This lab contains the following exercises and activities:

8.1 Creating a Reaction Timer Game.................................................................1
8.1.1 Displaying the Game State .........................................................................2
8.1.2 Displaying the Score Values ........................................................................3
8.2 Sorting the Score Display..............................................................................4
8.2.1 Getting Started with Sorting........................................................................4
8.2.2 Adding a Sort Behavior to the Reaction Timer Game...............................5
8.2.3 Tidying Up the Game..................................................................................6
8.2.4 Using the Sort and Display Code in Button Bash.................................7

SCENARIO

In the sessions for this part of the course you saw how a reaction timer game was created. In this lab, you will improve the game to sort the times into order and display them.

After completing this lab you will be able to:

- manage the display of multiple items on the game screen.
- sort an array of time values into ascending order.

Estimated lesson time: 50 minutes

8.1 CREATING A REACTION TIMER GAME

Estimated completion time: 20 minutes

At the moment, the reaction timer game just displays the winning button score on the screen at the end of the game. This means that the player with the fastest reaction time is displayed.

However, with lots of players it would be nice if all the results were displayed, so that everyone can see how well they did. It would be even better if the results were displayed in order, so that bragging rights could be awarded correctly.

In this exercise, you are going to improve the reaction timer game that was developed in the taught sessions. To start you are going to make the game easier to play by making the program display what the players must do at each stage of the game. Then you are going to make the game display a list of all the scores on the screen at the end of the game.

Note: To properly perform this practical session you will need to have a gamepad connected to your machine. You will also need to have audio output from your machine.
8.1.1 Displaying the Game State

At the moment, the game is not very easy to play, since it is not clear what the players have to do. The first thing we are going to do is change the Draw method so that it displays appropriate messages, depending on the state of the game.

1. Open the ReactionTimer project in the folder 8.3.1 Reaction Timer Starter.
2. Open the Game1.cs program file.
3. Navigate to the Draw method in this program. At the moment the method just contains statements that call SpriteBatch.Begin() and SpriteBatch.End().
4. If the timer value is less than 0 we should tell the players to get ready, as the sound will be playing soon. We can do this by adding a conditional statement which controls the display of a message. Change the Draw method to add this as shown below:

```csharp
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    spriteBatch.Begin();
    if (timer < 0)
    {
        Vector2 readyPosition = new Vector2(200, 200);
        spriteBatch.DrawString(font, "GET READY!!", readyPosition, Color.White);
    }
    spriteBatch.End();
    base.Draw(gameTime);
}
```

5. Run the program and press the start button. Note that as soon as the start button is pressed (which sets the timer value to -120) the "GET READY!!" message is displayed. Note that this message is also displayed the very first time the game starts, because the initial value of timer is set to -120 at the start of the game. When the sound is played the message disappears. This is because at that point the value of timer is greater than 0, and the condition controlling the text display is no longer true.

6. Now you need to add the message that is displayed when the players must press their buttons. This must be displayed when the timer value is between 0 and 120. The best way to do this is to "nest" if conditions (i.e. put one inside another) in the following way:
if (timer < 0)
{
    // Display the "Get Ready" message
}
else
{
    if (timer < 120)
    {
        // display the "Press Your Button" message
    }
    else
    {
        // display the scores
    }
}

7. You can test out the conditions above by trying different values and making sure that the program does the correct thing for each one. Now create the conditions above inside the Draw method (you already have the first part) and add the code below in the correct place. This is the code that displays the prompt to press a button on the gamepad as quickly as possible.

Vector2 goPosition = new Vector2(150, 300);
spriteBatch.DrawString(font, "** PRESS YOUR BUTTON **",
goPosition, Color.Red);

8. Run the program and confirm that when you press Start the program displays the "GET READY!!" message, followed by the "** PRESS YOUR BUTTON **" message.

8.1.2 Displaying the Score Values

The game is now a bit easier to play, in that the players can see what they need to do at each point in the game. Now you need to add some code that will display all the score values on the screen.

The easiest way to do this is to use a for loop to display each value, and the name of the button, in turn. Each time round the loop, the program must assemble a message from the name of the button and the score value. The code below does this by using a string variable called scoreString which is created from to the name and timer value, separated by a colon character.

The code also uses a Vector2 called scorePosition to position each draw action. The vector is moved down the screen after each draw, so that each score value is displayed on a new line on the display:

Vector2 scorePosition = new Vector2(20, 10);
for (int i = 0; i < 16; i++)
{
    string scoreString;
    scoreString = names[i] + " : " + scores[i].ToString();
    spriteBatch.DrawString(font, scoreString, scorePosition, Color.White);
    scorePosition.Y = scorePosition.Y + 30;
}
spriteBatch.DrawString(font, "Press Start to play again", scorePosition, Color.White);

1. Add the score display as shown above, at the correct point in the program.
2. Run the program and note that the game now displays the score table at the end.

8.2 **SORTING THE SCORE DISPLAY**

Estimated completion time: 30 minutes

In this exercise, you are going to update the display so that a sorted list of the winners is displayed on the screen. At the moment the list of players is displayed in the order that they are held in the scores array. You are going to have to create some C# that will sort the scores array into order.

8.2.1 **Getting Started with Sorting**

The simple act of getting a list of numbers into order is something which has exercised many of the great brains of computing over the years. If you look up "Sorting" in a Computer Science library you will find many volumes which have been written about the best way to do this. The names of some of the techniques sound great fun, with "Shell Sorts," "Bubble Sorts," "Tournament Sorts," and "Insertion Sorts," to name just a few.

Some of these come into their own when you have many thousands, or even millions, of items to sort. We only have sixteen values to put in order, and so we can use one of the simplest sorting techniques, the Bubble Sort. The fundamental process of Bubble Sort is very easy to state:

"If two adjacent numbers are in the wrong order, swap them over."

Remember that for a computer to understand the instructions that we give it, we need to express those actions in a very simple way. In this respect the Bubble Sort is good news, because it is easy to explain. The central part of a Bubble Sort is a simple test:

```csharp
if (scores[i] > scores[i + 1])
{
    // the two elements need to be swapped
}
```

We want to have the largest value at the bottom of array, with the smallest ones at the top. The variable \(i\) above contains the subscript of an element in the array, if the element at \(i+1\) is larger than this, the two values must be swapped around, because they are in the wrong order (remember we want the smallest elements at the "front" of the array).

Of course, just swapping two incorrectly positioned elements won't actually put the array into order, because many more than two elements might be in the wrong place. However, it will make the array "less out of order." To properly sort the array we have to repeat this sorting action many times, using a loop:
for (int i = 0; i < 15; i++)
{
    if (scores[i] > scores[i + 1])
    {
        // the two elements need to be swapped
        int temp = scores[i];
        scores[i] = scores[i + 1];
        scores[i + 1] = temp;
    }
}

This code makes a pass through the entire scores array, comparing adjacent elements and swapping ones that are out of order. Note that the swap behavior uses a temporary variable when the numbers are swapped over, if you think about this, it makes sense.

The for loop above only takes the value of i to 14, rather than 15. Can you think why this is?

Note Think about what happens when you are comparing the element at i with the element at i+1.

This sorting does not completely sort the array, but at least it means that the array contents are less out of order after it is complete.

To completely sort the array we need to repeat this entire sorting process a number of times. It turns out that if the scores array was in exactly the wrong order (i.e. sorted into descending order with the largest value at the top and the smallest at the bottom) we would have to make 15 passes through the array to get everything into the right place.

We can achieve this by adding another loop to repeat the passes through the data.

### 8.2.2 Adding a Sort Behavior to the Reaction Timer Game

We can add this sorting behavior to the Update method, so that when the timer value reaches 120 the sort is triggered.

if (timer == 120)
{
    for (int passes = 0; passes < 15; passes++)
    {
        for (int i = 0; i < 15; i++)
        {
            if (scores[i] > scores[i + 1])
            {
                // the two elements need to be swapped
                int temp = scores[i];
                scores[i] = scores[i + 1];
                scores[i + 1] = temp;
            }
        }
    }
}

1. Use the ReactionTimer project that you created in the previous exercise as the basis of this work.
2. Add the sorting behavior to the Update method so that the game sorts the scores into ascending order.
3. Play the game and note what happens. You should notice that while the numbers are displayed in ascending order, they are no
longer against the correct button names. The way to fix this problem is to swap the button names at the same time as we swap the scores values. This means that the button names will keep up with the scores values. Add this code in the correct place:

```csharp
string tempName = names[i];
names[i] = names[i + 1];
names[i + 1] = tempName;
```

**Note** If you are not sure where to add this code, remember that it must run when the number values are swapped.

4. Run the game, play it, and note what happens. The score values are now displayed in ascending order, alongside the correct button names.

5. Press the Start button and play the game again. You should find that there is a problem the second time that the game is played.

**Note** If you can't see the problem, check to see if the button names still match up when the game is played the second time.

6. This bug is caused because at the start of the first game the strings in the `names` array were all lined up with the score values in the `scores` array, i.e. the element at 0 in the `scores` array matched with the string "Gamepad 1 A" in the `names` array. However, when the scores were sorted the names were moved to match up with the sorted array, and this means that they are out of order when they are used next time. The best way to fix this is to reset the names back into their original positions at the start of each sort. This means adding an extra array, which we could call "masterNames," which is copied into the `names` array before the sorting is performed. Change the name of the `names` array to `masterNames`.

7. Add a new array of strings which is called `names` to the game data for this game. This array should have 16 elements:

```csharp
string[] names = new string[16];
```

8. Now add code to the sorting code which is obeyed when the timer reaches 120. This should be performed just before the sorting starts.

```csharp
// copy the master names into the names array
for (int i = 0; i < 16; i++)
{
    names[i] = masterNames[i];
}
```

9. Now run the program and note that after the first play of the game the names match the score values correctly.

#### 8.2.3 Tidying Up the Game

The game is quite playable at the moment, but it does have one or two gameplay issues.

- Players are able to press their buttons after 120 seconds, which causes the display to be overwritten with invalid values.
• Negative scores (when a player has pressed their button before the start of the game) are still displayed.
• Scores of 0 (which means that a player has not pressed their button) are displayed on the winner display, which is confusing.

The first two problems are easy to fix. In the `Update` method, at the moment the game tests for button presses all the time. Instead, all the button detection code should only be enabled when the timer value is between 0 and 120:

```csharp
if (timer > 0 && timer < 120)
{
    // Test for buttons being pressed
}
```

1. Use the `ReactionTimer` project that you created in the previous exercise as the basis of this work.
2. Add the code above so that the buttons are only tested when the timer is in the range given.

To fix the third problem you need to modify the display code so that only score values that are greater than 0 are displayed:

```csharp
for (int i = 0; i < 16; i++)
{
    if (scores[i] > 0)
    {
        string scoreString = names[i] + " : " + scores[i].ToString();
        spriteBatch.DrawString(font, scoreString, scorePosition, Color.White);
        scorePosition.Y = scorePosition.Y + 30;
    }
}
```

In this version of the code that draws the scores, they are only displayed if the score is greater than 0. This means that any unused buttons are ignored, which makes the display look much tidier.

3. Modify the drawing code in the `Draw` method so that only scores which are greater than 0 are displayed.

### 8.2.4 Using the Sort and Display Code in Button Bash

You now have a sort method and a display method that you can use in any game. You could even use all the code that you have just created in the Button Bash game, so that you could make it display a live leaderboard that changed as the game was played and players speeded up and slowed down.

I will leave this as an exercise for you to work on.