Projects for
Murach's C# (8th Edition)

The projects in this document let your students apply the programming skills that they learn as they progress through *Murach's C#*. Unlike the exercises in the book and the extra exercises in the instructor's materials, which lead your students step by step, the projects describe the operation of an app and present the specifications for the app. Then, your students must decide how to approach the development of the app.

If you review these projects, you'll see that they represent different degrees of difficulty, but you can easily modify them to make them easier or more challenging. Since most of these projects can be done from scratch in an hour or two, you can also use these projects as tests.

An introduction to the projects

The following topics give you some ideas about how you can use the projects presented in this document.

When to assign the projects

To give you an idea of when each project can be assigned, the projects are numbered by section. In addition, the description for each project indicates what chapters your students must complete before they can do that project. For example, your students will have the skills they need to complete project 2-1 after they've read chapters 1 through 9. Keep in mind, however, that not all of the chapters have to be read in sequence. After reading chapters 1 through 12, for example, you may choose to go on to the chapters in section 4 or 5. Because of that, the projects for sections 4 and 5 require only the skills presented in chapters 1 through 12.

Which projects to assign

This document includes one or more projects for each of sections 2 through 5 of the book. Because we feel that doing the exercises is the best way for your students to learn the skills in each chapter, you'll probably want to assign only one or two projects for each section. Then, if your students can do these projects successfully, you can be sure that they have mastered the skills they need to develop C# apps.

About the instructor notes

After the specifications for most of the projects, you'll find one or more instructor notes. Some of these notes describe the skills that are needed for the projects. Others present ideas for how you might modify the projects to make them easier or more difficult. And still others indicate how you might have your students modify a project after reading a later chapter. *Before you distribute the project descriptions to your students,* you will probably want to delete or modify these notes.

Labels vs. read-only text boxes

Unlike the apps presented in the book, which use read-only text boxes to display the results of calculations, most of the projects use labels. If you want the projects to adhere more closely to the apps in the book, however, you can instruct your students to use read-only text boxes instead of labels. Otherwise, information about how to get the labels to look and work like the ones in the projects is given in project 2-2.

About the TechSupport database

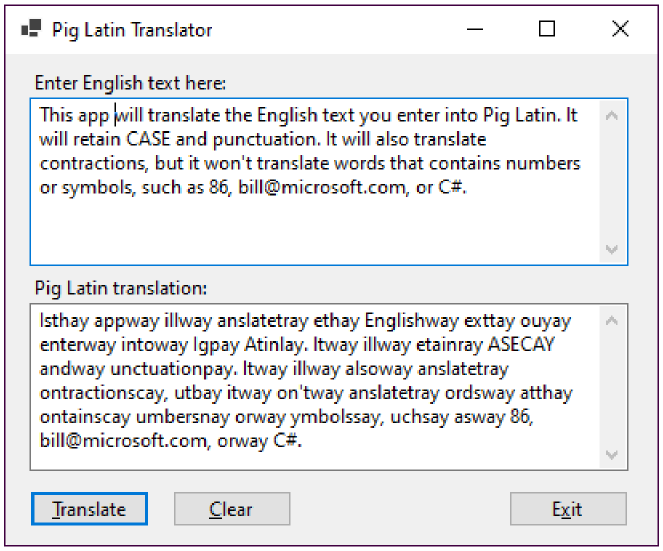
The projects for section 5 require the use of a SQL Server database named TechSupport. If you will be installing this database on your lab system, you just need to give your students the information they need for connecting to the database. However, if your students are going to use SQL Server Express LocalDB on their own systems, you need to distribute the TechSupport.mdf and TechSupport_log.ldf files in the Projects/Database folder to your students.

Section 2 projects

Project 2-1 Translate English to Pig Latin

For this project, you'll create a form that accepts a text entry from the user and converts the text to Pig Latin. Prerequisites: chapters 1-9.

The Pig Latin Translator form



This app will translate the English text you enter into Pig Latin. It will retain CASE and punctuation. It will also translate contractions, but it won't translate words that contains numbers or symbols, such as 86, bill@microsoft.com, or C#.

Isthay appway illway anslatetray ethay Englishway exttay ouyay enterway intoway Igpay Atinlay. Itway illway etainray ASECAY andway unctuationpay. Itway illway alsoway anslatetray ontractionscay, utbay itway on'tway anslatetray ordsway atthay ontainscay umbersnay orway ymbolssay, uchsay asway 86, bill@microsoft.com, orway C#.

Operation

- The user enters text in the first multi-line text box and clicks the Translate button or presses the Enter key.
- The app translates the text to Pig Latin and displays it in the second multi-line text box.
- To clear both text boxes, the user clicks the Clear button.

Specifications

- If a word starts with a vowel, just add *way* to the end of the word.
- If a word starts with a consonant, move the consonants before the first vowel to the end of the word and add *ay*.
- If a word starts with the letter Y, the Y should be treated as a consonant. If the Y appears anywhere else in the word, it should be treated as a vowel.
- Keep the case of the original word whether it's uppercase (TEST), title case (Test), or lowercase (test).
- Keep all punctuation at the end of the translated word.
- Translate words with contractions. For example, *can't* should be *an'tcay*.
- Don't translate words that contain numbers or symbols. For example, 123 should be left as 123, and bill@microsoft.com should be left as bill@microsoft.com.
- Check that the user has entered text before performing the translation.

Hints

- To create a multi-line text box, just set the Multiline property of the text box to True and size the text box accordingly.
- To add vertical scroll bars to the text box, set the ScrollBars property to Vertical.
- The integer values for all uppercase letters are 65 to 90, and the integer values for all lowercase letters are 97 to 122.

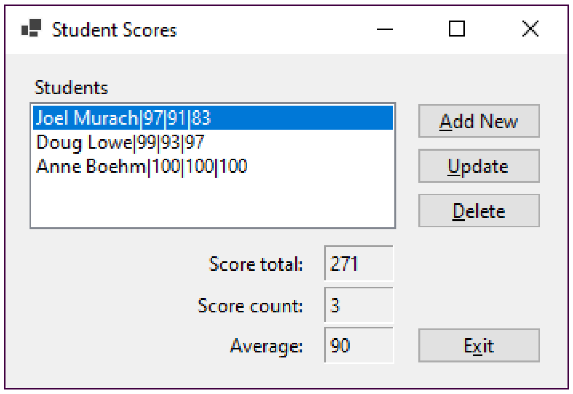
Instructor notes

- This is not a trivial assignment. Students will have to use string handling to parse the input string into separate words. Then, string handling will have to be used again to analyze letters at the beginning of each word, to identify consonants, vowels, numerals, and punctuation marks, and to add Pig Latin word endings.
- There are no official rules for Pig Latin. Most people agree on how words that begin with consonants are translated, but there are many different ways to handle words that begin with vowels. To enhance this app, you can have your students add radio buttons that provide for selecting one of several translations. To find out more about different Pig Latin translations, you can search the internet.
- Project 3-6 builds on this project by adding the ability to translate to another dialect that's similar to Pig Latin. That project uses classes and an interface.

Project 2-2 Maintain student scores

For this project, you'll develop an app that lets the user add students to a list, change the scores for a student in the list, and delete a student from the list. Prerequisites: chapters 1-10.

The Student Scores form



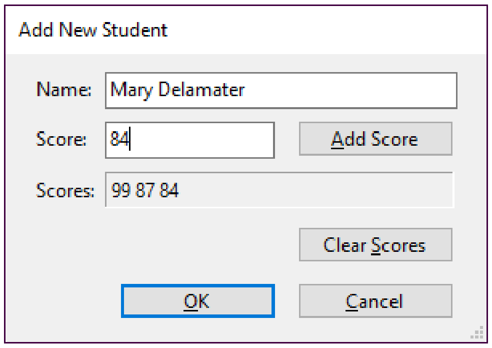
Joel Murach|97|91|83
Doug Lowe|99|93|97
Anne Boehm|100|100|100

Score total:	271
Score count:	3
Average:	90

Operation

- To display the total, count, and average for a student, the user selects the student from the list box. If the list box is empty, the total, count, and average labels should be cleared.
- To add a new student, the user clicks the Add New button to display the Add New Student dialog.
- To update an existing student's scores, the user selects the student in the list box and clicks the Update button to display the Update Student Scores dialog.
- To delete a student, the user selects the student in the list box and clicks the Delete button.

The Add New Student form



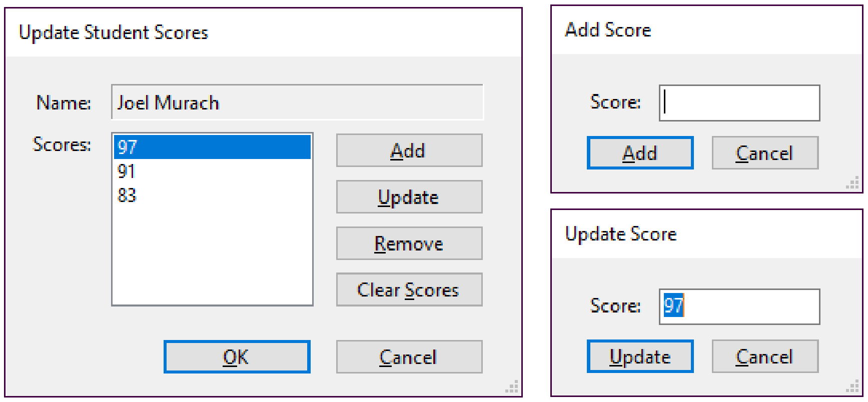
Name:	Mary Delamater
Score:	84
Scores:	99 87 84

Operation

- To add a new student, the user enters a student name and, optionally, one or more scores and clicks the OK button.
- To add a score, the user enters a score and clicks the Add Score button. The score is added to the list of scores in the Scores label.
- To remove all scores from the Scores label, the user clicks the Clear Scores button.

- To cancel the add operation, the user clicks the Cancel button.

The Update Student Scores and Add/Update Score forms


Update Student Scores

Name: Joel Murach

Scores: 97, 91, 83

Add Score

Score:

Update Score

Score: 97

Operation

- To add a score, the user clicks the Add button and enters the score in the Add Score dialog that's displayed.
- To update a score, the user selects the score, clicks the Update button, and changes the score in the Update Score dialog that's displayed.
- To remove a score from the Scores list box, the user selects the score and clicks the Remove button.
- To remove all scores from the Scores list box, the user clicks the Clear Scores button.
- To accept all changes, the user clicks the OK button.
- To cancel the update operation, the user clicks the Cancel button.

Specifications

- This app should make sure that the user enters a name for a new student. However, a new student can be added without any scores.
- This app should check all scores entered by the user to make sure that each score is a valid integer from 0 to 100.
- When the app starts, it should load the list box with three sample students.

Hint

- To get the labels that display the score total, score count, and average on the Student Scores form, the scores on the Add New Student form, and the name on the Update Student Scores form to look like the ones shown above, you'll need to set the AutoSize property of the labels to False and the BorderStyle property to Fixed3D.

Instructor notes

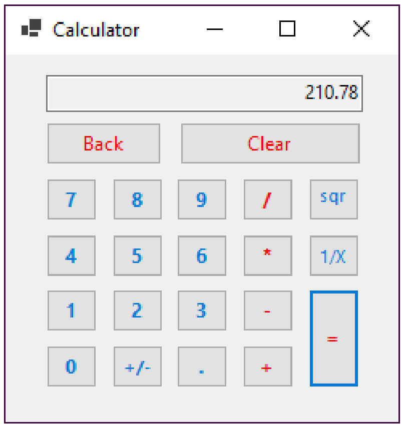
- This project requires the use of lists, list boxes, and some string handling. Although it uses four forms, it is still a manageable project.
- After your students complete chapters 12, 14, and 15, you can have them enhance this app by adding a Student class. See the description for project 3-5 for details.
- After your students complete chapter 17, you can have them enhance this app by maintaining the student data in a text file. See the description for project 4-1 for details.

Section 3 projects

Project 3-1 Create a basic calculator

For this project, you'll create a form that lets the user perform the operations provided by a basic calculator. You'll also create a class that performs the required operations. Prerequisites: chapters 1-12.

The Calculator form



Operation

- To perform an addition, subtraction, multiplication, or division operation, the user clicks the first number, followed by the appropriate operator key (+, –, *, /), followed by the second number and the equals key (=).
- To perform an addition, subtraction, multiplication, or division operation on the result of a previous operation, the user clicks another operator key, followed by another number and the equals key. The user can also repeat the previous operation on the result by clicking the equals keys without first clicking another operator and number.
- To perform a square root or reciprocal operation, or to change the sign of a number, the user clicks the number followed by the appropriate operator key (sqrt, 1/X, +/-).
- To perform a square root or reciprocal operation on the result of a previous operation, the user clicks the appropriate operator key.
- Each time the user clicks a number key, the number is displayed in the text box at the top of the form. This text box also displays the result of an operation when the user clicks the sqrt, 1/X, +/-, or = key.
- To erase the last digit entered, the user clicks the Back key.
- To clear all the values entered, the user clicks the Clear key.

Specifications

- Create a class named Calculator that implements the functions of the calculator. Design whatever methods and properties you need for this class.
- If the user tries to divide a number by zero, the calculator should display an error message in the text box. The form class should use a try-catch statement to catch a divide-by-zero exception.

Instructor notes

- The calculator that's created in this project is similar to the calculator that comes with Windows. If you want to, you can have your students study the behavior of the Windows calculator so they have a better idea of how it works.
- One of the differences between the calculator created in this project and the Windows calculator is that you can perform a calculation by clicking an operator key other than the = key with the Windows calculator. In other words, you can add a list of numbers like this using the Windows calculator: 3 + 12 + 14 + 8 =. With the calculator for this project, you have to perform each calculation separately by clicking the = key like this: 3 + 12 = + 14 = + 8 =. If you want to, you can have your students enhance this calculator so it works like the Windows calculator. Because it is difficult to maintain the state of the calculator, however, this will be a complicated task.
- You can also have your students enhance this project by adding a button that clears the last entry, but not the previously calculated value. Again, because of the difficulty maintaining the state of the calculator, this will be complicated.
- Another enhancement would be for the app to provide for entries from the keyboard. For example, if the user presses the 1 key, the number 1 should be displayed in the calculator. Similarly, if the user presses the = key, the current calculation should be performed. Note, however, that not all of the calculator buttons map to keyboard keys.
- To simplify this project, you can eliminate the square root and reciprocal functions.
- If you want to be more specific about the Calculator class, you can provide this class design:

Private field	Description
`displayString`	A string that stores the value that's currently displayed by the calculator.
`isNewValue`	A Boolean that controls numeric entry.
`hasDecimal`	A Boolean that tracks whether the display string contains a decimal point.
`operand1`	A decimal that stores the value of the first operand.
`operand2`	A decimal that stores the value of the second operand.
`op`	An Operation type that stores a member of the Operation enumeration.
`Operation`	An enumeration with these constants: Add, Subtract, Multiply, Divide, SquareRoot, Reciprocal, and None.
Constructor	**Description**
`()`	Creates a Calculator object with default values. The default value for the op field is Operation.None. The default value for the isNewValue field is true.
Property	**Description**
`DisplayString`	Gets the value of the displayString field, or "0" if it's null.
`displayValue`	Gets the value of the DisplayString property, converted to a decimal type. A private property that's only used in the class.

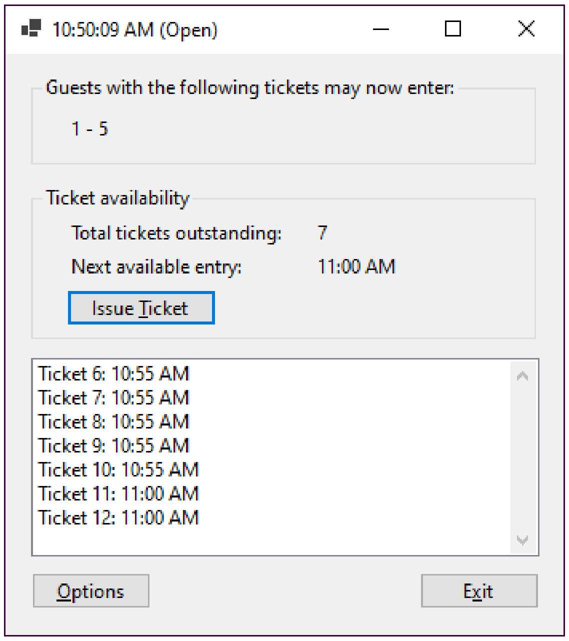
Method	Description
`Append(string)`	Adds the string value passed to it to the end of the displayString field.
`RemoveLast()`	Removes the last character from the displayString field.
`ToggleSign()`	Switches the display value from positive to negative or vice versa.
`AddDecimalPoint()`	Adds a decimal point to the displayString field, and ensures there can only be one decimal point.
`Add()`	Sets the op field to Operation.Add, the operand1 field to the value of the displayValue property, the isNewValue field to true, and the hasDecimal field to false.
`Subtract()`	Sets the op field to Operation.Subtract, the operand1 field to the value of the displayValue property, the isNewValue field to true, and the hasDecimal field to false.
`Multiply()`	Sets the op field to Operation.Multiply, the operand1 field to the value of the displayValue property, the isNewValue field to true, and the hasDecimal field to false.
`Divide()`	Sets the op field to Operation.Divide, the operand1 field to the value of the displayValue property, the isNewValue field to true, and the hasDecimal field to false.
`Equals()`	Sets the operand2 field to the value of the displayValue property. Then, performs the operation specified by the op field on the operand1 and operand2 fields, and stores the result in the displayString field. Sets the isNewValue field to true, and the hasDecimal field to false.
`SquareRoot()`	Sets the op field to Operation.SquareRoot, the operand1 field to the value of the displayValue property, the isNewValue field to true, and the hasDecimal field to false. Calculates the square root of the operand1 value and stores it in the displayString field.
`Reciprocal()`	Sets the op field to Operation.Reciprocal, the operand1 field to the value of the displayValue property, the isNewValue field to true, and the hasDecimal field to false. Calculates the reciprocal of the operand1 value and stores it in the currentValue field.
`Clear()`	Sets the private fields to their default values.

- After your students complete chapter 14, you can have them enhance this project by adding a memory function. To implement this function, your students can create a new class named MemoryCalculator that inherits the Calculator class. See project 3-4 for details.

Project 3-2 Assign tickets with time slots

For this project, you'll develop an app that assigns tickets that include a time slot when a guest can return to visit an attraction without waiting in line. Prerequisites: chapters 1-12.

The Tickets form



10:50:09 AM (Open)

Guests with the following tickets may now enter:

1 - 5

Ticket availability

Total tickets outstanding: 7

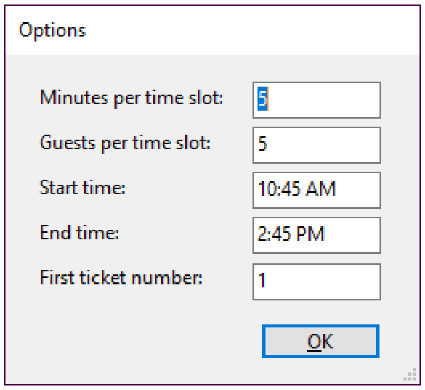
Next available entry: 11:00 AM

```
Ticket 6: 10:55 AM
Ticket 7: 10:55 AM
Ticket 8: 10:55 AM
Ticket 9: 10:55 AM
Ticket 10: 10:55 AM
Ticket 11: 11:00 AM
Ticket 12: 11:00 AM
```

Operation

- To issue a ticket, the user clicks the Issue Ticket button. A ticket with the next ticket number in the next available time slot is issued.
- Each time a ticket is issued, it is added to the list box.
- The title bar for this form displays the current time and an indication of whether the current time slot is open or closed. This information is updated once per second.
- The labels in the Ticket Availability section of the form indicate how many tickets are outstanding and the time slot that will be assigned to the next ticket that's issued. These labels are updated as tickets are issued and the time changes.
- When a time slot begins, any outstanding tickets for that time slot are removed from the list box and the beginning and ending ticket numbers assigned to that slot are displayed in the label inside the group box at the top of the form.
- To change the options for issuing tickets, the user clicks the Options button. Before the Options dialog is displayed, a dialog is displayed that warns the user that all outstanding tickets will be deleted and confirms that the user wants to continue.

The Options form



Minutes per time slot:	5
Guests per time slot:	5
Start time:	10:45 AM
End time:	2:45 PM
First ticket number:	1

Operation

- The user can enter values into the Options dialog to specify the number of minutes for each time slot, the number of guests allowed into the attraction during each time slot, the time the attraction opens, the time the attraction closes, and the number for the first ticket. The defaults are five minutes per time slot, five guests per time slot, a start time of the current time, an end time of four hours after the current time, and an initial ticket number of 1.
- When the user clicks the OK button, the Tickets form is displayed.

Specifications

- Use a Timer control to display the current time in the title bar of the main form and to determine the current time slot.
- When the app starts, it should display the Options dialog from the Load event handler for the Tickets form. For this to work without an exception being thrown, you'll need to disable the Timer control until the Options dialog is completed.
- The data the user enters in the Options dialog should be validated to be sure that the minutes per time slot, guests per time slot, and first ticket number are integers; that the start and end times are DateTime values; and that the difference between the start time and the end time provides for at least two time slots.
- Create a class that represents a time slot for assigning tickets. This class should have public properties that indicate the time the time slot begins, the length of the time slot, and the number of tickets that have been issued for the time slot. This class should also have a property that indicates whether the time slot is full. Use this class to create a time slot object for each time slot between the start and end times entered on the Options dialog, and store these objects in a collection.
- Create a class that represents a ticket. This class should have public properties that store the ticket number and the time slot that's assigned to the ticket, as well as a property that displays the ticket number and time slot as shown in the list box on the Tickets form. As each ticket is issued, the app should create a ticket object with the next ticket number and add it to a queue. Then, when the time slot for the ticket begins, the ticket should be removed from the queue.
- Once a time slot begins and the tickets in that time slot are removed from the outstanding list of tickets, the user should not be able to issue any more tickets in that time slot even if the maximum number of tickets for that time slot haven't been issued.

Instructor notes

This project uses several custom objects, including custom objects that have properties that are other custom objects. It also has extensive collection and date/time requirements. Finally, it requires that the student research and use the Timer control, which isn't presented in the book.

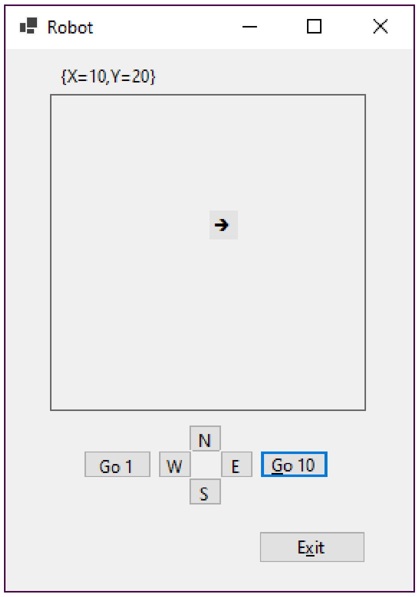
The complexity of the project can be varied as follows:

- Implement just the ticket assignment portion of the app, with no real-time monitoring of the time.
- Issue a ticket for the current time slot if tickets are still available.
- Instead of issuing a ticket for the next available time slot, let the user choose the time slot for the ticket. Then, the tickets in the current time slot will have to be listed individually at the top of the form. Depending on how the student designs the data store, this may result in a major redesign of the app.
- Use NumericUpDown controls to select the minutes per window and guests per window from the Options dialog, and use the DateTimePicker control to select the start and end times. This will require some additional research by the student.

Project 3-3 Direct a simple robot

For this project, you'll create a form that lets the user move a simple robot a given direction and distance. You'll also create a class that performs the robot movements. Prerequisites: chapters 1-13.

The Robot form



Operation

- To determine the direction the robot will move, the user clicks the N (North), S (South), E (East), or W (West) button.
- The robot is displayed as an arrow that points in the currently selected direction.
- The user can move the robot 1 or 10 units in the selected direction by clicking the Go 1 or Go 10 button.
- The robot's current X, Y position is displayed in a label at the top of the form.

Specifications

- Create a class named Robot that has a method that causes the robot to move, a property that contains the current direction, and a property that returns the robot's current position as a Point structure. When the robot is instantiated, the position should be set to 0, 0 (the center) and the direction should be set to North.
- Limit the range of the robot to 100 units in any direction. If the user attempts to move the robot beyond this range, the Robot class should raise an event. The app should respond to this event by displaying an appropriate message.

Hints

- The easiest way to create the moving arrow is to use a label control. Then, the program can adjust the control's Position property to move the arrow within a panel control. If you need to, you can use online help to research this control.
- To create the arrow, set the label's Font property to the Wingdings font. Then, you can use the integer values 231 through 234 to set the label to the appropriate arrow.

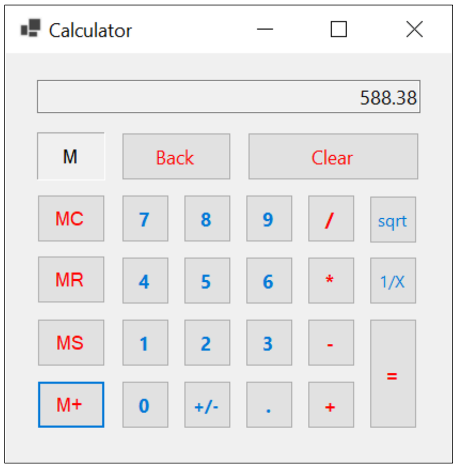
Instructor notes

- This project requires a class that performs linear robot movements. The class includes an event that is fired when the robot reaches a boundary. It also uses an enumeration and the Point structure.
- To simplify this project, you can eliminate the graphic display of the robot's position and just display the robot's X, Y coordinates in a label.
- If your students have a math background, you can enhance this project so it provides for nonlinear directions of travel. To do that, your students will need to use trig functions to calculate the robot's position.
- You might also enhance this project to use inheritance. For example, a subclass could inherit the basic Robot class and add some functionality, such as recording the movement of the robot and calculating the distance traveled.

Project 3-4 Create a memory calculator

For this project, you'll create a form that lets the user perform the functions provided by a memory calculator. This project builds on the calculator described in project 3-1. If you have already done project 3-1, you can modify it so it works as described here.
Prerequisites: chapters 1-12, 14.

The Calculator form



Operation

- To clear the contents of memory, the user clicks the MC button. To save the value that's currently displayed in memory, the user clicks the MS button. To recall the value that's currently in memory and display it in the calculator, the user clicks the MR button. And to add the value that's currently displayed to the value in memory, the user clicks the M+ button.
- An M is displayed in the box above the MC button when the memory contains a value.
- See project 3-1 for additional details.

Specifications

- Create a class named MemoryCalculator that inherits the Calculator class described in project 3-1. The MemoryCalculator class should add properties and methods as needed to implement the memory function. The private fields of the Calculator class should be made available to the MemoryCalculator class as needed.

Instructor note

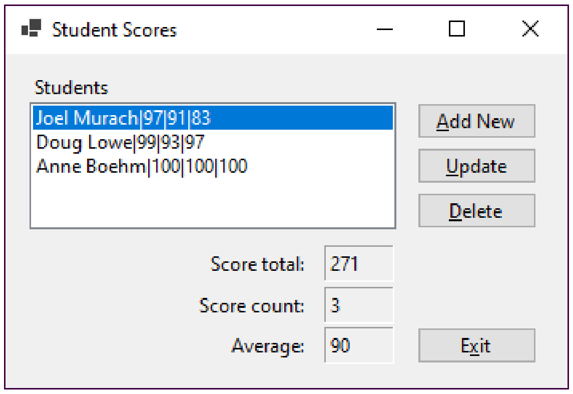
- If you want to be more specific about the MemoryCalculator class, you can provide this class design:

Private field	Description
`memoryValue`	A decimal that stores the current memory value.
Method	**Description**
`MemoryStore()`	Stores the calculator's current value in memory.
`MemoryRecall()`	Sets the calculator's current value to the value stored in memory.
`MemoryAdd()`	Adds the calculator's current value to the value currently stored in memory.
`MemoryClear()`	Clears the current memory value.

Project 3-5 Maintain student scores (Student class)

For this project, you'll develop an app like the one described in project 2-2 that maintains a list of student scores, but that uses a Student class. If you have already done project 2-2, you can use it as a starting point for this project. Prerequisites: chapters 1-12, 14, 15.

The Student Scores form



```
Joel Murach|97|91|83
Doug Lowe|99|93|97
Anne Boehm|100|100|100
```

Score total:	271
Score count:	3
Average:	90

Operation

- See project 2-2.

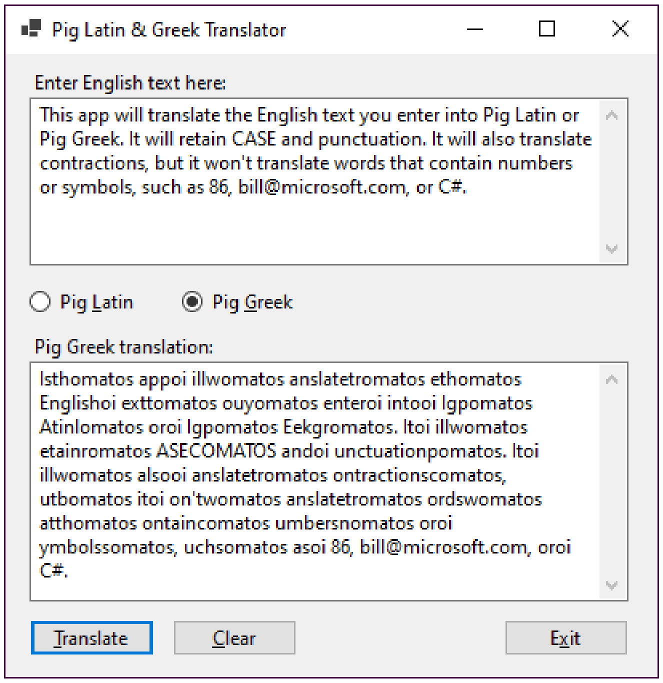
Specifications

- This app should use a class named Student to store information about each student.
- The Student class should override the ToString() method of the Object class to provide a string representation of each Student object that includes the student name and scores as shown above.
- The Update Student Scores form should create a clone of the current student object and then apply changes to the clone. That way, the changes will be saved to the current student only if the user clicks the OK button. To create a clone, the Student class will need to implement the ICloneable interface, and the Clone() method will need to implement a deep copy.

Project 3-6 Translate English to Pig Latin or Pig Greek

For this project, you'll create a form that accepts a text entry from the user and converts it to either Pig Latin or Pig Greek. This project builds on the Pig Latin translator described in project 2-1. If you have already done project 2-1, you can use it as a starting point for this project. Prerequisites: chapters 1-12, 15.

The Pig Latin & Greek Translator form



This app will translate the English text you enter into Pig Latin or Pig Greek. It will retain CASE and punctuation. It will also translate contractions, but it won't translate words that contain numbers or symbols, such as 86, bill@microsoft.com, or C#.

Isthomatos appoi illwomatos anslatetromatos ethomatos Englishoi exttomatos ouyomatos enteroi intooi Igpomatos Atinlomatos oroi Igpomatos Eekgromatos. Itoi illwomatos etainromatos ASECOMATOS andoi unctuationpomatos. Itoi illwomatos alsooi anslatetromatos ontractionscomatos, utbomatos itoi on'twomatos anslatetromatos ordswomatos atthomatos ontaincomatos umbersnomatos oroi ymbolssomatos, uchsomatos asoi 86, bill@microsoft.com, oroi C#.

Operation

- The user enters text in the first multi-line text box, selects whether to translate the text into Pig Latin or Pig Greek, and clicks the Translate button or presses the Enter key.
- The app translates the text to Pig Latin or Pig Greek and displays it in the second multi-line text box.
- When the user selects the Pig Latin or Pig Greek option, the label above the second text box changes accordingly, and the text box is cleared.

Specifications

- Create an interface named ITranslator that defines a method named Translate. This method should accept a string parameter and return a string result.
- Create an abstract base class that implements the ITranslator interface and contains methods that handle those parts of the translation task that are common to both dialects, such as parsing the string into separate words.
- Create two classes named PigLatin and PigGreek that inherit the base class and provide the appropriate translation for the dialect.

- Translate the text to Pig Greek using the specifications given in project 2-1 for translating text to Pig Latin except:

  If a word starts with a vowel, just add *oi* to the end of the word.

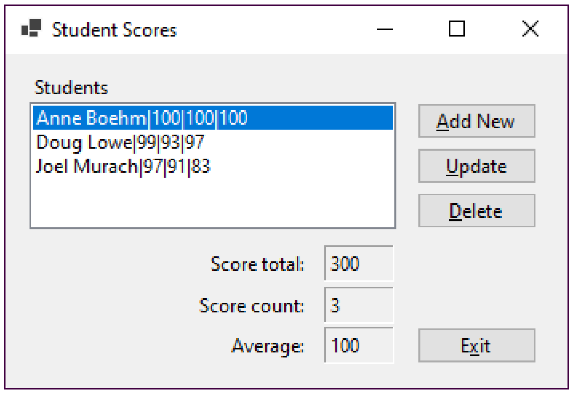
  If a word starts with a consonant, move the consonants before the first vowel to the end of the word and add *omatos*.

Section 4 projects

Project 4-1 Maintain student scores

For this project, you'll develop an app like the ones described in projects 2-2 and 3-5 that let the user maintain a list of student scores. For this project, though, you'll use a text file to store the student data. If you have already done project 2-2 or 3-5, you can use it as a starting point for this project. Prerequisites: chapters 1-12, 17.

The Student Scores form



Anne Boehm|100|100|100
Doug Lowe|99|93|97
Joel Murach|97|91|83

Score total: 300
Score count: 3
Average: 100

Operation

- See project 2-2.

Specifications

- When this app starts, it should read student data from a text file named StudentScores.txt in the C:\C#\Files directory and display this data in the Students list box. If this directory or file doesn't exist, it should be created.
- When students are added, updated, or deleted, this app should write the student data to the StudentScores.txt file, overwriting any existing data.
- This app should use a class named StudentDB to read data from and write data to the StudentScores.txt file.
- The students should be displayed in alphabetical order.

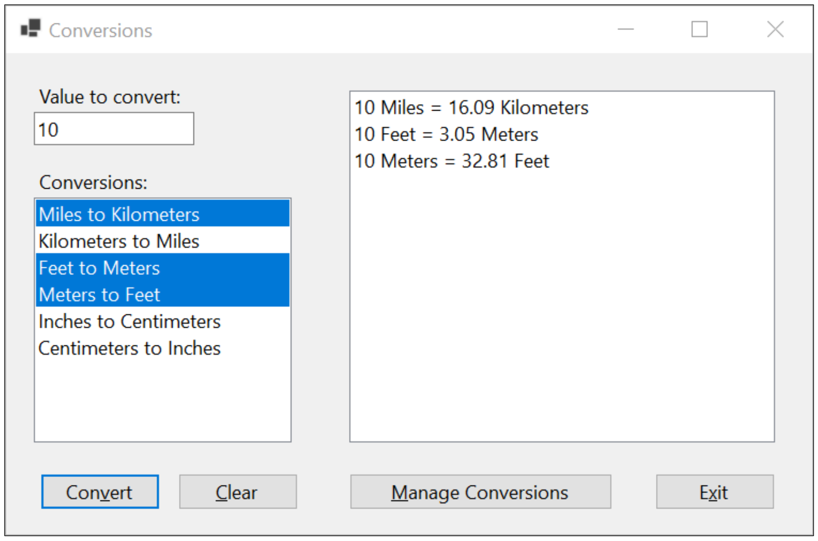
Instructor note

- Instead of storing the student data in a text file, you can modify this project so the data is stored in a binary file.

Project 4-2 Calculate conversions

For this project, you'll develop an app that lets the user perform conversion calculations. This app requires a main form that performs one or more conversion calculations, and two additional forms that let the user add or remove conversion options, as well as restore a list of default options. This app uses two text files and two classes. Prerequisites: chapters 1-12, 14, 17.

The main form

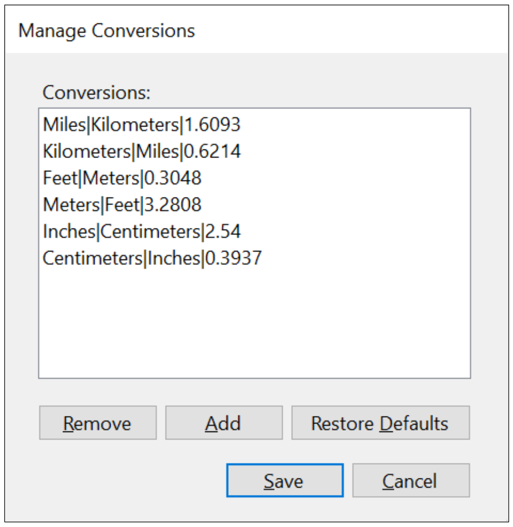


```
10 Miles = 16.09 Kilometers
10 Feet = 3.05 Meters
10 Meters = 32.81 Feet
```

Operation

- To perform a conversion, the user enters a value to convert in the text box and selects one or more conversions to be performed on that value from the Conversions list box. When the user clicks the Convert button, the selected conversion calculations are displayed in the list box on the right side of the main form.
- To clear the form, the user clicks the Clear button.
- To add or remove conversion options, or to restore the default conversion options, the user clicks the Manage Conversions button to display the Manage Conversions dialog.
- To close the app, the user clicks the Exit button.

The Manage Conversions form

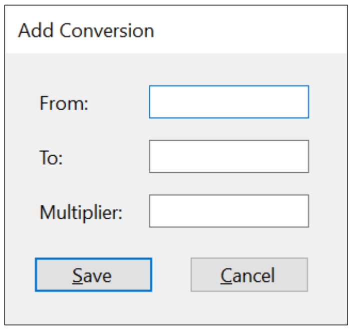


```
Miles|Kilometers|1.6093
Kilometers|Miles|0.6214
Feet|Meters|0.3048
Meters|Feet|3.2808
Inches|Centimeters|2.54
Centimeters|Inches|0.3937
```

Operation

- To remove a conversion so it's not included on the main form, the user selects the conversion in the list and then clicks the Remove button.
- To add a conversion to the main form, the user clicks the Add button to display the Add Conversion dialog.
- To restore the default options, the user clicks the Restore Defaults button.
- To save all the changes made to the options, the user clicks the Save button. To cancel the changes, the user clicks the Cancel button.

The Add Conversion form



Operation

- To add a conversion, the user enters unit descriptions in the From and To text boxes and a multiplier that indicates the number of *To* units in a *From* unit in the Multiplier text box, and then clicks the Save button.

Specifications

- This app should use a class named Conversion to store information about each Conversion.
- The Conversion class should override the ToString() method of the Object class to provide a string representation of each Conversion object that includes the From and To units as shown in the Conversions list box on the Conversion form.

- All user entries should be checked to be sure they're valid.
- When the app starts, the parent form should do the following:
  1. Read the conversion data from a text file named Conversions.txt and store it in a List.
  2. Display the conversion options in the List in a list box control that allows for multiple selections.
- If conversions are added, removed, or restored, the Conversions.txt file should be updated on save.
- The default conversions are stored in a text file named ConversionsDefault.txt in the same directory as the other project files. These conversions are as follows:

From	To	Conversion
Miles	Kilometers	1 mile = 1.6093 kilometers
Kilometers	Miles	1 kilometer = 0.6214 miles
Feet	Meters	1 foot = 0.3048 meters
Meters	Feet	1 meter = 3.2808 feet
Inches	Centimeters	1 inch = 2.54 centimeters
Centimeters	Inches	1 centimeter = 0.3937 inches

- The code for working with the Conversions and ConversionsDefault files should be stored in a class named ConversionsDB. The code for validating the user entries should be stored in a class named Validator.

Instructor note

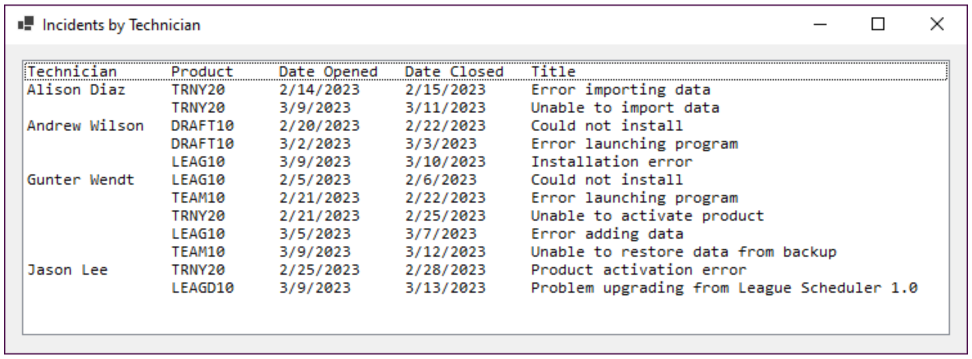
- For this project, you need to supply your students with the Conversions.txt and ConversionsDefault.txt files that are in the Projects/Files/Project 4-2 directory.
- If you want to be more specific about the Conversion class, you can provide this class design:

Constructor	Description
`()`	Creates a Conversion object with default From, To, and Multiplier values.
`(string row)`	Creates a Conversion object with the From, To, and Multiplier values in the row that's passed to it.
Property	**Description**
`From`	A string that stores the type of units to be converted from.
`To`	A string that stores the type of units to be converted to.
`Multiplier`	A decimal that stores the number of To units in the From unit.
`FullTeXt`	A string that stores the From, To, and Multiplier properties, separated by pipe characters as shown in the Conversions list box of the Manage Conversions form.
Method	**Description**
`DisplayConversion( decimal value)`	Returns the result of a conversion, formatted as shown in the right list box of the Conversions form.
`ToString()`	Returns the From and To properties, formatted as shown in the Conversions list box of the Conversions form.

Project 4-3 Display incidents by technician

For this project, you'll develop an app that displays closed incidents by technician.
Prerequisites: chapters 1-12, 17, 18.

The Incidents by Technician form



```
Technician     Product    Date Opened   Date Closed   Title
Alison Diaz    TRNY20     2/14/2023     2/15/2023     Error importing data
               TRNY20     3/9/2023      3/11/2023     Unable to import data
Andrew Wilson  DRAFT10    2/20/2023     2/22/2023     Could not install
               DRAFT10    3/2/2023      3/3/2023      Error launching program
               LEAG10     3/9/2023      3/10/2023     Installation error
Gunter Wendt   LEAG10     2/5/2023      2/6/2023      Could not install
               TEAM10     2/21/2023     2/22/2023     Error launching program
               TRNY20     2/21/2023     2/25/2023     Unable to activate product
               LEAG10     3/5/2023      3/7/2023      Error adding data
               TEAM10     3/9/2023      3/12/2023     Unable to restore data from backup
Jason Lee      TRNY20     2/25/2023     2/28/2023     Product activation error
               LEAGD10    3/9/2023      3/13/2023     Problem upgrading from League Scheduler 1.0
```

Operation

- When this app starts, all the closed incidents are listed by technician in a ListBox control.

Specifications

- Add the following classes to the project:

Incident

Properties
`CustomerID`
`DateClosed`
`DateOpened`
`Description`
`IncidentID`
`ProductCode`
`TechID`
`Title`

IncidentDB

Fields
`Dir`
`Path`
Method
`GetIncidents()`

Technician

Properties
`Name`
`TechID`

TechnicianDB

Fields
`Dir`
`Path`
Method
`GetTechnicians()`

The GetTechnicians() method in the TechnicianDB class should return a List<Technician> object, and the GetIncidents() method in the IncidentDB class should return a List<Incident> object. These objects should be loaded from text files named Technicians.txt and Incidents.txt that your instructor gives you.

- Use LINQ to define a query expression that will return the fields required by this app from the List<> objects. Only the closed incidents should be returned, and those incidents should be sorted by the date opened within the technician name.
- Include the technician name only for the first incident for each technician.

Hint

- Because the fields in the Incidents text file that store the technician ID and date closed can contain empty strings, you will need to define the TechID and DateClosed fields in the Incident class as nullable. Then, when you retrieve the values for an incident from the text file, you can check these fields to see if they contain empty strings. If they do, you can assign null to the fields in the Incident object.
- To make sure the fields align correctly in the list box, you may want to use a monospaced font such as Consolas for the list box and use spaces to align the fields. To do that, you can use the PadRight() method of the String class to add the spaces needed to align each field.

Instructor notes

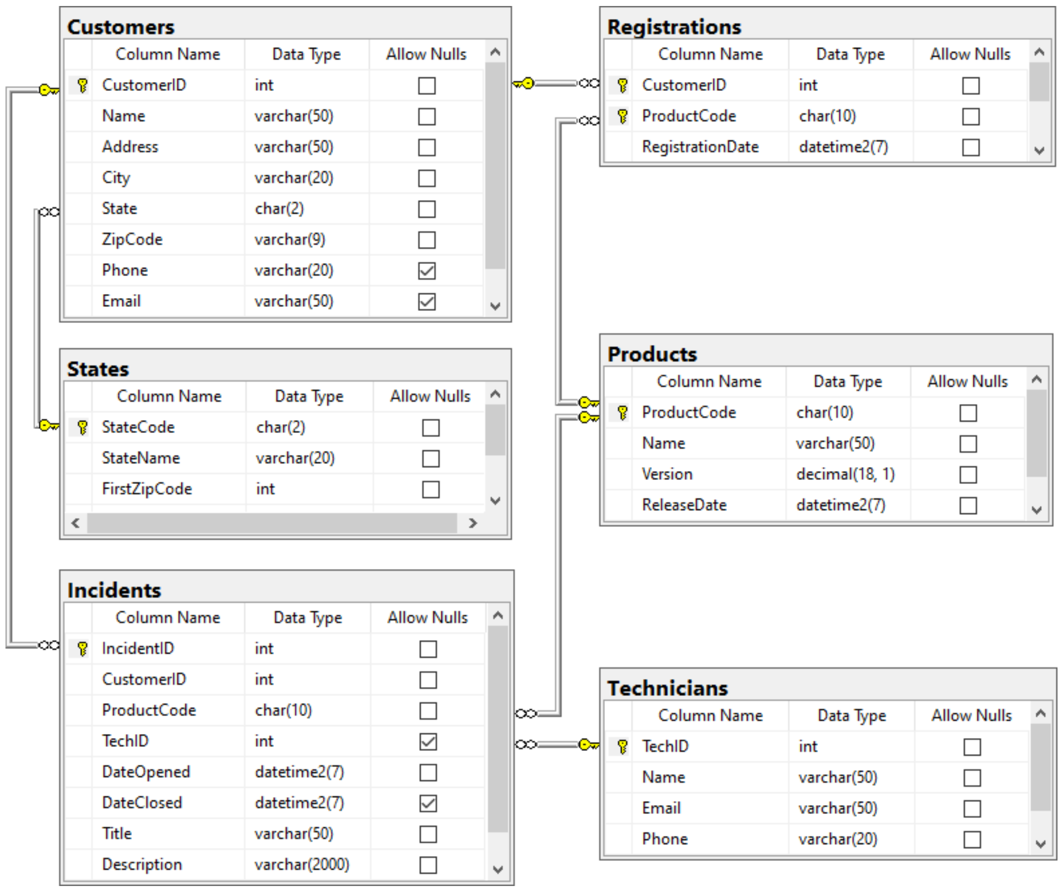
- For this project, you need to supply your students with the Technicians.txt and Incidents.txt files that are in the Projects/Files/Project 4-3 directory.
- You can simplify this project by providing your students with the TechnicianDB and IncidentDB classes.
- You can also simplify this project by not requiring that the technician names be printed only for the first incident for each technician.
- To enhance this project, you can have your students add a column for the customer name. This involves adding a business class that defines a Customer object and a database class with a method that loads a file of customers into a List<Customer> object. (The Customers.txt file is also in the Files/Project 4-3 directory.) Then, the List<Customer> object can be included in the query expression.

How to use the TechSupport database

The projects for section 5 have you develop apps that work with a database named TechSupport. These apps are used by a hypothetical software company that develops software for sports leagues. The main purpose of the database is to track technical support service calls, referred to as *incidents*. Before you use this database, you need to be familiar with its design and prepare it for use.

The design of the TechSupport database

The TechSupport database consists of the six tables shown below. The main table is the Incidents table, which contains one row for each technical support incident. Each row in the Incidents table is related to one row in the Customers table, which contains information about the company's customers; one row in the Products table, which contains information about the company's products; and one row in the Technicians table, which contains information about the company's technical support staff members. In addition, a table called Registrations keeps track of the products that are registered to each customer, and the States table contains state and zip code information.



Customers

	Column Name	Data Type	Allow Nulls
🔑	CustomerID	int	☐
	Name	varchar(50)	☐
	Address	varchar(50)	☐
	City	varchar(20)	☐
	State	char(2)	☐
	ZipCode	varchar(9)	☐
	Phone	varchar(20)	☑
	Email	varchar(50)	☑

Registrations

	Column Name	Data Type	Allow Nulls
🔑	CustomerID	int	☐
🔑	ProductCode	char(10)	☐
	RegistrationDate	datetime2(7)	☐

States

	Column Name	Data Type	Allow Nulls
🔑	StateCode	char(2)	☐
	StateName	varchar(20)	☐
	FirstZipCode	int	☐

Products

	Column Name	Data Type	Allow Nulls
🔑	ProductCode	char(10)	☐
	Name	varchar(50)	☐
	Version	decimal(18, 1)	☐
	ReleaseDate	datetime2(7)	☐

Incidents

	Column Name	Data Type	Allow Nulls
🔑	IncidentID	int	☐
	CustomerID	int	☐
	ProductCode	char(10)	☐
	TechID	int	☑
	DateOpened	datetime2(7)	☐
	DateClosed	datetime2(7)	☑
	Title	varchar(50)	☐
	Description	varchar(2000)	☐

Technicians

	Column Name	Data Type	Allow Nulls
🔑	TechID	int	☐
	Name	varchar(50)	☐
	Email	varchar(50)	☐
	Phone	varchar(20)	☐

In addition to the column properties shown above, you should know that the CustomerID, IncidentID, and TechID columns in the Customers, Incidents, and Technicians tables are identity columns. So, the values of these columns are set automatically when new rows are added to these tables.

How to prepare for using the TechSupport database

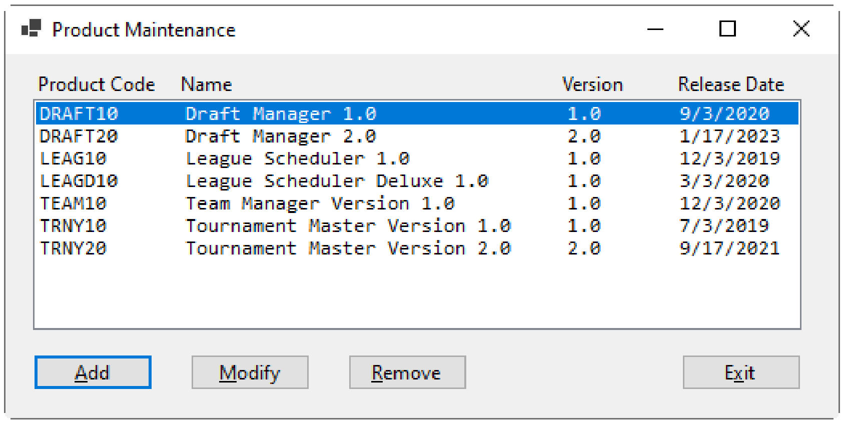
Unless your instructor tells you otherwise, you should use SQL Server Express LocalDB to work with the TechSupport database. In that case, you can just copy the TechSupport.mdf and TechSupport_log.ldf files your instructor gives you to your system and use them. By contrast, if you're doing the projects in a computer lab and the database is installed on the local computer or on a server, you will need to get the information for establishing a connection to that database before you do the projects.

Section 5 projects

Project 5-1 Maintain products

For this project, you'll create a form that lets the user maintain products. Prerequisites: chapters 1-12, 18-20.

The main form



```
DRAFT10     Draft Manager 1.0                 1.0     9/3/2020
DRAFT20     Draft Manager 2.0                 2.0     1/17/2023
LEAG10      League Scheduler 1.0              1.0     12/3/2019
LEAGD10     League Scheduler Deluxe 1.0       1.0     3/3/2020
TEAM10      Team Manager Version 1.0          1.0     12/3/2020
TRNY10      Tournament Master Version 1.0     1.0     7/3/2019
TRNY20      Tournament Master Version 2.0     2.0     9/17/2021
```

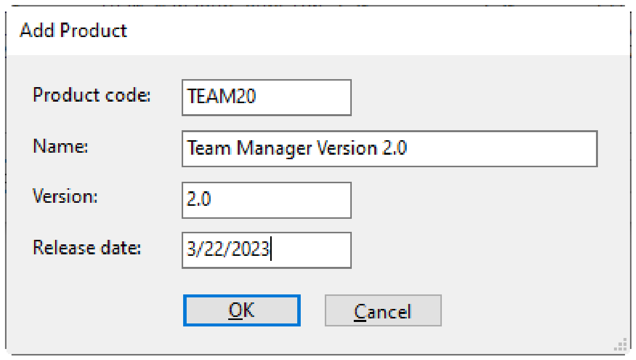
Operation

- When this app starts, all the rows in the Products table of the TechSupport database are displayed in the ListBox control, in alphabetical order by name.
- This display should use tabs to separate the product fields, and it should not include the navigation properties of the Product entity.
- To add a new product, the user can click the Add button and enter data in the Add Product form. The main form should refresh to display the new product.
- To modify a product, the user can select the product in the list box, click the Modify button, and update data in the Modify Product form. If a product isn't selected, the user should be notified to select a product. Otherwise, the main form should refresh to display the updated product.
- To remove a product, the user can select the product in the list box, click the Remove button, and click OK in the confirm deletion dialog. If a product isn't selected, the user should be notified to select a product. Otherwise, the main form should refresh to display the remaining products.
- If a product selected for removal has related Incidents or Registrations in the database, the app should notify the user and not allow the removal of the product.
- To close the app, the user clicks the Exit button.

Specifications

- To make sure the fields align correctly in the list box, you may want to use a monospaced font such as Consolas for the list box and use spaces to align the fields. To do that, you can use the PadRight() method of the String class to add the spaces needed to align each field.

The Add/Modify form


Add Product

Product code:	TEAM20
Name:	Team Manager Version 2.0
Version:	2.0
Release date:	3/22/2023

Operation

- To add or modify a product, the user enters the appropriate data and clicks OK.
- To cancel and go back to the main form, the user clicks Cancel.
- The form should validate the data entered by the user.
- When modifying a product, the Product code field should be disabled.

Specifications

- This app should use Entity Framework (EF) Core and a data access class to interact with the database, and it should handle database exceptions.
- Because the ProductCode column is a primary key, don't allow the user to change the value in this column.
- You can use the same form to add and modify products.
- Limit the number of characters that can be entered into the Product code and Name text boxes so the entries don't exceed the sizes allowed by the ProductCode and Name columns.
- Limit the date that can be entered into the Release date text box to no earlier than 01/01/0001 and no later than 12/31/9999. These are the limits of the DateTime2 data type in SQL Server.

Hint

- When you use the DataSource property of the ListBox control to bind the results of a query, the control displays the result of the ToString() method for each entity. So, one way to control how an entity displays in a ListBox control is to override the ToString() method of the entity.

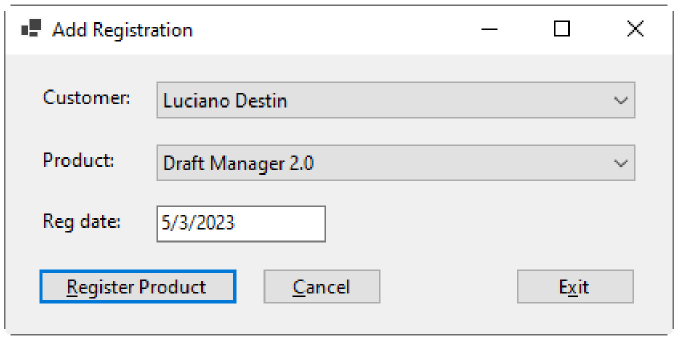
Instructor notes

- To enhance this project, you can have your students also handle concurrency exceptions. This will involve adding a RowVersion field to the database table and re-generating the EF entity classes and DB context class.
- If your students have read chapter 22, you can have them use a DataGridView control instead of a ListBox. This provides a more reliable and flexible way to align and format the data.

Project 5-2 Register products

For this project, you'll develop an app that lets the user register products to customers. Prerequisites: chapters 1-12, 18, 19, 21.

The Add Registration form



Operation

- The user selects a customer and product from the two combo boxes, enters a date in the text box, and clicks the Register Product button to add a new row to the Registrations table.
- If the selected product is already registered to the selected customer, a database error is thrown. Otherwise, the product is registered.
- After the user registers a product or clicks the Cancel button to cancel the add operation, the Add Registration form is cleared.

Specifications

- Use ADO.NET code to interact with the database.
- Add the following classes to the project:

Customer

Properties
CustomerID
Name

Product

Properties
Name
ProductCode

Registration

Properties
CustomerID
ProductCode
RegistrationDate

TechSupportDataAccess

Methods
GetCustomers()
GetProducts()
AddRegistration()

DataAccessException

Constructor
DataAccessException()
Properties
Message
ErrorType

The DataAccessException class should inherit the Exception class, and its constructor should accepts the message and error type as strings.

In the TechSupportDataAccess class, the GetCustomers() method should return a List<Customer> object that can be bound to the Customer combo box. The

GetProducts() method should return a List<Product> object that can be bound to the Product combo box. The AddRegistration() method should accept a Registration object and return void. All three methods should throw a DataAccessException object in the event of a database error.

- Validate the data on the Add Registration form to be sure that a customer and product are selected and a valid registration date no more than 20 years in the past is entered. The default registration date for a new registration should be the current date.

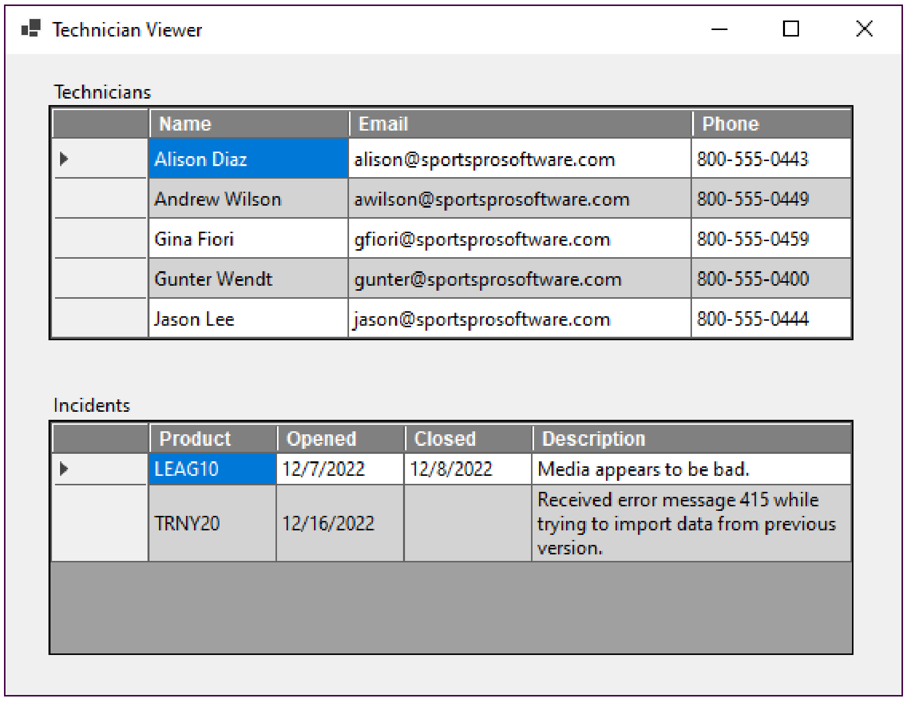
Instructor notes

- To enhance this project, you can have your students add another method to the data access class that checks if a product is already registered to a customer. Then, this method can be used when registering a product.
- If your students have read chapter 16, you can have them store the business and database classes in a class library.

Project 5-3 Display technicians and incidents in a Master/Detail form

For this project, you'll create a form that displays technicians in a DataGridView control and the related incidents for each technician in a DataGridView control. Prerequisites: chapters 1-12, 18-20, 22.

The main form



Technicians

	Name	Email	Phone
▶	Alison Diaz	alison@sportsprosoftware.com	800-555-0443
	Andrew Wilson	awilson@sportsprosoftware.com	800-555-0449
	Gina Fiori	gfiori@sportsprosoftware.com	800-555-0459
	Gunter Wendt	gunter@sportsprosoftware.com	800-555-0400
	Jason Lee	jason@sportsprosoftware.com	800-555-0444

Incidents

	Product	Opened	Closed	Description
▶	LEAG10	12/7/2022	12/8/2022	Media appears to be bad.
	TRNY20	12/16/2022		Received error message 415 while trying to import data from previous version.

Operation

- When this app first starts, the technicians are displayed in the first grid in the form. In addition, the related incidents for the first technician in the grid are displayed in the second grid in the form.
- When the user clicks on a row header in the Technicians grid, the incidents for that technician are displayed in the Incidents grid.

Specifications

- This app should use Entity Framework (EF) Core to interact with the database.
- Change the properties for the DataGridView controls so rows can't be added or deleted. In addition, format the columns so they look like the ones shown above.
- The TechID field should be included in the Technicians DataGridView control, so it can be used to select the related incidents. But it shouldn't display to the user.

Hint

- To have the contents of a cell display on multiple lines, you can include code like this:

```
dgv.AutoSizeRowsMode = DataGridViewAutoSizeRowsMode.AllCells;
dgv.Columns[4].DefaultCellStyle.WrapMode = DataGridViewTriState.True;
```

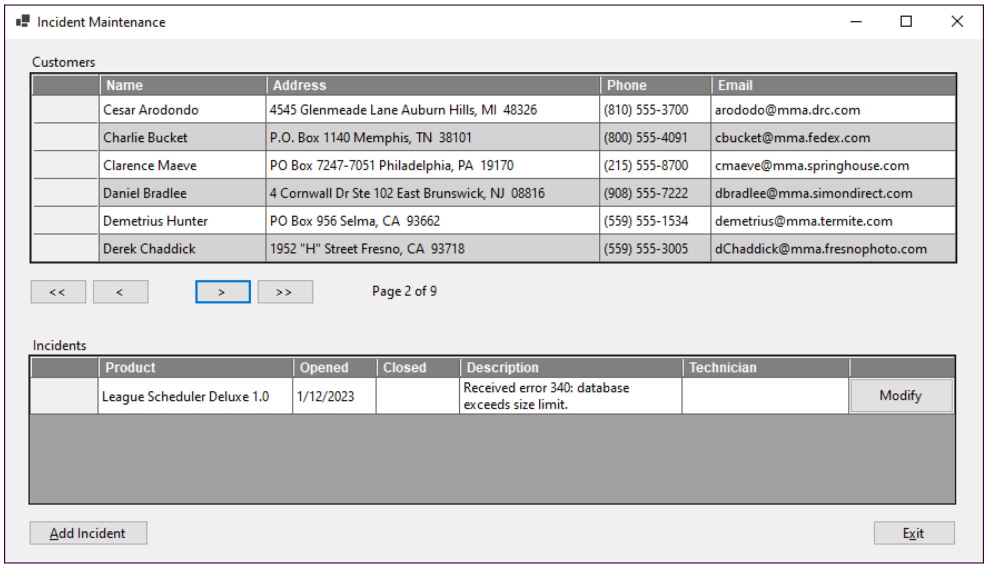
Instructor note

- To enhance this project, you can have your students use a data access class.
- If your students have read chapter 16, you can have them store the business and database classes in a class library.

Project 5-4 Maintain incidents

For this project, you'll develop an app that displays customers and their related incidents in a Master/Detail form, provides paging to move through the customers, and allows users to enter and modify incidents. Prerequisites: Chapters 1-12, 18-20, 22.

The Incident Maintenance form



	Name	Address	Phone	Email
	Cesar Arodondo	4545 Glenmeade Lane Auburn Hills, MI 48326	(810) 555-3700	arododo@mma.drc.com
	Charlie Bucket	P.O. Box 1140 Memphis, TN 38101	(800) 555-4091	cbucket@mma.fedex.com
	Clarence Maeve	PO Box 7247-7051 Philadelphia, PA 19170	(215) 555-8700	cmaeve@mma.springhouse.com
	Daniel Bradlee	4 Cornwall Dr Ste 102 East Brunswick, NJ 08816	(908) 555-7222	dbradlee@mma.simondirect.com
	Demetrius Hunter	PO Box 956 Selma, CA 93662	(559) 555-1534	demetrius@mma.termite.com
	Derek Chaddick	1952 "H" Street Fresno, CA 93718	(559) 555-3005	dChaddick@mma.fresnophoto.com

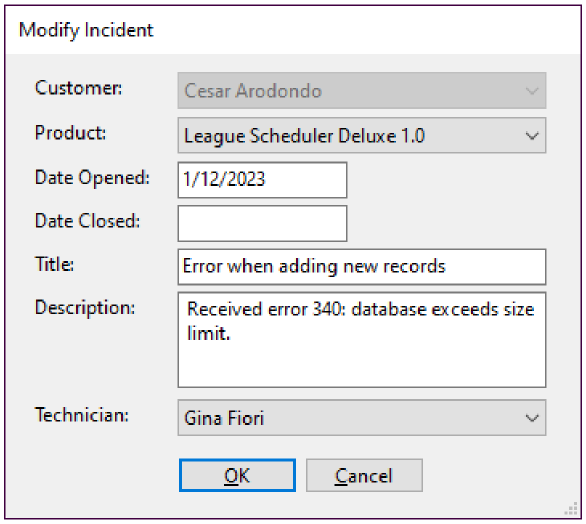
Page 2 of 9

	Product	Opened	Closed	Description	Technician	
	League Scheduler Deluxe 1.0	1/12/2023		Received error 340: database exceeds size limit.		Modify

Operation

- When this app first starts, the customers are displayed in the first grid in the form. In addition, the related incidents for the first customer in the grid are displayed in the second grid in the form.
- When the user clicks on a row header in the Customers grid, the incidents for that customer are displayed in the Incidents grid.
- Only 6 customers are displayed at a time. To view customers that aren't currently displayed, the user can click on the First, Next, Previous, and Last buttons below the first grid. The total number of pages and the current page number display next to these navigation buttons.
- When the user moves to a new page, the incidents for the first customer of the new page are displayed in the Incidents grid.
- To add a new incident, the user clicks the Add Incident button and enters data in the Add Incident form. The main form should refresh to display the new incident.
- To modify an incident, the user clicks the Modify button in the row for the incident, and updates data in the Modify Product form. The main form should refresh to display the updated incident.

The Add/Modify Incident form



Modify Incident

Customer:	Cesar Arodondo
Product:	League Scheduler Deluxe 1.0
Date Opened:	1/12/2023
Date Closed:	
Title:	Error when adding new records
Description:	Received error 340: database exceeds size limit.
Technician:	Gina Fiori

Operation

- To add or modify an incident, the user enters the appropriate data and clicks OK.
- To cancel and go back to the main page, the user clicks Cancel.
- The form should validate the data entered by the user, including making sure that Date Closed isn't earlier than Date Opened. The form should allow the user to leave the Date Closed field blank and to not select a technician.
- When modifying an incident, the Customer combo box should be disabled.

Specifications

- Use Entity Framework (EF) Core to interact with the database and handle database exceptions.
- Use the same form to add and modify incidents.
- Change the properties for the DataGridView controls so rows can't be added or deleted. In addition, change the properties of the columns so they look like the ones shown above.
- Include the CustomerID field in the Customers DataGridView control, so it can be used to select the related incidents. But don't display it to the user.
- Bind the combo boxes so the customer, product, and technician names are displayed and the related id values are stored in the control.
- Limit the number of characters that can be entered into the Title and Description text boxes so the entries don't exceed the sizes allowed by the Title and Description columns.
- Limit the date that can be entered into the Date Opened and Date Closed text boxes to no earlier than 01/01/1753 and no later than 12/31/9999. These are the limits of the DateTime data type in SQL Server.

Hint

- To have the contents of a cell display on multiple lines, you can include code like this:

```
dgv.AutoSizeRowsMode = DataGridViewAutoSizeRowsMode.AllCells;
dgv.Columns[4].DefaultCellStyle.WrapMode = DataGridViewTriState.True;
```

Instructor notes

- You can enhance this project by having students add the ability to delete incidents or transfer them to other customers.
- You can also enhance this project by allowing students to add, modify, and remove customers.
- You can enhance the Add/Modify Incident form by not allowing open and close dates to be in the future.
- You can enhance this project by having your students use a data access class.
- If your students have read chapter 16, you can have them store the business and database classes in a class library.