Intro to Sensors

- 4 types of sensors available in the standard kit

Programming Sensors

- Sensors return a varying range of values
  - Touch Sensors return a value of 0 or 1
  - Light and Sound Sensors return a value between 0 and 100
  - Sonar Sensors return a value in cm, up to 255cm.
- One function in ROBOTC returns this value for you to use in your program
  - **SensorValue[*sensorName*];**
  - **SensorValue(*sensorName*);**
- Another function works in the same manner for encoder counts.
  - **nMotorEncoder[*motorName*];**

Touch Sensor

- Digital Sensor
  - Returns either 0 or 1
- Useful for…
  - Detecting touches
  - Acting as a limit switch
  - User interfaces to robot

Configuring Sensors

- Live Demo: Motors and Sensor Setup
  - Opening Motors and Sensor Setup
  - Configuring Sensors
  - Getting feedback from a Touch Sensor

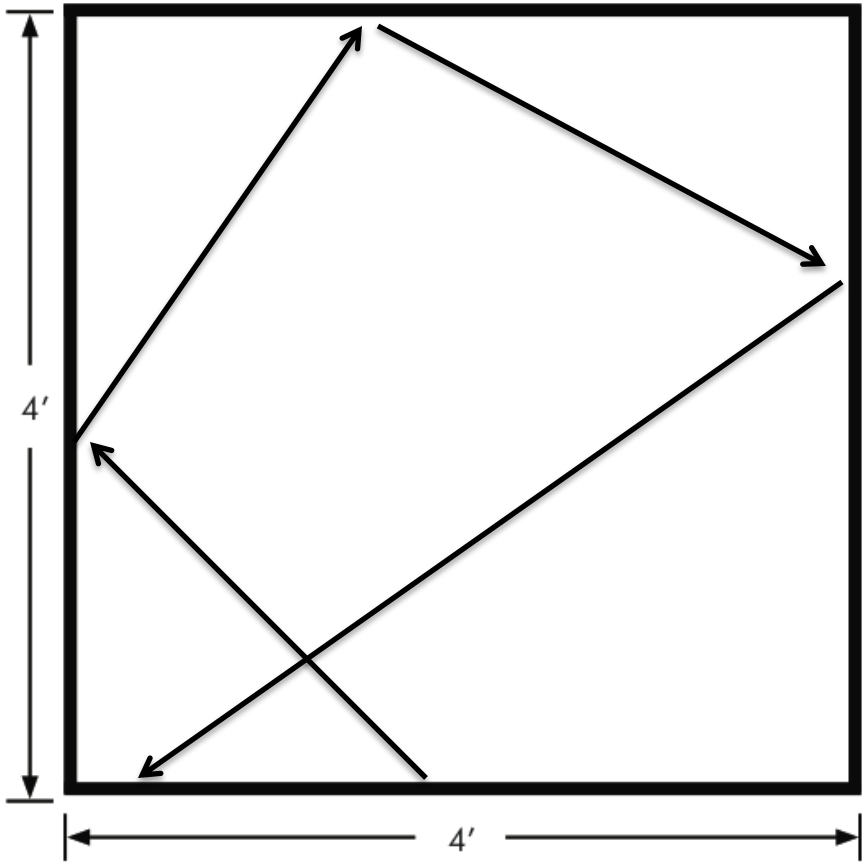
```
#pragma config(Sensor, S1,     touch1,              sensorTouch)
//*!!Code automatically generated by 'ROBOTC' configuration wizard

task main()
{
  SensorValue[touch1];
}
```

While Loop & Boolean Logic

- Self Paced Lesson
  - While Loop & Boolean Logic
- Watch the following 3 lesson videos:
  - Sensing – Wall Detection (Touch) – Touch vs. Timing
  - Sensing – Wall Detection (Touch) – The While Loop
  - Sensing – Wall Detection (Touch) – Boolean Logic Pt.1

Challenge: RoboMower

- Complete the ”RoboMower Challenge”
  - First “flowchart” or plan your program
  - Then program your robot
- If finished early…
  - Take look at **“Random Numbers”** under the **“Wall Detection (Ultrasonic)”**
  - **Make your turns be random!**
  - Think… what commercial products work in a similar manner?



While Loops

- A while loop is a structure within ROBOTC which allows a portion of code to be run over and over, as long as the specified Boolean condition remains “true”.

```
task main()
{
 while(nMotorEncoder[motorC]<360)
 {
   motor[motorC]=100;
   motor[motorB]=100;
 }
}
```

The condition is true as long as the rotation sensor detects less than 360 degrees of rotation.

While the condition is true, both motors will receive 100% power.

While Loops

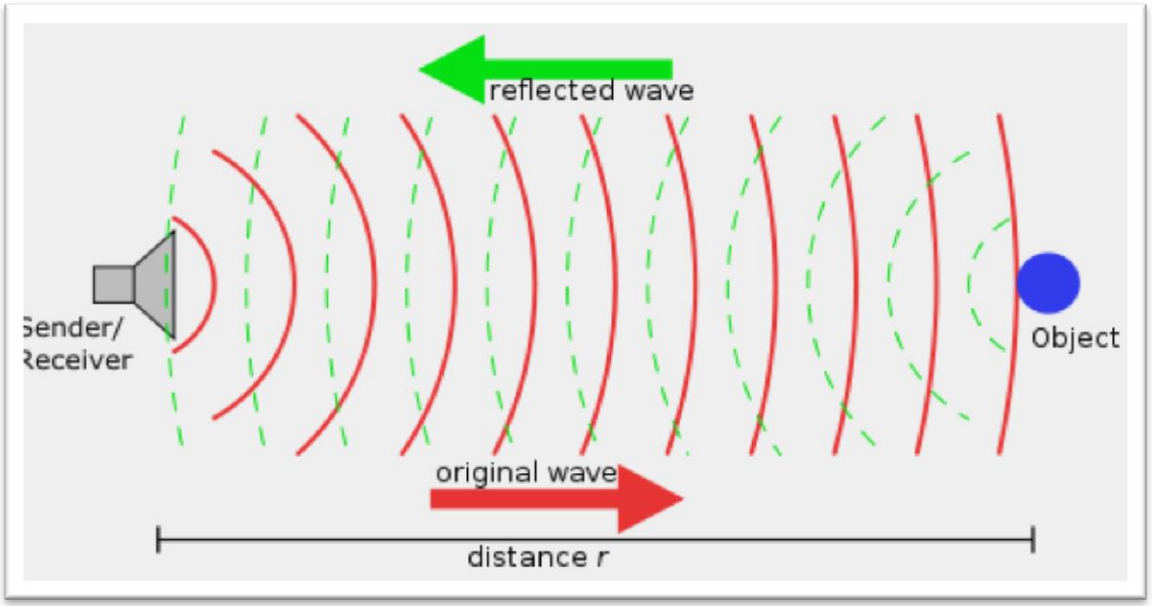
- Things to avoid with “while loops”
  - Having a condition that could never be true
    - Example: while(SensorValue[touch1] < 0)
  - Using a semicolon
    - Example: while(SensorValue[touch1] == 0);
    - This code will make an “idle” loop
      - i.e. a loop with no code
  - Not using curly braces
    - While the code will still compile, it will be very difficult to track what is in the loop and what isn’t.

Sonar Sensor

- I²C Sensor
  - Returns a value between 0 and 255
  - Value returned is number of cm
- Useful for…
  - Detecting flat objects
  - Measuring distances

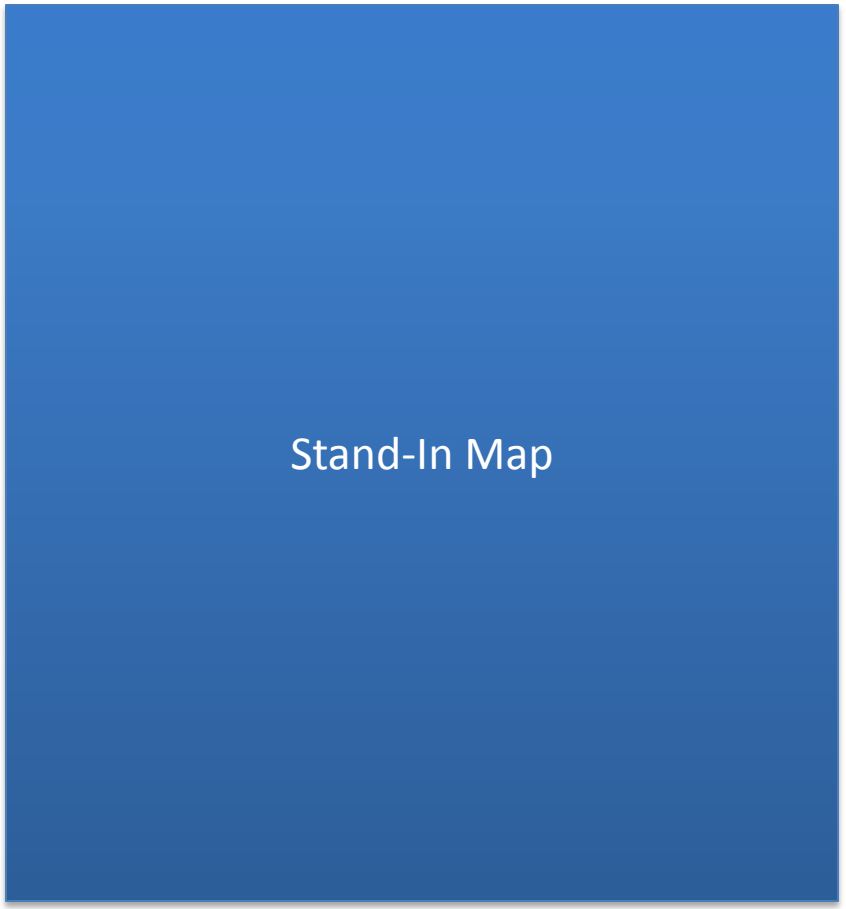
Sonar Sensors

- Self Paced Lesson
  - Learning about Sonar Sensors
- Watch the following lesson video:
  - Sensing – Wall Detection (Ultrasonic) – A Sonic Sojourn



Challenge: Sonar Maze

- Complete the ”Sonar Maze Challenge”
  - First “flowchart” or plan your program
  - Then program your robot
- If finished early…
  - Make sure your code is commented
  - Think… what else do you know of that uses sonar?
- Note:
  - If your wires/other sensors are interfering with your reading, feel free to remove them.

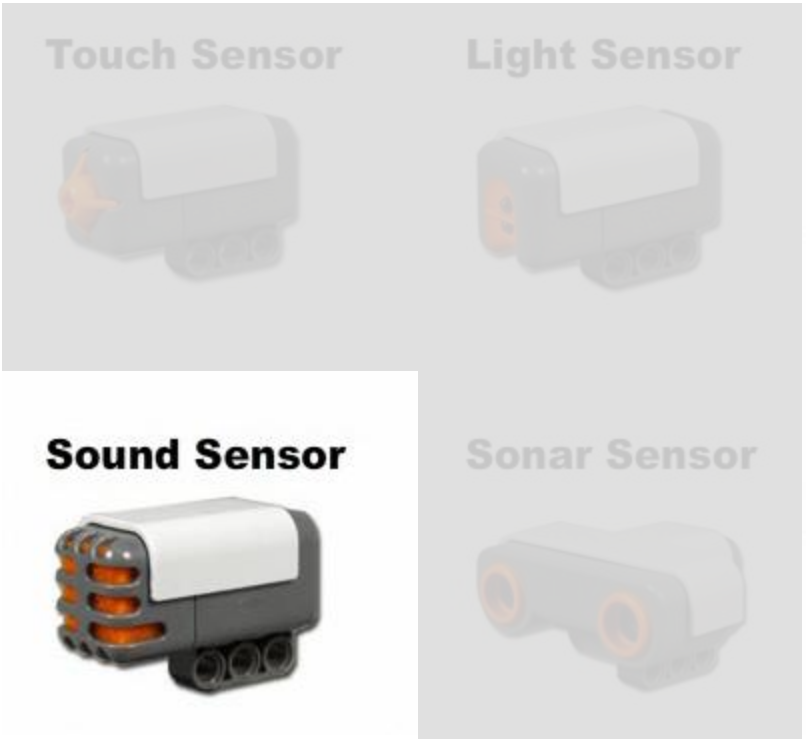


Sonar Sensors

- Debriefing
  - How does a sonar sensor work?
  - Sonar sensors are not very effective on round objects
    - The “echo” can’t return back to the sonar sensor very well
  - Multiple sonar sensors in the same area can cause “cross-talk”
    - They could interfere and produce random results
  - How could you convert the cm to inches?
    - Divide by 2.54!
    - (SensorValue[sonar4] / 2.54)

Sound Sensor

- Analog Sensor
  - Returns a value between 0 and 100
- Useful for…
  - Detecting Volume of Sounds
- Not really useful for TETRIX
  - The sound of the DC motors will drown out the Sound Sensor



Sound Sensors

- Self Paced Lesson
  - Learning about Sound Sensors
- Watch the following lesson videos:
  - Sensing – Volume & Speed – Values and Assignments Pt. 1
  - Sensing – Volume & Speed – Values and Assignments Pt. 2
- If you finish early…
  - Move the sound sensor closer to the motors and see what happens

Light Sensor

- Analog Sensor
  - Returns a value between 0 and 100
- Useful for…
  - Detecting reflect light
  - Detecting ambient light
  - Detecting changes in surfaces (dark vs. light)
  - Following Lines

Light Sensors

- Self Paced Lesson
  - Introduction to Light Sensors and Thresholds
- Watch the following 3 lesson videos:
  - Sensing – Forward Until Dark – The Light Sensor
  - Sensing – Forward Until Dark - Thresholds 201
  - Sensing – Forward Until Dark – Wait for Dark
- If you finish early…
  - Think about how you could use variables and math when working with thresholds.

Light Sensors

- Debriefing
  - Thresholds are the most important thing!
    - Every environment that you will be in will cause a different threshold value
  - Distance away from an object is the second most important thing!
    - If the light sensor is 1cm away from an object, the threshold will be very different if the light sensor becomes 3cm away.
  - Light Sensors can be used without the red LED
    - Set your sensor type to “Light Inactive”
    - This will make the light sensor a passive light sensor, good for detecting room brightness.

Light Sensors

- Great Abstraction Bridge for Thresholds
  - In View Mode (with the reflected light setting), your robot shows a value of 63 over the white area and 37 over the black line. What value should you use as the threshold in your while loop?
  - Abstraction: Philip has yardstick with equal sized weights attached by string to the 7 inch mark and 29 inch mark. At which inch mark should Philip place his finger in order to balance the yardstick?

Advanced Control with Sensors

- Self Paced Lesson
  - Line Tracking with Light Sensor
- Watch the following 2 lesson videos:
  - Sensing – Line Tracking – Line Tracking (Basic)
  - Sensing – Line Tracking – Line Tracking (Better)
  - Watch both videos before asking questions!
- If you finish early…
  - Look up “Conditional (computer programming)” on Wikipedia

If/Else Statements

- An if-else Statement is one way you allow a computer to make a decision.
  - With this command, the program will check the (condition) and then execute one of two pieces of code, depending on whether the (condition) is true or false.
- If/Else statements typically need a while loop as well!
  - Otherwise, the If/Else statement will only be checked once and the program will continue onwards.

If/Else Statements

```
task main()
{
 while(true)
 {
    if(SensorValue(sonarSensor)>25)
    {
      motor[motorC]=100;
      motor[motorB]=100;
    }

    else
    {
      motor[motorC]=0;
      motor[motorB]=0;
    }
 }
}
```

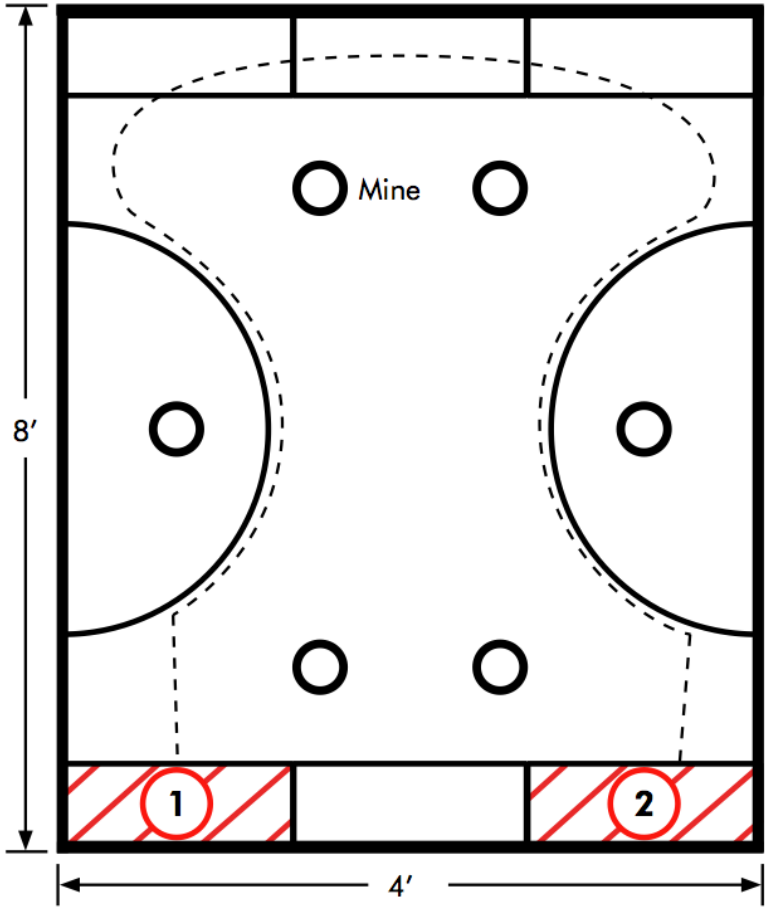
(condition)
true if the sensor reads over 25
false otherwise

(true) commands
These commands run if the (condition) is **true.**

(false) commands
These commands run if the (condition) is **false.**

Sensors Final Challenge

- Watch Line Tracking Lessons #1-#5
  - Line Tracking (Basic & Better)
  - Line Track with Timing
  - Line Track for Distance (2 parts)
- Complete the ”Minefield Challenge”
  - **First “flowchart” or plan your program**
  - Then program your robot
- If finished early…
  - Take a look at “Forward for Distance” under the Sensing Section in the Curriculum
  - Lesson #1 is the method for using encoders in the Virtual Worlds
  - Lesson #2/#3 is the method for removing the “wait” time with Targets.



Note: Diagrams are not drawn to scale

Timers

- Timers are very useful for performing a more complex behavior for a certain period of time.
  - Wait statements (wait1Msec) do not let the robot execute commands while waiting period
- Timers allow you to track the amount of elapsed time while having other code run in your program.

Timers

```
task main()
{
 ClearTimer(T1);
 while(time1[ T1]  < 3000)
 {
    Code that will loop for 3 seconds
    (3000 1ms increments)
 }
```

Clear the Timer
Clearing the timer resets and starts the timer. You can choose to reset any of the timers, from T1 to T4.

Timer in the (condition)
This loop will run "while the timer's value is less than 3 seconds", i.e. **less than 3 seconds have passed since the reset.** The line tracking behavior inside the {body} will continue for 3

Timers

- First, you must reset and start a timer by using the ClearTimer() command. Here's how the command is set up:
  - ClearTimer(Timer_number);
  - ROBOTC has 4 built in timers: T1, T2, T3, and T4.
- Then, you can retrieve the value of the timer by using…
  - time1[T1] – Returns the number of 1ms increments that have elapsed.
  - time10[T1] – Returns the number of 10ms increments that have elapsed.
  - time100[T1] – Returns the number of 100ms increments that have elapsed.

ting Code

- How far will the motors travel?

```
 4  task main()
 5  {
 6    nMotorEncoder[motorB] = 0;
 7    nMotorEncoder[motorC] = 0;
 8
 9    nMotorEncoderTarget[motorB] = 360;
10    nMotorEncoderTarget[motorC] = 360;
11
12    motor[motorB] = 50;
13    motor[motorC] = 50;
14
15    wait1Msec(5000);
16
17    while(true)
18    {
19      if(nMotorEncoder[motorB] < 540)
20      {
21        motor[motorB] = 50;
22        motor[motorC] = 50;
23      }
24      else
25      {
26        motor[motorB] = 0;
27        motor[motorC] = 0;
28      }
29    }
30  }
```

Variables

- What is a variable?
  - A variable is a facility for storing data in a program.
- How do they work?
  - When a variable is “declared”, the compiler sets aside a piece of memory to store the variable’s numeric value.
  - When the variable is called, the processor “retrieves” the numeric value of the variable and “returns” the value to be used in your program in that specific location.

Variables

- How do I create a variable?
  - To create ("declare") a variable you need 3 things.
    - Decide the data type of the variable (more on this soon)
    - Give the variable a name
    - Assign the variable a value (optional, but recommended)
  - Example:
    - **int myVariable = 1000;**
    - Type: Integer
    - Name: myVariable
    - Value: 1000

Data Types

- There are 8 different types of variables in ROBOTC
  - Integer (**int**) – Memory Usage: 16 bits / 2 bytes
    - Integer Numbers Only
    - Ranges in value from -32768 to +32767
  - Long Integer (**long**) – Memory Usage: 32 bits / 4 bytes
    - Integer Numbers Only
    - Ranges in value from -2147483648 to +2147483647
  - Floating Point (**float**) – Memory Usage: 32 bits / 4 bytes
    - Integer or Decimal Numbers
    - Variable precision, maximum of 4 digits after decimal

Data Types

- There are 8 different types of variables in ROBOTC
  - Single Byte Integer (**byte**) – Memory Usage: 8 bits/1 byte
    - Integer Numbers Only
    - Ranges in value from -128 to +127
  - Unsigned Single Byte Integer (**ubyte**) – Memory Usage: 8 bits / 1 byte
    - Integer Numbers Only
    - Ranges in value from 0 to +255
  - Boolean Value (**bool**) – Memory Usage 4 bits / .5 bytes
    - True (1) or False (0) values only.

Data Types

- There are 8 different types of variables in ROBOTC
  - Single Character (**char**) - Memory Usage: 8 bits / 1 byte
    - Single ASCII Character only
    - Declared with apostrophe – **'A'**
  - String of Character (**string**) - Memory Usage: 160 bits / 20 bytes
    - Multiple ASCII Characters
    - Declared with quotations – **"ROBOTC"**
    - 19 characters maximum per string (NXT Screen limit)

Variables

- Some additional notes
  - Adding “const” in front of a variable will make that variable a constant.
    - This will prevent the variable from being changed by the program
  - Constants do not take up any memory on the NXT
  - The NXT has room for 15,000 bytes of variables

Variables

- Some additional notes
  - Variable's names must follow a specific rules:

Rules for Variable Names

- A variable name can not have **spaces** in it
- A variable name can not have **symbols** in it
- A variable name can not **start with a number**
- A variable name can not be the same as an existing **reserved word**

Proper Variable Names	Improper Variable Names
linecounter	line counter
threshold	threshold!
distance3	3distance
timecounter	time1[T1]

Using Variables

- Variables can be used in your program anywhere!
  - Motor Speeds, If/Else Loops, Conditional Statements
- Commands you know act just like variables
  - nMotorEncoder – Returns the value of a motor encoder
  - SensorValue – Returns the value of a sensor value
- Variables are just numbers
  - You can perform math operations on variables
  - **newVariable = oldVariable + 15;**

Using Variables

```
task main()
{
  motor[motorB] = 50;
  motor[motorC] = 50;
  wait1Msec(3000);
}
```

VS.

```
task main()
{
  int speed = 50;
  int waitTime = 3000;

  motor[motorB] = speed;
  motor[motorC] = speed;
  wait1Msec(waitTime);
}
```

Using Variables

```
1  task main()
2  {
3    motor[motorB] = 50;
4    motor[motorC] = 50;
5    wait1Msec(3000);
6
7    motor[motorB] = 50;
8    motor[motorC] = 50;
9    wait1Msec(3000);
10
11   motor[motorB] = 50;
12   motor[motorC] = 50;
13   wait1Msec(3000);
14
15   motor[motorB] = 50;
16   motor[motorC] = 50;
17   wait1Msec(3000);
18
19   motor[motorB] = 50;
20   motor[motorC] = 50;
21   wait1Msec(3000);
22
23   motor[motorB] = 50;
24   motor[motorC] = 50;
25   wait1Msec(3000);
26 }
```

VS.

```
1  task main()
2  {
3    int speed = 60;
4    int waitTime = 2500;
5
6    motor[motorB] = speed;
7    motor[motorC] = speed;
8    wait1Msec(waitTime);
9
10   motor[motorB] = speed;
11   motor[motorC] = speed;
12   wait1Msec(waitTime);
13
14   motor[motorB] = speed;
15   motor[motorC] = speed;
16   wait1Msec(waitTime);
17
18   motor[motorB] = speed;
19   motor[motorC] = speed;
20   wait1Msec(waitTime);
21
22   motor[motorB] = speed;
23   motor[motorC] = speed;
24   wait1Msec(waitTime);
25
26   motor[motorB] = speed;
27   motor[motorC] = speed;
28   wait1Msec(waitTime);
29 }
```

Repeating Code…

- Copy and pasting only works so well.
- What if we could make each behavior one line of code?

```
1  task main()
2  {
3    int speed = 60;
4    int waitTime = 2500;
5
6    motor[motorB] = speed;
7    motor[motorC] = speed;
8    wait1Msec(waitTime);
9
10   motor[motorB] = speed;
11   motor[motorC] = speed;
12   wait1Msec(waitTime);
13
14   motor[motorB] = speed;
15   motor[motorC] = speed;
16   wait1Msec(waitTime);
```

Functions

- What is a function?
  - A function (or subroutine) is a portion of code within a larger program, which performs a specific task and is relatively independent of the remaining code.
- How does a function work?
  - A function has to be first “declared” in your program, with code inside of the function.
  - Once the function is created and “declared” it can then be “called” from task main or another function to be executed.

Creating Functions

- Set the “type” of function by declaring the “data type” of the function.
  - Void is a special data type which means no value will be returned
- Give the function a name.
  - Following the same rules that variable have!
- Add a set of parenthesis and curly braces
  - Parenthesis are used for “parameters” – We’ll cover this soon.
  - Curly braces define the beginning and end of the function.

```
void movingForward()
{
  motor[motorB] = 50;
  motor[motorC] = 50;
  wait1Msec(3000);
}
```

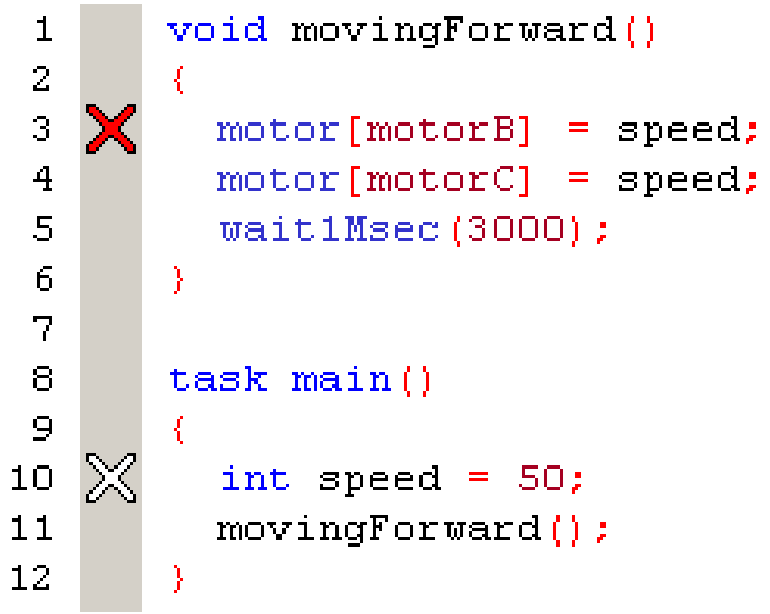
Using Functions

```
void movingForward()
{
  motor[motorB] = 50;
  motor[motorC] = 50;
  wait1Msec(3000);
}

task main()
{
  movingForward();
}
```

- Once the function is created, “call” the function inside of task main by referencing the name and passing any parameters.
- Don’t forget your semicolon
- Note: Keep functions above task main, or else you will have to “prototype your function”.

Functions and Variables

- Key Concept: Variable "Scope"
- Variables are "local" to where they are declared.
- Just because "speed" exists in "task main" doesn't mean it can be used in a function.
- "speed" is currently localized to only "task main"



Functions and Variables

- Solution #1 – “Globalization”
  - Setting the variable outside any function will cause it to become a “global” variable.
  - This is not an ideal solution because multiple functions can use and modify this variable!
  - Use sparingly, but don’t be afraid to use it…

```
1   int speed = 50;
2
3   void movingForward()
4   {
5     motor[motorB] = speed;
6     motor[motorC] = speed;
7     wait1Msec(3000);
8   }
9
10  task main()
11  {
12    movingForward();
13  }
```

Functions and Variables

- Solution #2 –
  "Passing Values"
  - Instead of using the variable, we can just pass the value as a parameter to our function instead.
  - This method is ideal because it gives you flexibility in your functions.
  - This requires us to edit our existing functions, however.

```
void movingForward(int speed)
{
  motor[motorB] = speed;
  motor[motorC] = speed;
  wait1Msec(3000);
}

task main()
{
  movingForward(50);
}
```

Functions and Variables

Variables are now declared inside of the parameter field of the function. Multiple variables are separated by a comma.

```
void movingForward(int speed, int waitTime)
{
  motor[motorB] = speed;
  motor[motorC] = speed;
  wait1Msec(waitTime);
}

task main()
{
  movingForward(50,4000);
  movingForward(30,2000);
}
```

Variables are used inside of the function, but are localized to this function only.

Each time the function is called a different value can be passed. This promotes code flexibility and reuse!

Other Function Types

- Functions don’t always have to be a “void” function.
  - You can use any data type that you would assign to a variable! – int, float, bool, etc.
  - A special command “return” is required to return a value back to the parent function.

Other Function Types

Instead of "void" this function uses "int". This tells us that this function will return an integer result.

```
int addTen(int myValue)
{
  myValue = myValue + 10;
  return myValue;
}

task main()
{
  int starting = 30;
  int result = 0;
  result = addTen(starting);
}
```

After we add 10 to the parameter value that was passed to use by task main, we send the new value back by using the "return" command.

We assign the returned value to a variable so we can use the Debugger to verify the correct value was returned.

Global Variables Debugger

```
task main()
{
  int MyInt = 12345;
  long MyLong = 1234567890;
  float MyFloat = 3.1415;
  byte MyByte = 127;
  ubyte MyuByte = 255;
  bool MyBool = true;
  char MyChar = 'C';
  string MyString = "ROBOTC IS GREAT!";
}
```