Counting Turns in a Maze J. La Favre June, 2026

The image to the right is the maze pattern for the 2025 NRC Robot Maze Contest, Middle School Division. Your assignment this year is to program the ActivityBot360 to solve the maze. In previous lessons you learned how to code for initiating a turn using odometry and also by using a Ping sensor. There remains one problem to solve: deciding which way to turn, left or right.

Let us start by analyzing the required turns. There are thirteen 90 degree turns in the path between the start and finish, seven right turns and six left turns. For twelve of the turns, there is only one way to turn, either left or right. The last turn is at a T-shaped intersection, where the robot could turn left or right. At this last turn the robot must turn right.

For eleven turns there is a wall directly in front of the robot in the cell where the turn is required. For these turns it would be possible to specify a distance to the front determined by Ping sensor to trigger the turn. However, turns 7 and 12 do not have a wall in front in the cell where the turn must occur. A different distance to front must be used to trigger the turn. How can we solve the problem of different distances to the front wall depending on the turn?

One solution to the problem is for the robot to count its turns. If you want to trigger a turn either by odometry or Ping sensor, then the robot must know which turn it is located at to use the proper distance to travel (odometry) or proper distance to front wall.

First determine the sequence of turns (see table on next page)

Turn number	Direction of turn
1	Left
2	Left
3	Right
4	Left
5	Right
6	Right
7	Right
8	Left
9	Right
10	Left
11	Left
12	Right
13	Right

Now you need to develop code for counting turns. For this lesson you will code for only the first two turns.

Make sure your code includes the libraries **simpletools.h, abdrive360.h** and **ping.h**:

```
#include "simpletools.h"
#include "abdrive360.h"
#include "ping.h"
```

Create variables to store the turn number and distance measurements:

```
int turnNumber = 1; // when program starts the robot approaches turn number one
int distR;          // distance to right
int distF;          // distance to front
```

Then add the main() function:

```
int main()
{
  // all other code goes here
} // end of main()
```

In between the open and close curly { } braces, add your drive code as provided on the next page;

```
while (1)             //run this loop forever
{
distR = ping_cm(16); // make a ping measurement in cm to the right (port 16)
pause(20);            // wait between measurements to prevent crosstalk
distF = ping_cm(17); // make a ping measurement in cm to the front (port 17)
if (turnNumber == 1)       // turn #1 is a left turn
  {
  if (distF < 19)          // if distance to front is less than 19 cm
    {                            // make a left turn
    drive_speed (27, 64);        // this makes robot turn left
    pause (1350);                // robot turns left for 1350 milliseconds,
                                 // about 90 degree turn
    turnNumber = turnNumber + 1;      // add 1 to value of turnNumber
    }                                 //end of if (distF < 19)
  else                       // robot is not yet in position to make turn
    {
    drive_speed (64, 64);        // drive forward at speed 64 both wheels
    pause (20);                  // wait 20 milliseconds
    }                            //end of else
  }                              //end of if (turnNumber == 1)
else if (turnNumber == 2)        // turn #2 is a left turn
  {
  if (distF < 19)          // if distance to the front is less than 19 cm
    {                            // make left turn
    drive_speed (27, 64);        // this makes robot turn left
    pause (1350);                // robot turns left for 1350 milliseconds,
                                 // about 90 degree turn
    turnNumber = turnNumber + 1;      // add 1 to value of turnNumber
    }                                 // end of if (distF < 19)
  else                       // robot is not yet in position to make turn
    {
    drive_speed (64, 64);        // drive forward at speed 64 both wheels
    pause (20);                  // wait 20 milliseconds
    }                            //end of else
  }                              // end of else if (turnNumber == 2)
else                             // after turns one and two have completed
  {
  break;                         // escape the while loop
  }                              //end of else
}                                // end of while loop

drive_speed (0, 0);              // stop the robot
```

Let us take a look at the line: **if (turnNumber == 1)** Notice that two equal signs are used. Why? Well, let us take a look at another line of the code: **int turnNumber = 1;** Here we see that only one equal sign is used. The use of one equal sign is used to **assign** a value. But in the line **if (turnNumber == 1)** we don't want to assign a value. We are asking the question "*is the value of turnNumber equal to one*?" Since one equal sign is used to assign a value, in C language, if you want to ask if a variable is equal to a value, you MUST USE TWO EQUAL SIGNS.

Now let's take a look at the decision structure used in the code. Inside the while loop there is an **if-else if-else** structure:

```
if (turnNumber == 1)
else if (turnNumber == 2)
else
```

This scheme divides the drive code into three sections: **1)** drive code for driving up to and making the first turn, **2)** drive code for driving up to and making the second turn and **3)** drive code to execute after both turns have been completed

Below is the complete code for the first section:

```
if (turnNumber == 1)            // turn #1 is a left turn
  {
  if (distF < 19)               // if distance to front is less than 19 cm
    {                           // make a left turn
    drive_speed (27, 64);       // this makes robot turn left
    pause (1350);               // robot turns left for 1350 milliseconds,
                                // about 90 degree turn
    turnNumber = turnNumber + 1;      // add 1 to value of turnNumber
    }                                 //end of if (distF < 19)
  else                     // robot is not yet in position to make turn
    {
    drive_speed (64, 64);       // drive forward at speed 64 both wheels
    pause (20);                 // wait 20 milliseconds
    }                           //end of else
  }                             //end of if (turnNumber == 1)
```

There is an **if-else** decision structure nested inside this first section of code:

```
if (distF < 19)
else
```

If the distance to the front is less than 19 cm, then there is a wall close to the front of robot. It is time to initiate the turn. The code directly underneath the **if** codes for a left turn of approximately 90 degrees, After the turn is completed, a value of one is added to the turn-counting variable. From this point on, the code inside if (turnNumber == 1) will not execute because the value of turnNumber is now greater than a value of one.

As the code jumps back to the top of the while loop, now the value of **turnNumber** is equal to two. Thus, the next section of the code will execute:

```
else if (turnNumber == 2)       // turn #2 is a left turn
  {
  if (distF < 19)          // if distance to the front is less than 19 cm
    {                            // make left turn
    drive_speed (27, 64);        // this makes robot turn left
    pause (1350);                // robot turns left for 1350 milliseconds,
                                 // about 90 degree turn
    turnNumber = turnNumber + 1;      // add 1 to value of turnNumber
    }                                 // end of if (distF < 19)
  else                       // robot is not yet in position to make turn
    {
    drive_speed (64, 64);        // drive forward at speed 64 both wheels
    pause (20);                  // wait 20 milliseconds
    }                            //end of else
  }                              // end of else if (turnNumber == 2)
```

The second code section is essentially the same as the first section since we are again using the front sensor to determine when to turn left a second time.

```
if (distF < 19)
```

If the distance to the wall in front of the robot is LESS than 19 centimeters, then the robot will turn 90 degrees to the left. After the second left turn is completed, the value of turnNumber will be three. Therefore, the third code section executes:

```
else                   // after turns one and two have completed
  {
  break;               // escape the while loop
  }                    //end of else
}                      // end of while loop

drive_speed (0, 0);    // stop the robot
```

The statement **break** causes the program to escape out of the while loop. Just underneath the close brace } for the while loop, the line **drive_speed (0, 0);** causes the robot to stop.

Test your robot in the maze to see how well it navigates the first two turns of the maze. The robot should be able to make the turns. However, it is likely that the turns could be improved. I found that the robot turns a little more than 90 degrees for each turn. If your robot is turning more than 90 degrees, reduce the number of milliseconds for the turn (less than 1350). If your robot is turning less than 90 degrees, increase the number of milliseconds.

You may also find that your robot does not execute the turns at the best position. It may turn a little early or late. Try editing your code to make the robot turn at a better position.

Once you have your robot executing the first two turns, well-centered in the lanes, then you have completed this lesson.

File name: Counting_Turns_in_a_Maze.pdf
online location: https://www.lafavre.us/robotics/Counting_Turns_in_a_Maze.pdf
last update: 6/25/26