Let us look at the maze plan again in Figure 1. The first three turns are left turns of 90 degrees each. The second and third turns combined are one turn of 180 degrees, but let us think of them as two 90 degree turns. After the three left turns, the robot must turn right. In this lesson, you will add code for right turns.

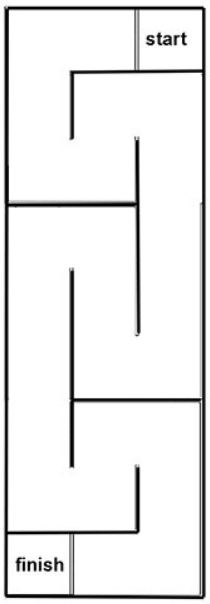

Figure 1 National Robotics Challenge maze - elementary and middle school contestants

We should review the code structure you have completed up to this point (just the while(1) loop):

```
while(1)
   {
   int wallDistR = ping_mm(17); // take measurement to
                                // right wall
   int wallDistF = ping_mm(16); //take measurement to
                                //front wall
   if(wallDistF < ___)          //robot turn left at
                                // corner
      {
         drive_ramp(__, 64);
         pause(____);
      }
   else if(wallDistR < 118)      //turn slightly left
      {
         drive_ramp(__, 64);
      }
   else if(wallDistR > 128)      //turn slightly right
      {
         drive_ramp(64, __);
      }
   else                          //drive straight forward
      {
         drive_ramp(64, 64);
      }
   pause(20);
   }                            }
```

As the code stands now, the robot always turns left when it approaches a wall in front. Therefore, the robot will turn left even when it should turn right. We need to change that! What is different about the wall configuration at a right turn compared to a left turn? As the robot passes the end of the wall on the right, the distance to a wall on the right increases by the amount of 15.75 inches (the width of the maze corridor plus the thickness of a ¾-inch wall). We can use this change in distance on the right side of the robot as the event to trigger a turn to the right. We should place this part of the code as the first item in the **if-else if-else** structure. Why? Because we do not want the code for turning left to activate if a right turn is needed. By placing the right turn code before the left turn code, we can help insure the robot will turn right when needed. However, we must be sure that the robot sees an increase in distance to the right before the distance to front drops below the value where a left turn is triggered. You may have to do some trial and error in adjusting the trigger value for the front wall.

Alter your code by changing the **if()** statement to an **else if()** statement for the left turn. Above this add an **if()** statement for the right turn. The **if()** and first **else if()** in your code should then appear as below. You must select your value carefully for this line:

```
if(wallDistR > ___)
```

Place the robot in a corridor of the maze, up against the wall on the left. Then measure the distance in millimeters from the right-facing sensor to the right wall. This is the maximum distance the robot can be from a right wall while in a corridor. Your trigger value to turn right should be greater than the value for maximum distance to right wall while in a corridor. Just to be safe, it is a good idea to make your trigger value somewhat larger than the maximum value to right wall in corridor. However, if you make the value too large, the robot will not turn right as needed.

```
if(wallDistR > ___)           //robot turn right at corner
      {
         drive_ramp(64, __);  // enter value for right wheel
         pause(____);         // enter value for pause
      }

else if(wallDistF < ___)      //robot turn left at corner
      {
         drive_ramp(__, 64);
         pause(____);
      }
```

You now have a complete code structure for solving the maze. Run your robot through the maze. If it hits any walls, then you will need to adjust the values in your code. Watch the behavior of the robot as it runs through the maze. Where does it start to get in trouble? How can you alter its behavior?

Once you have your robot consistently running the maze without hitting a wall, you should increase the speed of your robot. Remember that the NRC objective for the maze contest is to run it as fast as possible. Using a speed value of 64 for your wheels in drive_ramp results in the robot running at 50% of maximum speed. Change your code so that the robot runs at maximum speed (value of 128 for drive_ramp). Once you have your robot running the maze at maximum speed without hitting any walls, you have completed this lesson. You now have a robot program for the NRC! However, you may wish to refine your code some more. As it stands now, your robot does not stop at the end of the maze. You must grab it after it crosses the finish line so that it will not crash into the front wall.

Think about how you can alter your program to make the robot stop automatically at the end of the maze. One way to do this is to make the robot count each turn that it makes. That way it will know when it has made the last turn in the maze. Then have it stop when it reaches a certain distance before the front wall. Try to add the necessary code to make your robot stop.

Here is a hint. Make a variable for counting the turns. Initialize the variable with a value of zero. Then add a line of code after the pause for the right and left turns. That line should add a value of one to your variable. When the variable has reached a value equal to the number of turns in the maze, the robot will know it has made its last turn. All that is needed then is to stop after crossing the finish line.