTED SOLUTION:

Create two text boxes to accept numbers to convert. Below each, create a command button labeled for the appropriate conversion.

Converting from Arabic to Roman

Use the Convert class to copy the number in the text box into an integer variable. If the number is greater than or equal to 4,000, subtract 1000 from the number and append an M character to the string used to build the Roman numeral. Do the same for 3,000 and on down to 1,000. Compare the number to 900, if still greater then subtract 900 and append CM to the end of the string. Continue the same pattern to 500, 400, 300, 200, 100, 90, 50, 40, 30, 20, 10, 9, 5, 4, 3, 2, and 1. At this point, the whole number will be accounted for and the program should display the result.

Converting from Roman to Arabic

Initialize a counter variable. Also, initialize a string variable that will be used to hold the previous character for characters after the first. Start a loop to examine each character in the string entered into the text box. If the letter is an I add one to a counter. If it is a V, add 5 if the previous character was not an I. If the previous character was an I then that one should have been subtracted rather than added. Compensate for that addition by adding 3 to the counter rather than 5. Follow this format for the remaining characters up to M for a thousand. Remember that there is only one possible character to subtract before each symbol. The I comes before X (10) as well as V (5). The X comes before L (50) and C (100). The C comes before both D (500) and M (1000).

After evaluating all characters, display the resulting number in the appropriate box.

Roman Numeral Conversion Student Project

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 3-4 Hours

OBJECTIVES:

- Use complex decision making structures

OVERVIEW OF PROJECT:

This program will convert regular numbers into roman numerals and convert Roman numerals into regular numbers. The program will read in a number or string of characters as appropriate. The user will indicate which conversion is to take place. The number will then be converted and displayed properly. The program must correctly convert numbers up to 4,999.

PROJECT INSTRUCTIONS:

1. Create two text boxes on a form. Label them.
2. Create two command buttons. One to convert an Arabic number into a Roman numeral. The second, to convert a Roman numeral into an Arabic number.
3. Write the code to convert each number type to the other type.
4. Place the converted number in the appropriate box.
5. Add an option to exit the program.

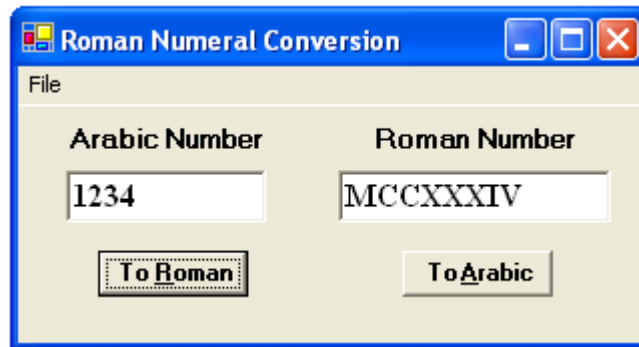
ADDITIONAL RESOURCES:

- Textbook
- The following Roman numeral conversion chart

1	I	40	XL
2	II	50	L
4	IV	90	XC
5	V	100	C
6	VI	500	D
9	IX	900	CM
10	X	1,000	M

SUGGESTED SOLUTION:

A completed program might look something like this:



PROJECT EXTRAS:

- Numbers over five thousand require symbols with lines above them. Create picture files that contain individual characters including the characters at 5,000 and above. Use these picture files to display numbers greater than 5,000.
- If the user enters Roman numerals in lower case, redisplay them as upper case.

Palindromes Instructor Notes

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 2-3 Hours

OBJECTIVES:

- Use advanced string manipulation
- Understand involved loop statements

SKILLS NEEDED:

- Understanding of loops
- Understanding of string manipulation functions

MATERIALS NEEDED:

- C#

TEACHING SUGGESTIONS:

Extra characters, characters outside the range of characters you are interested in, and differing case of characters are the biggest obstacle in determining palindromes. Be sure that students are aware of what characters and characteristics of a character matter.

The `String.Substring` method or indexing a string can be used in this program to examine individual letters. The `Length` property will be used to control when the loop starts and ends.

Consider reviewing user written functions if necessary. Functions make this project much easier.

Valid test data is a necessity for any program. Consider supplying good test data to students to ensure that they are able to test their program effectively. If you do not supply test data, emphasize the importance of selecting good test data.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

Test a number of strings with known solutions to ensure that all cases are found. The classic palindrome "Madam I'm Adam" makes a good test of a longer palindrome. In the sentence "Hello Madam, I'm Adam and this is my friend Bob", the following sub-strings of length 3 are palindromes:

ada - from the word "Madam"
mim - from the words "Madam, I'm"
ada - from the word "Adam"
ama - from the words "Adam and"
isi - from the words "this is"
sis - from the words "this is"
bob - from the word "bob"

SUGGESTED SOLUTION:

Removing extra characters is the first step. Use the `ToUpper` method to set the string in the input text box to all upper case. Use the `Length` property to determine the length of the string and set-up a loop that examines each character. Concatenate every valid character, between A and Z or 0 to 9, into a string variable. This will discard the characters outside the valid range.

Create a *bool* method to evaluate a string as a possible palindrome. Initialize one counter to one. Initialize a second counter to the length of the string. Use these counters to compare the first and last character. If they are the same, increment the first counter and decrement the second. If they are not the same, set a flag indicating that the string is not a palindrome. Continue comparing characters until the center of the string is reached or a mismatch is found. If the center is reached without a mismatch, return `True` as the value of the method. If a mismatch is found, return `False` as the method value.

If no sub-string length is entered, submit the whole input string to the check palindrome method. Display the appropriate message based on the return from the method.

If a sub-string length is entered, create a routine to select sub-strings and send them to the check palindrome method. The loop will start at the beginning of the string and increment up to the length of the string minus the one less than the length of the sub-string length. Use the `SubString` method to select a series of sub-strings of the requested length. Send these sub-strings to the check palindrome method. If the method returns `True` for a sub-string, display the sub-string in the picture box. If no palindromes are found, display an error message.

Palindromes Student Project

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 2-3 Hours

OBJECTIVES:

- Use advanced string manipulation
- Understand involved loop statements

OVERVIEW OF PROJECT:

A palindrome is a word or phrase that reads the same backwards as forwards. Some common examples are "Mom", "Dad", and "Bob" or even the sentence "Madam, I'm Adam." When determining a palindrome, the case of letters is ignored, as is any spacing or punctuation if the phrase happens to be a sentence.

The goal of this program is to determine if a word or phrase is a palindrome and to determine if any sub-strings of words or phrases are palindromes. For example, are there any groups of three letters that are a palindrome? Only the 26 letters (a...z) and 10 numbers (0...9) are to be considered.

PROJECT INSTRUCTIONS:

1. Create a text box to accept the possible palindrome.
2. Create a second text box to accept the size of palindromes to find.
3. Create a picture box to display the palindromes found in the input text box.
4. Write code in a command button routine to check for palindromes.
5. If sub string palindromes were requested, display any palindromes found in a list box. (Clear the list box before adding new palindromes.)
6. Display a message box if no palindromes were found.
7. Create an end program option

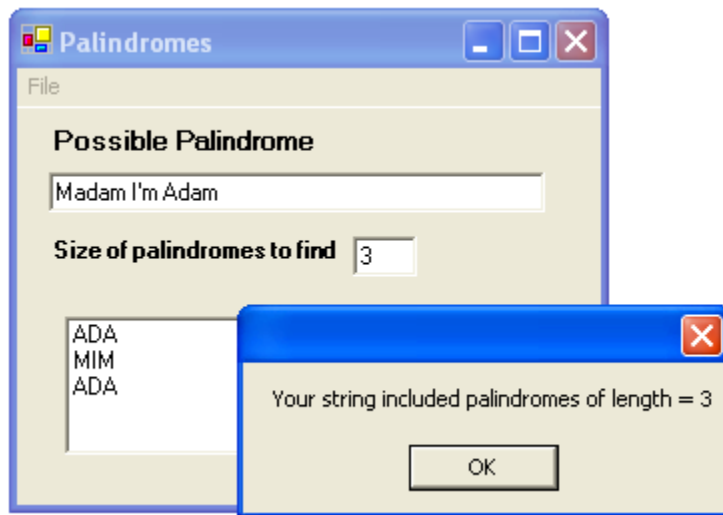
ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:

Consider creating a *bool* method to examine a string and return True if the string is a palindrome. Examine the whole string by sending the contents of the text box to this method. Search for smaller palindromes by sending a number of sub-strings to the method using a loop.

A completed program might look something like this:



PROJECT EXTRAS:

- Display a count of the number of palindromes found.

Dice Class Project Instructor Notes

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 90 minutes

OBJECTIVES:

- Create a new class

SKILLS NEEDED:

- Basic understanding of the Random class
- Knowledge of class coding syntax

MATERIALS NEEDED:

- C#

TEACHING SUGGESTIONS:

The dice class can be used in a wide variety of projects. This makes it particularly interesting and meaningful for many students. The emphasis in this project should be on the class and not what ever program is used to test it. Students may need to be reminded that the nature of a class means that it should be designed to be as simply and consistently as possible.

Explain how a non-default constructor and the use of variables in formulas can be used to allow for dice with more then just the standard six sided die. In fact, you may want to talk about how a two sided die could substitute for a coin class. If you plan to cover inheritance, the dice class could be used as a base for other selection classes if properly designed.

You may want to create your own driver program and require that all students use it to test their class. This helps prevent driver programs that avoid or compensate for flaws in the class design.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

The class should have a default constructor that creates six-sided die and a second constructor which takes an integer value to represent the number of sides on the dice. The class should have a property that returns the current value of the die. A Roll method should select a new random value for the die. This

method may be of type void and only set the value or it may be of type int and return the new value. In either case, the result should be fully commented.

If the test program is designed and written by the student it should not count for a major part of the project grade. It should however test all constructors, properties and methods of the class.

SUGGESTED SOLUTION:

The class should use three data variables:

- the number of sides of the die
- the current value of the die
- a Random variable for selecting new values

The number of sides on the die should be used in the formula for the random values. For example:

```
myValue = rnd.Next(1,mySides);
```

Dice Class Project Student Project

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 90 minutes

OBJECTIVES:

- Create a new class

OVERVIEW OF PROJECT:

A dice class can be used in many projects. This makes a dice class a good example of how classes can be used for reusable code. Create a dice class that returns a value and “rolls” to select a new value.

PROJECT INSTRUCTIONS:

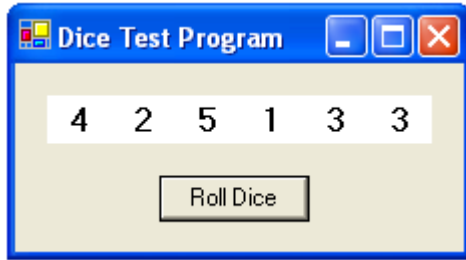
1. Open a new C# project or one provide by your instructor.
2. Add a new class file (Add Class from the Project menu) and call it DiceClass.
3. Create two class constructors. One that creates a standard six-sided die and one that allows the client program to specify the number of sides.
4. Both constructors should set the value of the die to a random value.
5. Create a property that returns the current value of the die.
6. Create a Roll method to change the value of the die and return the new value.
7. If your instructor has not provided you with a test program for your class, write a simple program that uses both constructors, the value property and the Roll method.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:

A completed program will display a number of random values and might look something like this:



PROJECT EXTRAS:

- Add a constructor that accepts a minimum and maximum value. This would allow you to have a die with values from 0 to 5, or 10 to 20 for example.
- Make the value property a write property so that a value can be set. Make sure that this does not allow for values greater than the number of sides.
- Create a more involved program to use your class.

Conway's Game of Life Instructor Notes

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 4 - 5 hours

OBJECTIVES:

- Loading and manipulating object arrays
- Evaluating complex conditions using nested *if* blocks

SKILLS NEEDED:

- Understanding of array processing
- Understanding of decision structures
- Understanding of Boolean operations

MATERIALS NEEDED:

- C#

TEACHING SUGGESTIONS:

John Conway's Game of Life was one of the first "artificial life" programs. It remains one of the most popular life simulations for computer programmers. There are many sites on the World Wide Web devoted to it. You may want to suggest that interested students do some independent research on the topic. One of the things they may find are interesting patterns for use with the game. There are also a number of shareware and freeware versions of the game already written and available on the Internet. These may serve as inspiration for more advanced or interested students.

The rules of the game (outlined in the student's section) are simple. The implementation can get involved. Students may come up with overly complicated solutions so you should review them before they get too far along in program development.

Explain the benefits of using constants for important variables to allow program growth and change. For example, by using constants to indicate the number of columns and rows the board may be expanded and contracted just by changing those values in one place. These constants are throughout the program.

Students may attempt to create all the cells needed manually. This should be discouraged as it often leads to serious problems in debugging, severely limits program expandability, and misses an opportunity to let the computer do more of the work. Having the program create and position objects in the grid does require more up front planning but that is probably more an advantage than disadvantage.

This project lends itself to the use of different functions or methods to handle parts of the process. For example, counting the number of neighbors a cell has is more easily tested if done in a function.

RESOURCES:

- Textbook

SUGGESTED EVALUATION:

A complete Life program should:

- Allow a user to select initial cells to contain “life.”
- Generate and display a new life generation.
- Allow the user to clear the board
- Allow the user to exit the game.

SUGGESTED SOLUTION:

Checks for neighbors may be implemented several ways. One way, often chosen by new programmers, is to treat corner cells, top and bottom rows, and left and right columns as special cases. This adds considerable complexity and creates many additional opportunities for errors.

Creating a sort of neutral zone around the edge of the board allows the programmer to treat all cells identically. The top and bottom rows and left and right columns are initialized as empty cells and rendered invisible (Visible property set false). This means those cells will never be changed or have life in them. The programmer can set up a simple set of If statements to check all cells.

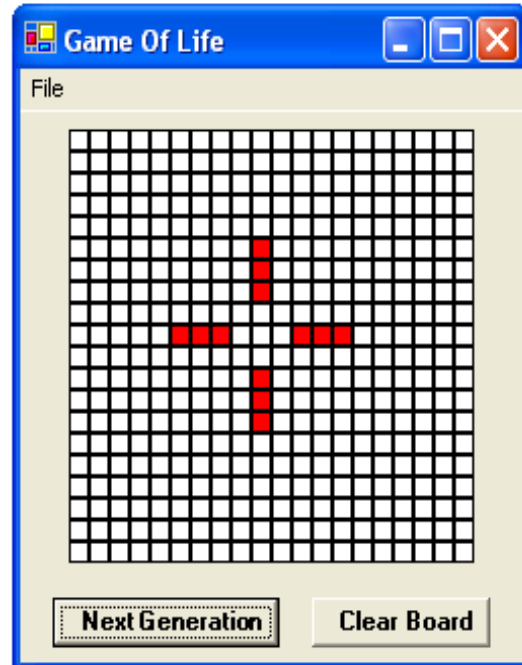
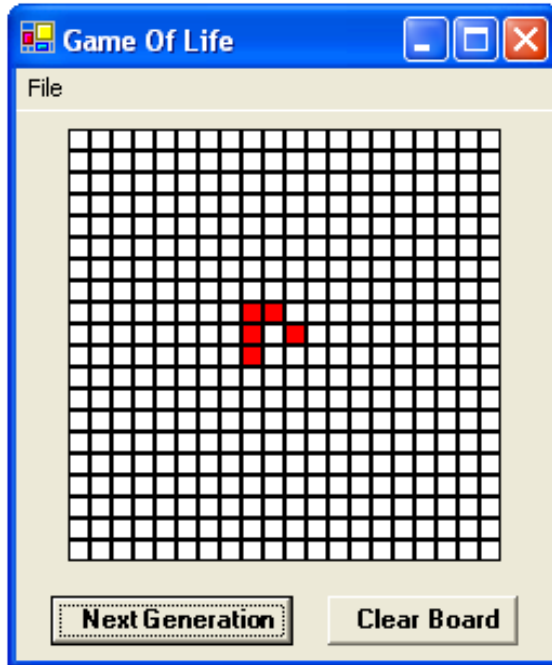
A third option is to use a try/catch structure and return a value indicating a cell is empty if an exception is generated when an attempt to read a cell outside the range of the array. This uses the built-in exception handling that is part of all C# arrays.

Once the number of neighbors has been determined, the status of the current cell is easily determined.

Students should use an array separate from the array holding the current state of the board to hold the state for the next generation. Not using a separate array allows any changes made to affect the evaluation of other cells.

When a user sets or clears the life in a cell, the programmer must be careful to change both the visible indication of life (background color or picture) and the internal indication (object tag value, status array, etc.) to maintain a consistent state. Failure to do this results in undetermined results that may be hard to debug.

Here are a few interesting shapes to start. Students and instructor alike will easily discover others.



Conway's Game of Life Student Project

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 4 - 5 hours

OBJECTIVES:

- Loading and manipulating object arrays
- Evaluating complex conditions using nested *if* blocks

OVERVIEW OF PROJECT:

John Conway's Game of Life was one of the first "artificial life" programs. It remains one of the most popular life simulations for computer programmers.

The game is played on a collection of cells. Each cell has eight neighboring cells. Each cell either is occupied by a "life" or is empty. The state of a cell for a new generation is determined by a few simple rules.

1. If a life has no neighbors or only one, it dies of loneliness.
2. If a life has four or more neighbors, it dies of overcrowding.
3. If a life has either two or three neighbors, it survives to the next generation.
4. If a cell has no life in it but has exactly three neighbors, a new life is born.

Create a program to allow a user to set up a game board with initial life values and generate new generations of the board.

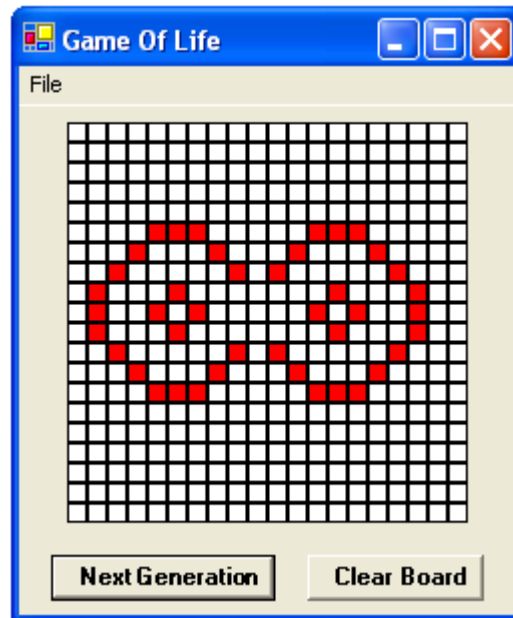
PROJECT INSTRUCTIONS:

1. Create an object to use to display a "life."
2. Create a board by loading new copies of the initial object in grid form.
3. Write code using the rules in the overview to determine which live cells "die" or "survive".
4. Write code using the rules in the overview to determine which cells have lives "born" into them.
5. Create an option to generate a new life generation using the code that determines life status.
6. Create a clear option with the appropriate code to empty all cells.
7. Create an exit option with the appropriate code to end the program.

ADDITIONAL RESOURCES:

- Textbook

SUGGESTED SOLUTION:



PROJECT EXTRAS:

- Add a button to have the program automatically generate and display several generations.
- Use different colors for new life and life that has "survived" from the previous generation.
- Allow the user to select a color for the life or background colors.
- Use files to save, load, and edit interesting patterns.

Checkers Game Program Instructor Notes

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 6-8 hours

OBJECTIVES:

- Use complex and nested decision constructs
- Understanding of graphics methods
- Understanding of event handlers

SKILLS NEEDED:

- Understanding of looping constructs
- Understanding of variable and object arrays

MATERIALS NEEDED:

- C#

TEACHING SUGGESTIONS:

The game of checkers is familiar to most students. They will find programming the rules of checkers surprisingly complex and it may be useful to use a simplified set of rules if time is a problem. Have students write out the rules they will be using. Make sure they have them all listed before they start designing their program. A sample list follows.

1. Pieces at the top of the board may only move downward.
2. Pieces at the bottom of the board may only move upward.
3. Pieces may only move into empty squares.
4. Pieces may only move diagonally.
5. Pieces must move to an adjacent square or by jumping an opponent's piece to the next square past that piece on the same diagonal.
6. Remove jumped pieces from the board.

Encourage students to have the computer duplicate an original square object. The same loop that creates the board can be used to set the initial values for tracking piece locations. Students must give careful consideration to how they will check for valid moves. This is especially true for moves involving the top and bottom rows of the board. Encourage them to desk check their algorithms before implementation.

Review with students the advantages of nesting If statements. If they don't nest decision constructs, they will find that they need a more complex set of flags and indicators for this program.

Suggest to students that they provide some indication to the user of what piece they have selected for movement. They should also allow the user to deselect a piece if it has not been moved.

If students will be drawing discs on picture boxes using the FillCircle method, explain the paint event. The paint event occurs when part or all of an object is exposed after being moved or enlarged. A paint event also occurs after a window that was covering the object has been moved. If a Paint event handler is not written, students may find parts of their display missing if the form is covered or minimized at any time.

RESOURCES:

- Textbook
- Checker board and pieces to demonstrate the game and its rules

SUGGESTED EVALUATION:

Completed programs must follow all the rules listed in the teaching suggestions section. Be sure to test jumps and moves in a number of directions. Attempt to make moves into and out of the end rows.

SUGGESTED SOLUTION:

The first step is the creation of the playing board. Create one square using a picture box. Set the initial properties so that they can be copied into dynamically loaded objects. Set the border style to none so that boxes will appear seamlessly next to one another.

To keep track of what boxes have what pieces on them, create an integer array. Create 63 more elements in the board square array and move them into position. As squares are moved into position, change the BackColor of alternate squares in each row. Set the array holding contents of squares in odd numbered squares in the first 24 squares to indicate the first color. Set the odd numbered squares in the last 24 squares to hold the other color. Flag odd numbered squares in the middle of the board as empty.

Writing the code that colors the squares and initializes the board-tracking array as a method separate from the routine that loads the full object array. This allows that method to be called as a "New Game" routine.

Drawn objects in picture boxes may disappear if the board is hidden by another window. Write a Paint event handler to redraw the picture boxes on the board. This method will examine the flag for a square and draw the appropriate disk in occupied squares. These event handlers will be called automatically when the form is first loaded.

Draw disks on a picture box using the FillCircle method. Create a SolidBrush object and set it to the appropriate color for the piece being drawn. Colors for the different playing pieces. The PictureBox properties of Height and Width can be used to specify the rectangle for the playing piece to be drawn.

Clicking on a square starts an important and involved set of events. Create a form class variable to hold a flag indicating if a piece has been selected for movement. Determining if a valid square has been clicked on requires a number of checks.

1. If a piece has already been selected, clicking on a square with a piece on it is not valid.
2. If a piece has not been selected, clicking of a square that is empty or has a piece of the wrong color, that is not a valid click.
3. If a piece has been selected and the square clicked on is flagged as an invalid square, not on the proper set of diagonals, that is not a valid click.
4. If a piece has already been selected and an empty square on the proper set of diagonals has been clicked check the following cases.
5. Is the square adjacent and in the right direction?
6. If the square is not adjacent, is there a piece owned by the opponent between the square the piece is on and the square selected?

Once a valid move has been selected, move the piece appropriately. Remove the disk from its original location by clearing the box and re-setting the flag in the board array. Draw a new disk in the new square and set the board array to indicate what color disk occupies it. If an opponent's piece was jumped, clear the square the piece was on and refresh the picture box to indicate that square is now empty.

Switch a current player flag after each valid move. Turn off the piece selected flag if a player clicks on the currently selected piece to allow a player to deselect a piece. One way to indicate the currently selected piece is to redraw the disk a different, but related, color. Deselecting a piece implies that it must be redrawn in its primary color after it has been deselected.

This program serves easily as a base for writing a more complete checkers program.

Checkers Game Program Student Project

ABILITY LEVEL: Advanced

APPROXIMATE COMPLETION TIME: 6-8 hours

OBJECTIVES:

- Use complex and nested decision constructs
- Understanding of graphics methods
- Understanding of event handlers

OVERVIEW OF PROJECT:

Create a program that allows two players to play a simple form of checkers. Draw a 64 square board with alternating color squares. Place different colored discs on the dark squares in the top and bottom three rows. Allow the user to select a disc and move it to diagonally adjacent squares or to “jump” opponent’s discs on diagonally adjacent squares. Discs may only move to empty squares. Jumps may only be made over opponent’s discs. A piece (disc) that is jumped must be removed from the board. The program should refresh its display if the board is resized or temporarily hidden and unhidden.

Allow the user to reset the board. Resetting the board puts all pieces back on their original squares.

Allow the user to quit the game and exit the program.

PROJECT INSTRUCTIONS:

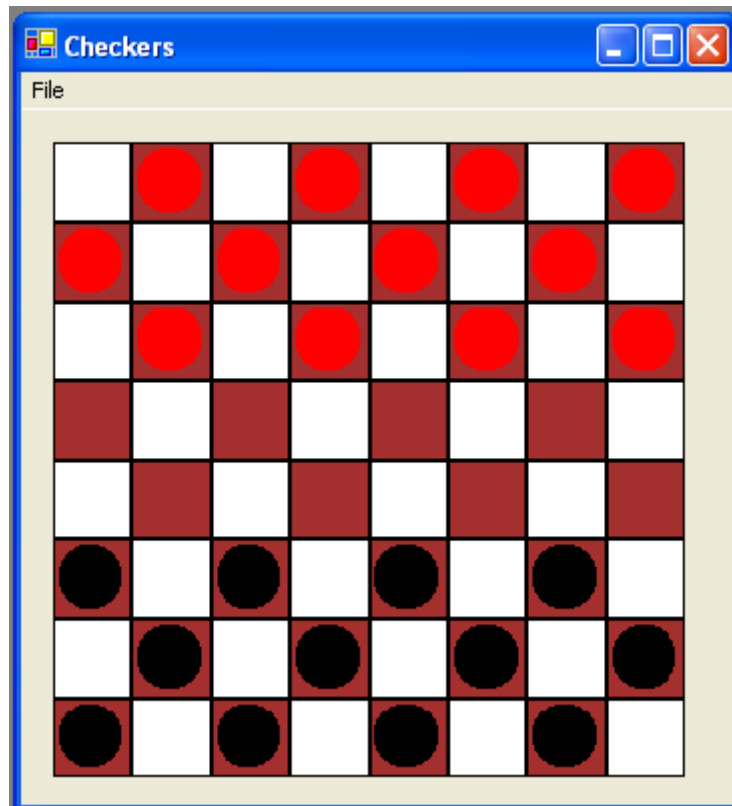
1. Write out a list of rules the program will enforce on piece movement.
2. Create an 8 by 8 board consisting of squares of different background colors.
3. Create a subroutine to populate the board with different color pieces as with a game of checkers.
4. Create code behind the object array representing the board squares to allow the user to select a playing piece.
5. Create code to allow the user to indicate a location to move a selected piece.
6. Create code to verify that a target square is a valid move according to the rules of the game.
7. If an opponent’s piece is jumped by a player’s move, remove that piece from the board and identify that pieces square as empty.
8. Create an option for the game to be reset to its initial set-up.
9. Write program code to redraw pieces on the board if a paint event occurs.
10. Allow the user to exit the program.

ADDITIONAL RESOURCES:

- Textbook
- Checker board and pieces to desk check algorithms

SUGGESTED SOLUTION:

A completed program might look something like this:



PROJECT EXTRAS:

- Allow the users to select the colors of:
 - The discs used
 - The squares on the board
 - The form background
- Determine the end of the game when there are no remaining moves.
- Keep a running count of how many discs each player has on the board.
- Make a more complete game of checkers by allowing pieces that reach the end of the board to become "kings" and move backwards. Provide an identification of kings on discs as appropriate.