Mr. Bellavia

LESSON 05 - PROBLEM SOLVING

In this lesson we will look at a few examples of how we can solve basic computer problems whereby we use an input file that is run it through a program we write which will produce output representing the solution.

Sections:

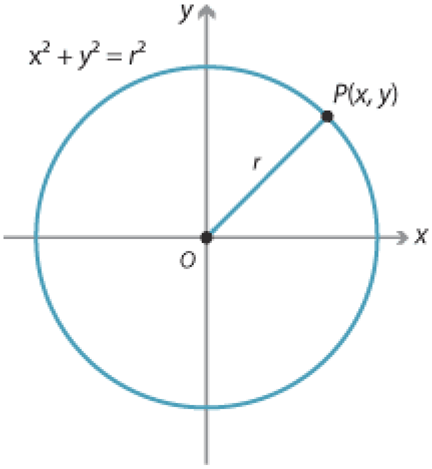
I. EXAMPLES:

So far in this course, we have explored fundamental concepts such as variables, selection (e.g., if-statements), repetition (e.g., for-loops), and reading/writing to/from files. Now we are going to use these fundamentals to solve problems where we will be presented with a problem with a given set of data (in the form of an input file). We will then run this input through a program we write to produce a desired output.

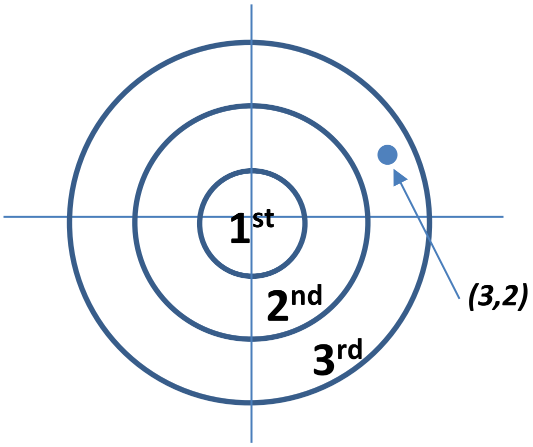
Problem 1: Dart Board

Program Description:

The equation of circle at the origin of a cartesian plane is represented by the following:



Imagine a dartboard on the same cartesian plane. The **first circle** is the center (bullseye) with a radius of 6, the **second** has a radius of 12 and the **third** a radius of 20:



If a dart was represented by a coordinate *(x, y)* you would be able to determine if a dart landed in one of the circles. For example, coordinate ***(3, 2)***:

Circle #	$x^2 + y^2$ **= r**	**Hit?**
1st	$3^2 + 2^2 > 6$	no
2nd	$3^2 + 2^2 > 12$	no
3rd	$3^2 + 2^2 < 20$	**yes**

From the above table we can see that the dart hit the 3rd circle. Using the above dartboard layout, write a program that will determine if a coordinate has landed on a circle on the dartboard.

Note: With the above equation there will be scenarios where the dart hits more than one circle. For example, if a dart landed on circle 2 it would technically be in circle 3 as well. Your program should determine the correct circle in such scenarios.

Input:
The input should come from a file named **"input.txt"**. The first and second lines contain an integer *x* and *y* respectively representing the dart coordinate.

Output:
The output should be the circle number that was successfully hit (1, 2, or 3). A ‘0’ should be outputted if the dartboard was not hit.

Sample Input 1:

```
3
2
```

Sample Output 1:

```
3
```

Sample Input 2:

```
5
3
```

Sample Output 2:

0

Sample Input 3:

```
3
1
```

Sample Output 3:

```
2
```

Sample Input 4:

```
-1
-2
```

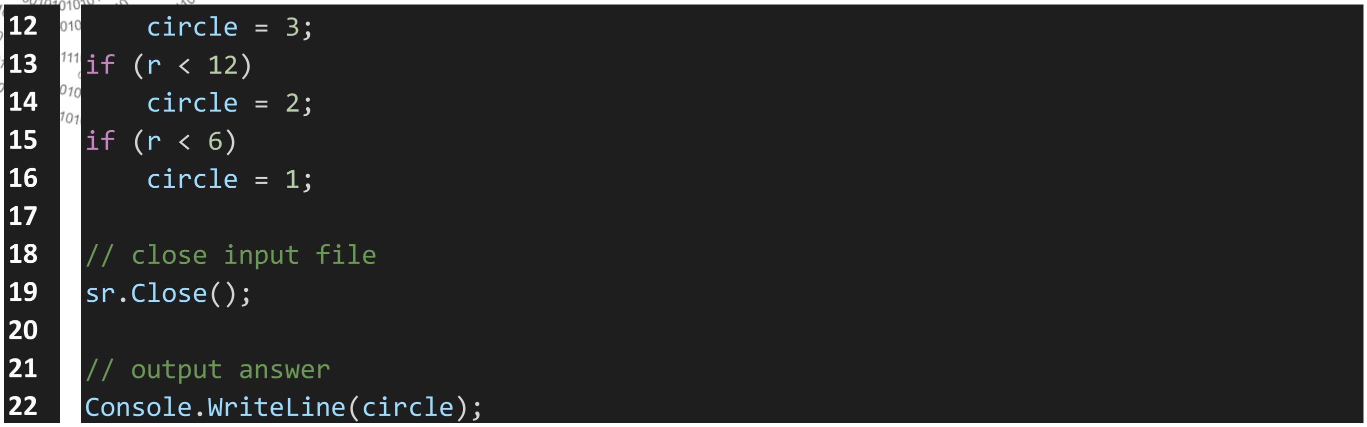
Sample Output 4:

```
1
```

All we need to do for this question is read in the file, calculate the equation of the radius and compare its value.

Here is the solution for this question:

```
// open input file 'input.txt'
StreamReader sr = new StreamReader("input.txt");

// get coordinate
int x = Convert.ToInt32(sr.ReadLine()); // read first line and convert to integer
int y = Convert.ToInt32(sr.ReadLine()); // read second line and convert to integer

// determine which cricle the coordinate (x, y) is in
double r = Math.Pow(x, 2) + Math.Pow(y, 2);
int circle = 0;
if (c < 20)
```



```
12      circle = 3;
13  if (r < 12)
14      circle = 2;
15  if (r < 6)
16      circle = 1;
17
18  // close input file
19  sr.Close();
20
21  // output answer
22  Console.WriteLine(circle);
```

First, we open the input file **(line 2)**, then we read in the number of IDs that exist in this input file **(line 5).** This number is stored in a variable ‘n’ and be used in our for-loop condition to loop ‘n’ times.

Line 9: We calculate the radius of the circle ‘r’

Line 10 to 16: We determine which circle the dart hit by comparing the radius ‘r’ to the numbers indicated in the table above in the problem description. Notice that our if-statements are not if-else-statements due to the nature of what is being asked (i.e., our program is setup so that we test every circle).

Once the for-loop completes we close the input file **(line 19)** and output our number of stops **(line 39)**.

Note: We did not have to use any loops for this question!

Problem 2: ID Priority

Program Description:

Consider an online lottery system where a user can submit an entry to win a prize. A user's entry is identified by an ID number. An ID number consists of a hashtag (#) followed by 3 letters and 3 numbers. Before the draw is processed, the online lottery system will prioritize the IDs for the draw. IDs in the online lottery system are weighted and prioritized based on a point system using the following criteria:

- 10 points added if the 3 letters in the ID are the same
- Added Points equal to the addition of the last 3 digits in the ID
- A higher weight will mean a higher priority

Examples:

- *#bbb345* will have a rating of 22
- #aab998 will have a rating of 26

Therefore, in this case *#aab998* will have a higher priority over *#bbb345*

Input:
The input should come from a file named **"input.txt"**. The first line is an integer N indicating how many IDs will be processed. The following N lines each contain a 7-character ID to be processed (*you can assume that each ID will have a unique weight*).

Output:
The output should be the ID with the highest priority.

Sample Input 1:

```
2
#bbb345
#aab998
```

Sample Output 1:

```
#aab998
```

Sample Input 2:

```
3
#bbb345
#aab998
#ccc989
```

Sample Output 2:

```
#ccc989
```

Sample Input 3:

```
7
#bcd888
#eef987
#ccc865
#ccd865
#eee787
#aaa999
#www777
```

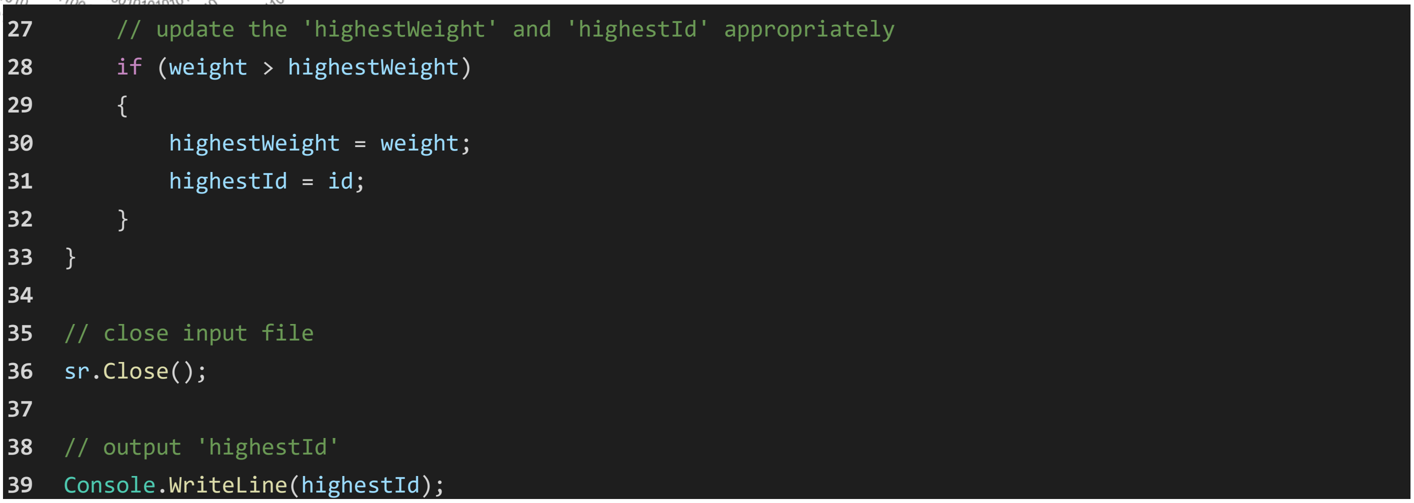
Sample Output 3:

```
#aaa999
```

We can see that for every input above our output is the ID with the highest priority based on the criteria from the problem description.

Here is the solution for this question:

```
// open input file
StreamReader sr = new StreamReader("input.txt");

// get 'n'
int n = Convert.ToInt32(sr.ReadLine());

// declare and set variables that are needed
string? id = "", highestId = "";
int weight = 0, highestWeight = 0;

// loop through all id's in file
for (int x = 0; x < n; x++)
{
    // get current id from file
    id = sr.ReadLine();

    // set weight initially to '0'
    weight = 0;

    // check if the current id has the first the letters equal and set 'weight' to 10
    if (id[1] == id[2] && id[2] == id[3])
        weight = 10;

    // add the values of the last 3 numbers in the 'id' together and add that total to the 'weight'
    weight += Convert.ToInt32(id.Substring(4, 1)) + Convert.ToInt32(id.Substring(5, 1)) + Convert.ToInt32(id.Substring(6, 1));

```



```
27      // update the 'highestWeight' and 'highestId' appropriately
28      if (weight > highestWeight)
29      {
30          highestWeight = weight;
31          highestId = id;
32      }
33  }
34
35  // close input file
36  sr.Close();
37
38  // output 'highestId'
39  Console.WriteLine(highestId);
```

First, we open the input file **(line 2)**, then we read in the number of IDs that exist in this input file **(line 5).** This number is stored in a variable ‘n’ and be used in our for-loop condition to loop ‘n’ times.

Lines 8 & 9: We then setup our variables that will be used within our loop.

On **line 12** our for-loop starts and will cycle for ‘n’ times. **Line 15** we read in the next ‘id’ from the file and on **line 18** we set its current weight to zero.

Lines 21 & 22: We check if the first 3 letters of the current ‘id’ are equal and if they are then our weight equals 10.

Line 25: We add the sum of last 3 digits of our ‘id’ to the ‘weight’

Line 28: This is where we determine if the current ‘id’ has the highest priority. We compare the current ‘weight’ with the ‘highestWeight’ (this will equal ‘0’ on our first iteration). If the ‘weight’ is larger than the ‘highestWeight’ then the current ‘id’ has the highest priority thus far and we update ‘highestWeight’ and ‘highestId’ appropriately.

Once the for-loop completes we close the input file **(line 36)** and output the circle that was hit **(line 39)**.

Problem 3: Light Sync

Program Description:

In certain cities you will notice that streetlights are synchronized on the main streets in a particular direction (e.g., heading east and west). Let's assume that every 20 to 30 seconds the next traffic light turns green. During rush hour you may or may not hit all the green synchronized lights (i.e., traffic, accidents, etc.).

Write a program to determine how many times the driver had to stop while driving through the synchronized lights.

Input:

The input should come from a file named "input.txt" which consists of 3 lines. The first line is an integer N indicating how many synchronized lights the driver is approaching. The second line consists of N integers each separated by a space where each integer represents the amount of time (in seconds) until the next light turns green. The third line consists of N integers each separated by a space where each integer represents the time it took the driver (in seconds) to get to the next light.

You can assume that if the driver reaches a light at the same time the light turns green then the driver does not stop.

Output:

The output should be the number times the driver had to stop.

Sample Input 1:

```
6
22 30 26 27 21 25
18 29 26 29 19 22
```

Sample Output 1:

```
3
```

If we analyze the input, we can see that the driver arrived at the **first light** in 18 seconds, but the light did not turn green for 22 seconds. Therefore, the driver had to stop, so our number of stops will equal 1 at this point.

For the **second light** the driver arrived at the light in 29 seconds, but the light did not turn green for 30 seconds. Therefore, once again, the driver had to stop, so our number of stops will equal 2 at this point.

For the **third light** the driver arrived at the light in 26 seconds, and the light turned green at 26 seconds, therefore we can assume that the driver did not stop (recall that the problem description above says that if both times are the same then no stop is required).

For the **fourth light** the driver arrived at the light at 29 seconds, but the light turned green at 27 seconds. This means that the light synchronization is now 2 seconds ahead of the driver. Therefore, there is no stop required, however, we need to subtract 2 seconds from our next green light time.

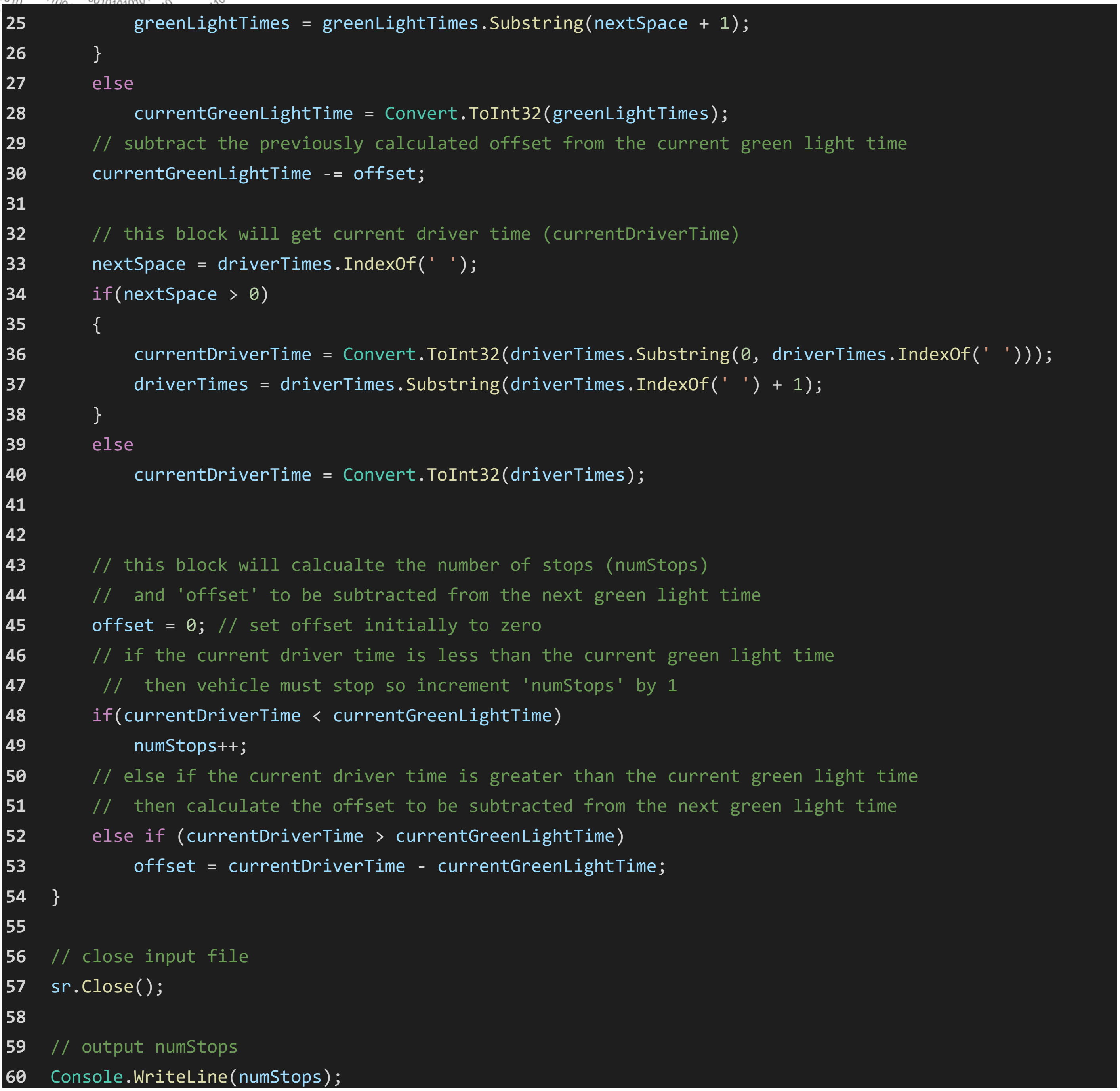
For the **fifth light** the driver arrived at 19 seconds, and the light turns green at 21 – 2 = 19 seconds (remember we must subtract 2 seconds due to the previous light scenario). Therefore, both times are the same and no stop is required.

For the **sixth light** the driver arrives at 22 seconds, but the light turned green at 25 seconds. Therefore, the driver had to stop, so our number of stops will equal 3 at this point.

There are no more lights, so our total number of stops is **‘3’** which is shown in our above **sample output.** The trick to this question is when the driver arrives at a light after the light turns green which means that the green light synchronization will be ahead of the driver’s current time (as was demonstrated at the fourth light above).

Here is the solution code for this question:

```
// open input file 'input.txt'
StreamReader sr = new StreamReader("input.txt");

// get number of lights
int numLights = Convert.ToInt32(sr.ReadLine());

// get lines containing green light times & driver times
string? greenLightTimes = sr.ReadLine();
string? driverTimes = sr.ReadLine();

// initiate variables
int currentGreenLightTime = 0, currentDriverTime = 0;
int nextSpace = 0;
int numStops = 0;
int offset = 0;

// loop through the number of lights
for (int x = 0; x < numLights; x++)
{
    // this block will get the current green light time (currentGreenLightTime)
    nextSpace = greenLightTimes.IndexOf(' ');
    if(nextSpace > 0)
    {
        currentGreenLightTime = Convert.ToInt32(greenLightTimes.Substring(0, nextSpace));
```



```
25          greenLightTimes = greenLightTimes.Substring(nextSpace + 1);
26      }
27      else
28          currentGreenLightTime = Convert.ToInt32(greenLightTimes);
29      // subtract the previously calculated offset from the current green light time
30      currentGreenLightTime -= offset;
31
32      // this block will get current driver time (currentDriverTime)
33      nextSpace = driverTimes.IndexOf(' ');
34      if(nextSpace > 0)
35      {
36          currentDriverTime = Convert.ToInt32(driverTimes.Substring(0, driverTimes.IndexOf(' ')));
37          driverTimes = driverTimes.Substring(driverTimes.IndexOf(' ') + 1);
38      }
39      else
40          currentDriverTime = Convert.ToInt32(driverTimes);
41
42
43      // this block will calcualte the number of stops (numStops)
44      //  and 'offset' to be subtracted from the next green light time
45      offset = 0; // set offset initially to zero
46      // if the current driver time is less than the current green light time
47       //  then vehicle must stop so increment 'numStops' by 1
48      if(currentDriverTime < currentGreenLightTime)
49          numStops++;
50      // else if the current driver time is greater than the current green light time
51      //  then calculate the offset to be subtracted from the next green light time
52      else if (currentDriverTime > currentGreenLightTime)
53          offset = currentDriverTime - currentGreenLightTime;
54  }
55
56  // close input file
57  sr.Close();
58
59  // output numStops
60  Console.WriteLine(numStops);
```

Lines 1 to 9: According to the problem description we know that the input file will contain exactly 3 lines of data. Therefore, the first thing we do is read in the first 3 lines into appropriate variables**: numLights, greenLightTimes,** and **driverTimes.**

Lines 11 to 15: We then setup our variables that will be used within our loop.

Our for-loop starts on **line 18** and we need to loop through the number of traffic lights **(numLights).** We then need to extract the next green light time and driver time. If you look at the input file, you will notice the times for each are separated by spaces. Therefore, **lines 20 to 41** we do some string manipulation to get the next green light time and driver time.

Lines 44 to 54: This is the bulk of where problem solving begins. We know if the current driver time is less than the current green light time then we must stop and therefore we increment **numStops** by 1. If the current driver time is greater than the current green light time then the driver did not stop, but it also means that the next green light has already started. Therefore, we calculate a variable named **'offset'** to equal the current driver time minus the current green light time. This **'offset'** can then be subtracted from the next green light time on next for-loop iteration **(line 30).**

Once the for-loop completes we close the input file **(line 57)** and output our number of stops **(line 60)**.