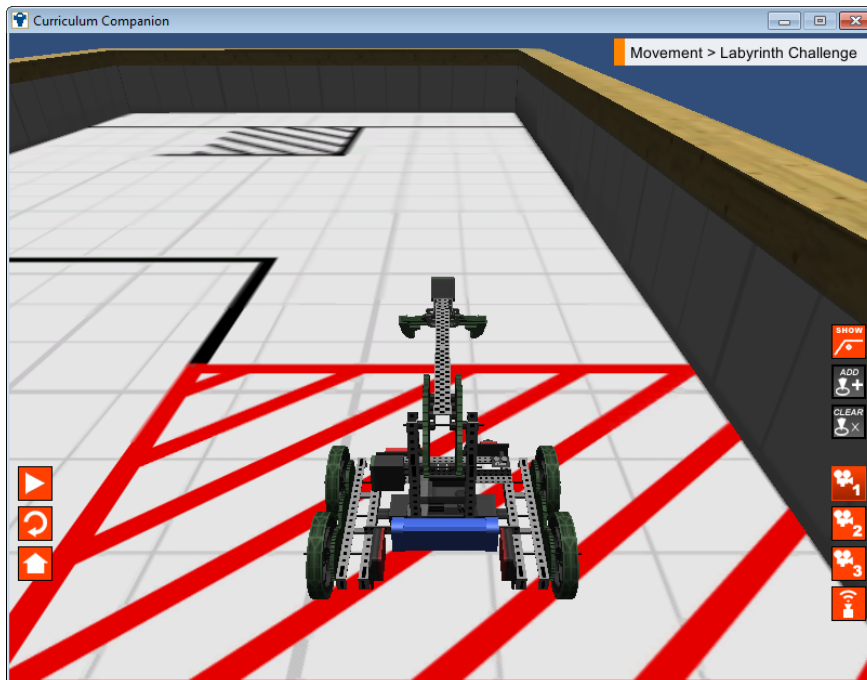




# Getting Started in RobotC

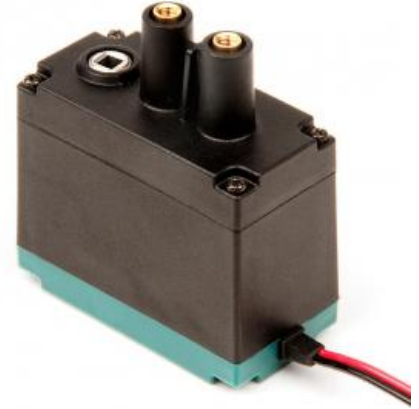
- // Comment
  - task
  - main()
  - motor[]
    - { }
- wait1Msec()
  - ;
  - =
- Header
  - Code
- Compile
- Download
  - Run



# Learning Objectives

- Understand Motion
  - Motors: How they work and respond.
  - Fuses: Understand why they keep blowing
- Understand how to control Motors with a program including
  - Setting up the motors
  - Reading the basic outline of a program
  - Using commands for controlling motors
    - `motor[port1] = ...`
    - `motor[rightMotor] = ...`
    - `wait1Msec();`
- Be able to write programs for a Robot to complete r virtual challenges.

# VEX Motion: Motors



- 2-Wire Motor 393
  - 100 RPM
    - No load
  - Torque peaks at 13.5 in-lbs at
    - 0 RPM
    - **3.6 amp draw**
  - Continually at 3.375 in-lbs
    - +/- 77 RPM
    - 0.9 amp draw

# High Speed Gears

- High Speed Gearing: **60% faster**
- Unscrew the motor and replace internal gearing.
  - 160 RPM
    - No Load
  - Torque 8.4 in-lb in bursts
    - 0 RPM
    - 3.6 AMP
  - Continually at 2.1 in-lbs
    - +/- 123 RPM
    - 0.9 amp draw



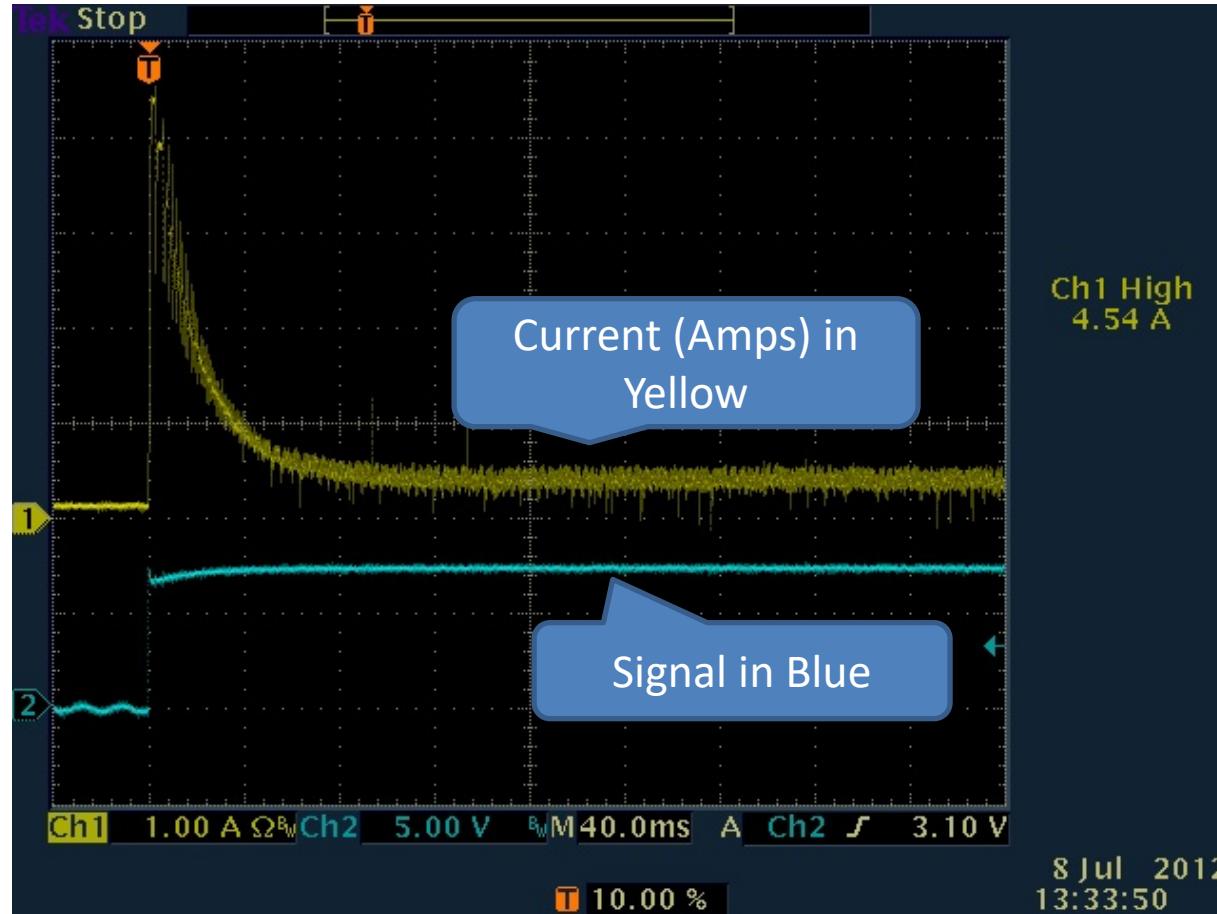
# Motor Controller

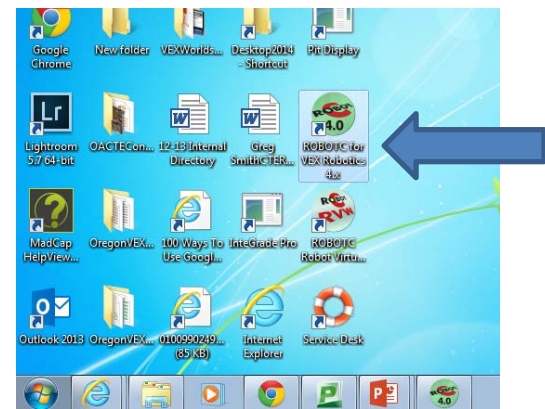
- Motor Controller: 2-Wire to 3-Wire
- Not needed for motor ports 1 and 10



# What happens when you floor it?

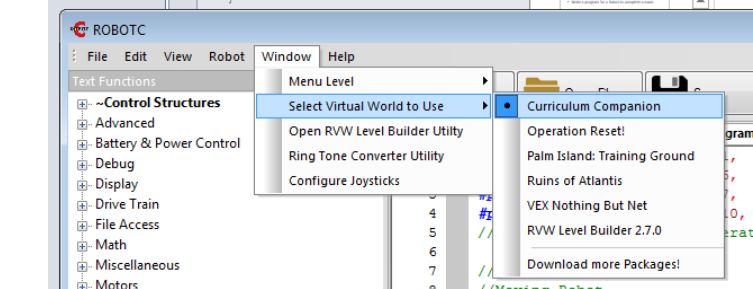
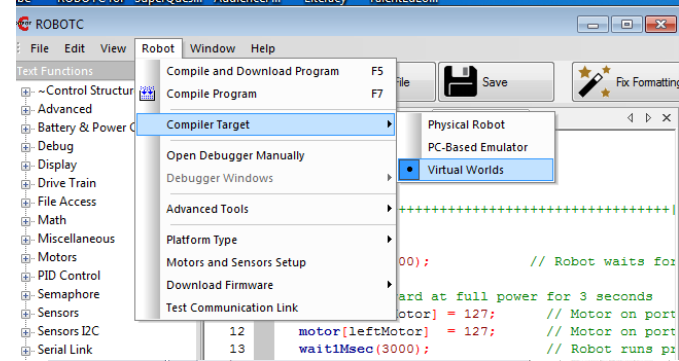
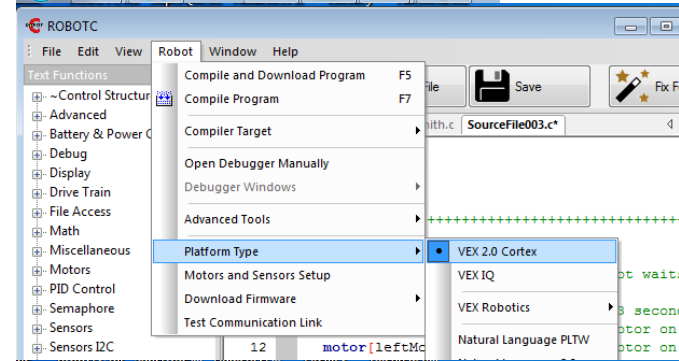
- Fuses you can blow
- Motor: 3.6 Amp
  - One Motor Stops
- Controller: 3 Amp
  - One motor stops
- Cortex Port: 4 amps combined with four other ports. Robot Stops





# Getting Started

- Open RobotC
- Select VEX 2.0 Cortex Platform
  - Robot-> Platform ->VEX 2.0 Cortex
- Make the robot compile to Virtual Worlds
  - Robot-> Compiler Target -> Virtual Worlds
- Select Virtual World
  - Window->Select Virtual World to Use -> Curriculum Companion



# Your Robot

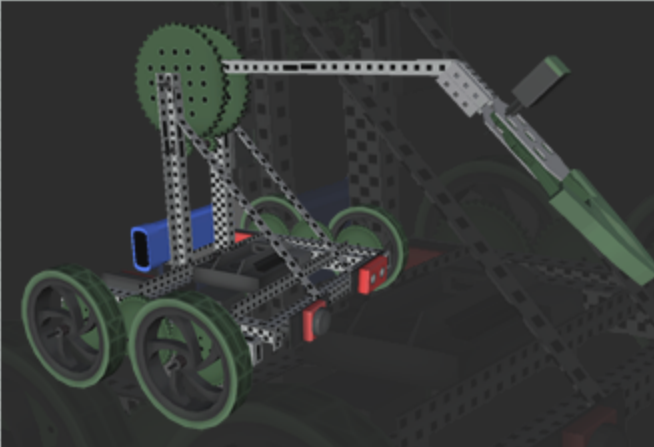
Curriculum Companion

HOME LOGIN OPTIONS BADGES

ROBOTS MOVEMENT REMOTE CONTROL SENSING UTILITY

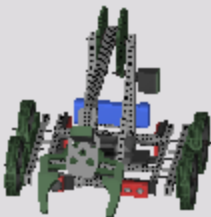
## ROBOTS

- VEX Squarebot
- VEX Clawbot**
- VEX Swervebot
- Buggy Bot
- Mammal Bot



### VEX Clawbot

Length: 47 cm  
Width: 31 cm  
Front Left Wheel: 4.7cm radius  
Front Right Wheel: 4.7cm radius  
Rear Left Wheel: 4.7cm radius  
Rear Right Wheel: 4.7cm radius

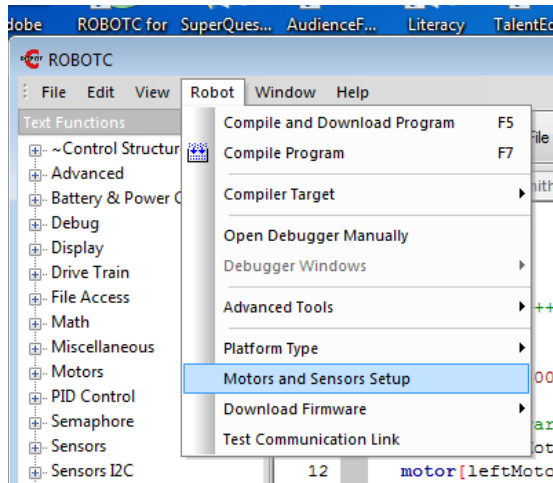


- Left Motor: Motor 1
- Claw Motor: Motor 6
- Arm Motor: Motor 7
- Right Motor: Motor 10
- Left Light Sensor: Analog 1
- Middle Light Sensor: Analog 2
- Right Light Sensor: Analog 3
- Gyro Sensor: Analog 6
- Right Shaft Encoder: Digital 1
- Left Shaft Encoder: Digital 3
- Touch Sensor: Digital 6
- Sonar Sensor: Digital 8

Curriculum Companion for VEX v4.3.1 (C) 2015 Robomatter Inc.



# Configuring the Robot: Focus on Motors



- Robot -> Motors and Sensors Setup
- Select the motor
  - Currently can only purchase 393 Motors, also modify for internal gearing (high speed, turbo speed)
- Naming Convention
  - Rules
    - Start with a letter
    - No spaces, punctuation or reserved words (blue)
  - Style
    - Describes what it represents
    - First letter is lowercase
    - otherWordsStartWithUppercaseLetters
  - For these motors
    - leftMotor
    - clawMotor
    - armMotor
    - rightMotor

Left Motor: Motor 1  
Claw Motor: Motor 6  
Arm Motor: Motor 7  
Right Motor: Motor 10

# Motors and Sensors Setup

## Page

1) Select the 'Motors' tab.

3) Use the pull down menus to select the motor.

4) The left motor will need to be reversed so the robot does not go in circles.

2) Name the motor in the desired port.

5) Select the side for drive motors.

6) Complete the setup for the remaining motors.

### Naming Conventions

#### Rules

- Start with a letter
- No spaces, punctuation or reserved words (blue)

#### Style

- Describes what it represents
- First letter is lowercase
- otherWordsStartWithUppercaseLetters

7) Click on Apply to remember the changes.

| Port   | Name       | Type          | Reversed                            | Encoder Port | PID Control              | Drive Motor Side |
|--------|------------|---------------|-------------------------------------|--------------|--------------------------|------------------|
| port1  | LeftMotor  | VEX 393 Motor | <input checked="" type="checkbox"/> | None         | <input type="checkbox"/> | Left             |
| port2  |            | No motor      |                                     |              |                          |                  |
| port3  |            | No motor      |                                     |              |                          |                  |
| port4  |            | No motor      |                                     |              |                          |                  |
| port5  |            | No motor      |                                     |              |                          |                  |
| port6  | clawMotor  | VEX 393 Motor | <input type="checkbox"/>            | None         | <input type="checkbox"/> | None             |
| port7  | armMotor   | VEX 393 Motor | <input type="checkbox"/>            | None         | <input type="checkbox"/> | None             |
| port8  |            | No motor      |                                     |              |                          |                  |
| port9  |            | No motor      |                                     |              |                          |                  |
| port10 | rightMotor | VEX 393 Motor | <input type="checkbox"/>            | None         | <input type="checkbox"/> | Right            |

OK

Cancel

Apply

Help

# Code the setup creates 'pre-processor directives'

```
VEX Start Page | Moving ForwardSmith.c | SmithFirstProgramRobotc.c
1  #pragma config(Motor, port1, leftMotor, tmotorVex393_HBridge, openLoop, reversed, driveLeft)
2  #pragma config(Motor, port6, clawMotor, tmotorVex393_MC29, openLoop)
3  #pragma config(Motor, port7, armMotor, tmotorVex393_MC29, openLoop)
4  #pragma config(Motor, port10, rightMotor, tmotorVex393_HBridge, openLoop, driveRight)
5  /*!!Code automatically generated by 'ROBOTC' configuration wizard      !!*/
6
7  //Greg Smith
8  //Moving Robot
9  //8-4-2015
10 ..
```

# Now we can start looking at RobotC

- `motor[motorName] = motorPower;`
- `wait1Msec(milliseconds);`
- `wait10Msec();`

Vocabulary  
//  
Comment  
task  
main()  
motor[]  
motorB.  
motorC  
{  
wait10Msec()  
;  
=  
Header  
Code  
Compile  
Download  
Run

{  
Marks  
the  
begin  
and  
end of  
a block  
of code

What do you think this  
code will do?

The Header  
// In front of the line  
makes this line a  
comment  
/\* \*/ for multiple  
line comments.

task main()  
Marks the beginning of the  
instructions for the Robot.  
RobotC Is CaSe SeNsItIvE!

;  
is used to mark the end of a command.

motor[motorB] = 127;  
motor[] Used to select  
the motor.  
rightMotor = This  
represents the place  
where the motor is  
attached.  
motor[port10] = 127;  
does the same thing.  
= 127;  
127 = full power  
-127 = Reverse  
0 = stop

wait1Msec(2000);  
The robot continues what it  
was doing for (2000)  
milliseconds.  
Two seconds in this case.

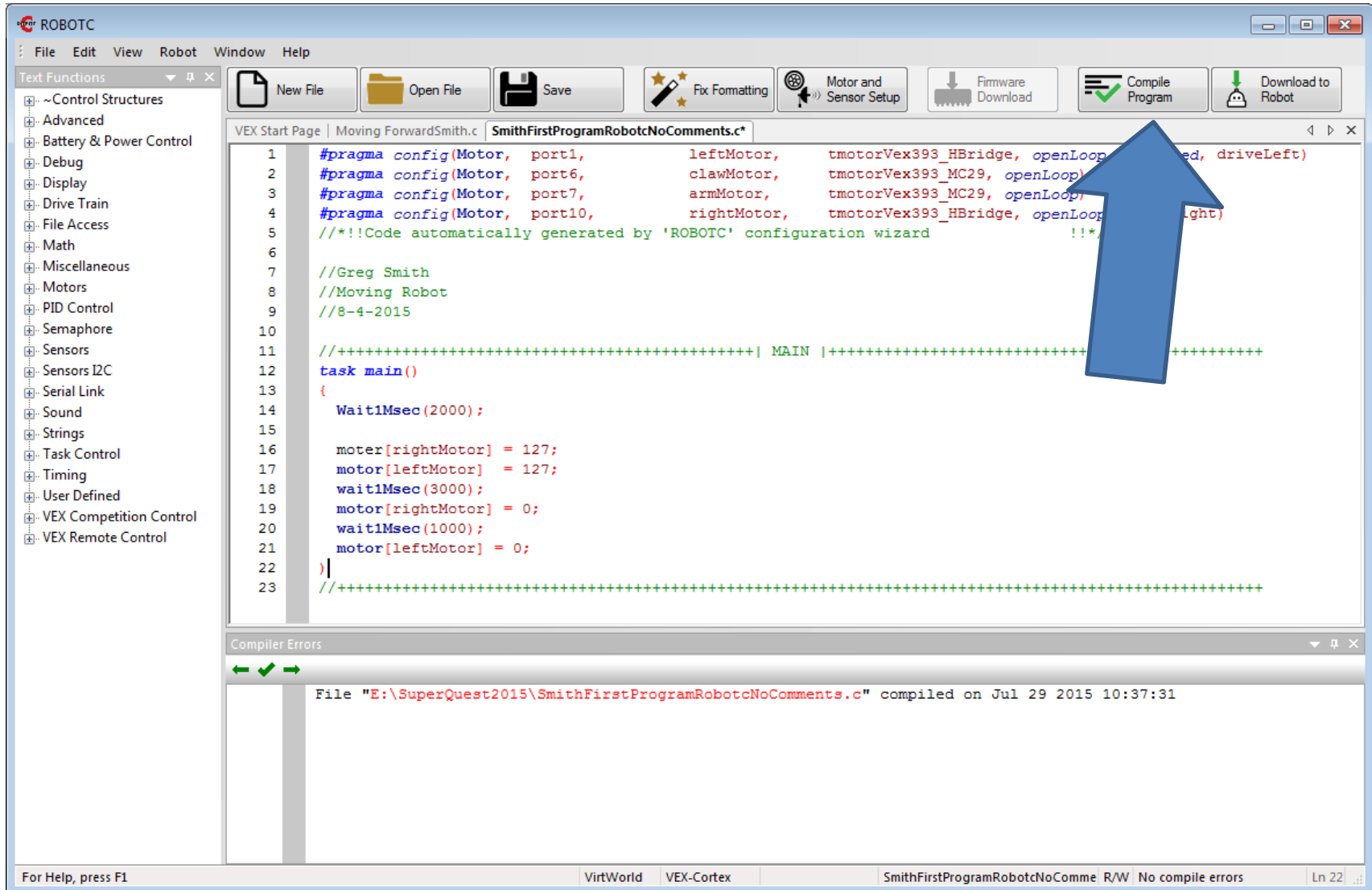
Code Break. Open  
RobotC, configure the  
motors and enter the  
above code.

```
3 #pragma config(MOTOR, port10)
4
5 {
6
7 //Greg Smith
8 //Moving Robot
9 //8-4-2015
10
11 //+++++
12
13 task main()
14 {
15
16     wait1Msec(2000);
17
18     motor[rightMotor] = 127;
19     motor[leftMotor] = 127;
20     wait1Msec(3000);
21     motor[rightMotor] = 0;
22     wait1Msec(1000);
23     motor[leftMotor] = 0;
24 }
25 //+++++
```

# Testing the Program

- Compile the program
  - Changes into machine code that the robot understands.
- Download the program
  - Moving the machine language to your Virtual or Physical Robot
- Virtual Robot
  - Log in
  - Select Robot
  - Select Challenge
  - Start Activity

# Compiling the Program



The screenshot displays the ROBOTC software interface. The main window shows a C program named `SmithFirstProgramRobotcNoComments.c`. The code includes motor configurations and a `task main()` function. A blue arrow points to the **Compile Program** button in the toolbar. Below the code editor, the **Compiler Errors** window shows a successful compilation message: `File "E:\SuperQuest2015\SmithFirstProgramRobotcNoComments.c" compiled on Jul 29 2015 10:37:31`. The status bar at the bottom indicates "No compile errors".

```
1  #pragma config(Motor, port1, leftMotor, tmotorVex393_HBridge, openLoop, driveLeft)
2  #pragma config(Motor, port6, clawMotor, tmotorVex393_MC29, openLoop)
3  #pragma config(Motor, port7, armMotor, tmotorVex393_MC29, openLoop)
4  #pragma config(Motor, port10, rightMotor, tmotorVex393_HBridge, openLoop, right)
5  /**!!Code automatically generated by 'ROBOTC' configuration wizard !!**
6
7  //Greg Smith
8  //Moving Robot
9  //8-4-2015
10
11 //+++++| MAIN |+++++
12 task main()
13 {
14     Wait1Msec(2000);
15
16     motor[rightMotor] = 127;
17     motor[leftMotor] = 127;
18     wait1Msec(3000);
19     motor[rightMotor] = 0;
20     wait1Msec(1000);
21     motor[leftMotor] = 0;
22 }
23 //+++++|
```

File "E:\SuperQuest2015\SmithFirstProgramRobotcNoComments.c" compiled on Jul 29 2015 10:37:31

For Help, press F1 | VirtWorld | VEX-Cortex | SmithFirstProgramRobotcNoComme R/W | No compile errors | Ln 22

# Oops!

The screenshot shows the ROBOTC IDE interface. The main editor window displays the following code:

```
1 #pragma config(Motor, port1, leftMotor, tmotorVex393_HBridge, openLoop, reversed, driveLeft)
2 #pragma config(Motor, port6, clawMotor, tmotorVex393_MC29, openLoop)
3 #pragma config(Motor, port7, armMotor, tmotorVex393_MC29, openLoop)
4 #pragma config(Motor, port10, rightMotor, tmotorVex393_HBri
5 /**!!Code automatically generated by 'ROBOTC' configuration wizard
6
7 //Greg Smith
8 //Moving Robot
9 //8-4-2015
10
11 //+++++-----| MAIN |-----
12 task main()
13 {
14   Wait1Msec(20)
15
16   moter[rightMotor] = 0;
17   motor[leftMotor] = 0;
18   wait1Msec(30)
19   motor[rightMotor] = 0;
20   wait1Msec(10)
21   motor[leftMotor] = 0;
22 }
23 //+++++-----
```

The code contains several syntax errors marked with red 'X's: line 13 (missing closing brace), line 14 (misspelled 'wait1Msec'), line 16 (undefined variable 'moter'), line 17 (invalid assignment to a motor object), line 18 (missing closing brace for the task main), and line 22 (missing closing brace for the task main).

The 'Compiler Errors' window at the bottom shows the following messages:

```
File "E:\SuperQuest2015\SmithFirstProgramRobotcNoComments.c" compiled on Jul 29 2015 10:38:34
13 **Error**:Unmatched left brace '{'
14 **Warning**:Substituting similar variable 'wait1Msec' for 'Wait1Msec'. Check spelling and letter case.
16 **Error**:Undefined variable 'moter'. 'short' assumed.
16 **Error**:LValue for '[' operator must be a pointer
16 **Error**:'[' operator requires pointer value on left hand side [2].
22 **Error**:Unexpected ')' during parsing
23 **Error**:Expected->}'. Found 'EOF'
```

The compiler catches syntax (typing) errors and gives some hints on how to fix them.



# Errors

```
11 //+++++
12 task main()
13 {
14     Wait1Msec(2000);
15
16     moter[rightMotor] = 127;
17     motor[leftMotor] = 127;
18     wait1Msec(3000);
19     motor[rightMotor] = 0;
20     wait1Msec(1000);
21     motor[leftMotor] = 0;
22 }
23 //+++++
```

Red X = error  
Yellow X = Warning

Any guesses on  
how to fix these  
mistakes?

Errors and hints  
on the bottom of  
the page. If you  
click on an error  
it will highlight  
the line of the  
error.

## Compiler Errors

```
13 **Error**:Unmatched left brace '{'
14 *Warning*:Substituting similar variable 'wait1Msec' for 'Wait1Msec'. Check spelling and letter
16 **Error**:Undefined variable 'moter'. 'short' assumed.
16 **Error**:LValue for '[' operator must be a pointer
16 **Error**:'[' operator requires pointer value on left hand side [2].
22 **Error**:Unexpected ')' during parsing
23 **Error**:Expected->}'. Found 'EOF'
```

# Corrected and Compiled

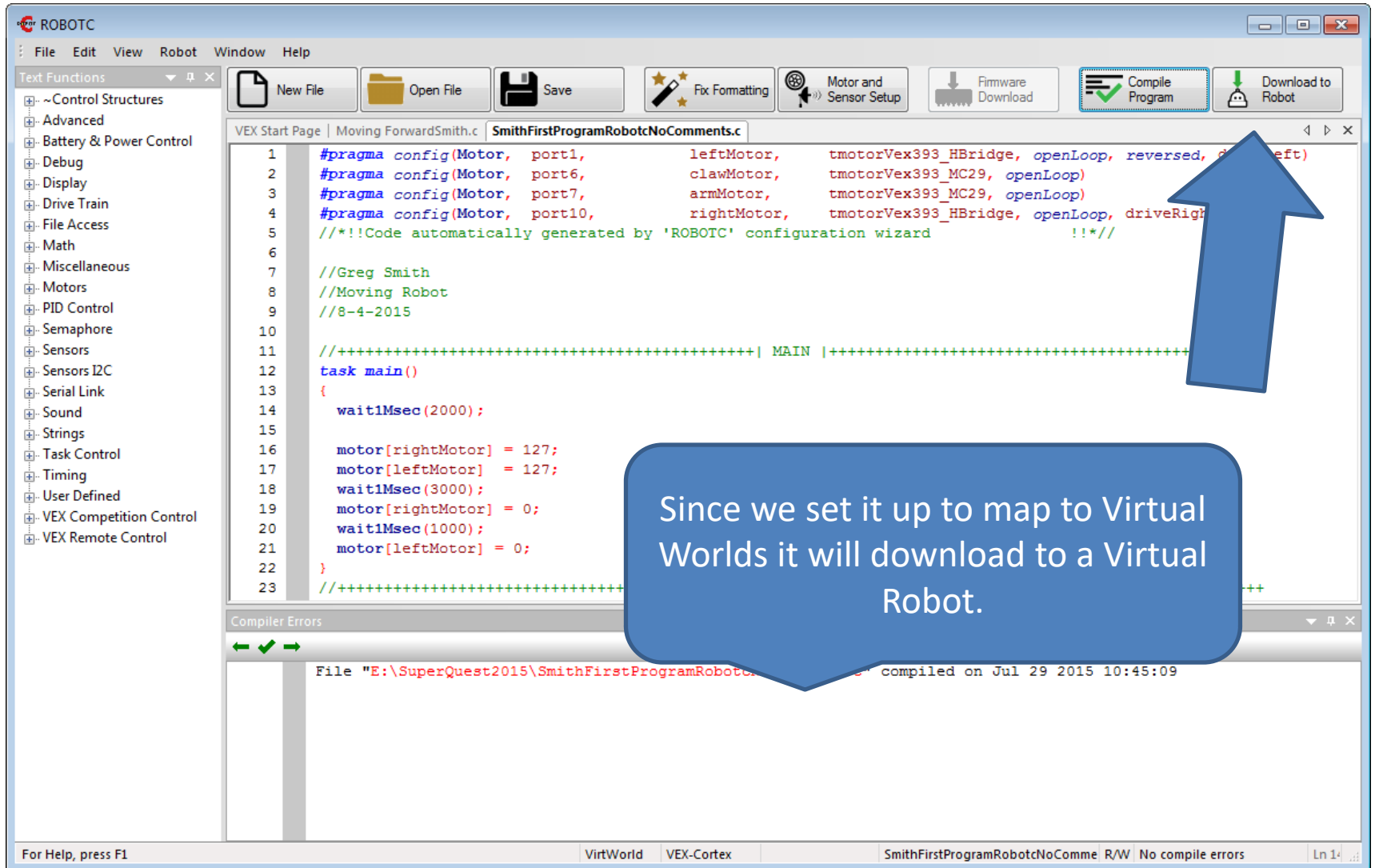
The screenshot displays the ROBOTC IDE interface. The main window shows a C program named `SmithFirstProgramRobotcNoComments.c` with the following code:

```
1  #pragma config(Motor,  port1,          leftMotor,    tmotorVex393_HBridge, openLoop, reversed, driveLeft)
2  #pragma config(Motor,  port6,          clawMotor,    tmotorVex393_MC29, openLoop)
3  #pragma config(Motor,  port7,          armMotor,     tmotorVex393_MC29, openLoop)
4  #pragma config(Motor,  port10,         rightMotor,   tmotorVex393_HBridge, openLoop, driveRight)
5  /*!!Code automatically generated by 'ROBOTC' configuration wizard      !!*/
6
7  //Greg Smith
8  //Moving Robot
9  //8-4-2015
10
11 //+++++| MAIN |+++++
12 task main()
13 {
14     wait1Msec(2000);
15
16     motor[rightMotor] = 127;
17     motor[leftMotor]  = 127;
18     wait1Msec(3000);
19     motor[rightMotor] = 0;
20     wait1Msec(1000);
21     motor[leftMotor]  = 0;
22 }
23 //+++++| MAIN |+++++
```

The bottom panel, titled "Compiler Errors", shows a green checkmark and the message: "File "E:\SuperQuest2015\SmithFirstProgramRobotcNoComments.c" compiled on Jul 29 2015 10:45:09".

The status bar at the bottom indicates: "For Help, press F1", "VirtWorld VEX-Cortex", "SmithFirstProgramRobotcNoComme R/W", "No compile errors", and "Ln 1".

# Download the program



The screenshot shows the ROBOTC software interface. The main window displays a C program named `SmithFirstProgramRobotcNoComments.c`. The code includes motor configurations and a `task main()` function. A blue arrow points to the `Download to Robot` button in the toolbar. A blue callout box contains the text: "Since we set it up to map to Virtual Worlds it will download to a Virtual Robot." The bottom status bar shows the file was compiled on Jul 29 2015 10:45:09.

```
1 #pragma config(Motor, port1, leftMotor, tmotorVex393_HBridge, openLoop, reversed, portLeft)
2 #pragma config(Motor, port6, clawMotor, tmotorVex393_MC29, openLoop)
3 #pragma config(Motor, port7, armMotor, tmotorVex393_MC29, openLoop)
4 #pragma config(Motor, port10, rightMotor, tmotorVex393_HBridge, openLoop, driveRight)
5 /*!!Code automatically generated by 'ROBOTC' configuration wizard !!*/
6
7 //Greg Smith
8 //Moving Robot
9 //8-4-2015
10
11 //+++++| MAIN |+++++
12 task main()
13 {
14     wait1Msec(2000);
15
16     motor[rightMotor] = 127;
17     motor[leftMotor] = 127;
18     wait1Msec(3000);
19     motor[rightMotor] = 0;
20     wait1Msec(1000);
21     motor[leftMotor] = 0;
22 }
23 //+++++|
```

File "E:\SuperQuest2015\SmithFirstProgramRobotcNoComments.c" compiled on Jul 29 2015 10:45:09

Set up and Account with CS2N. It will track progress.  
Can log in locally as a guest without tracking.



# Information for CS2N



## Welcome to CS-STEM Network

To get started, fill out the form below.

**Birthday** ⓘ

Month

Day

Year

**New Username** ⓘ

Enter Username

**Email**

Enter Email

**Confirm Email**

Confirm Email

**Password**

Enter Password

**Confirm Password**

Confirm Password

**First Name**

Enter First Name

**Last Name**

Enter Last Name

**Gender**

# Select Your Robot

Curriculum Companion

HOME LOGOUT OPTIONS BADGES

ROBOTS MOVEMENT REMOTE CONTROL SENSING UTILITY

**ROBOTS**

- VEX Squarebot
- VEX Clawbot**
- VEX Sweeiebot
- Bug Bot
- Marshall Bot

**VEX Clawbot**

Length: 47 cm  
Width: 31 cm  
Front Left Wheel: 4.7cm radius  
Front Right Wheel: 4.7cm radius  
Rear Left Wheel: 4.7cm radius  
Rear Right Wheel: 4.7cm radius

Left Motor: Motor 1  
Claw Motor: Motor 6  
Arm Motor: Motor 7  
Right Motor: Motor 10  
Left Light Sensor: Analog 1  
Middle Light Sensor: Analog 2  
Right Light Sensor: Analog 3  
Gyro Sensor: Analog 6  
Right Shaft Encoder: Digital 1  
Left Shaft Encoder: Digital 3  
Touch Sensor: Digital 6  
Sonar Sensor: Digital 8

Curri

(C) 2015 Robomatter Inc.

1) Robots Tab

2) We will be using the Clawbot for discussion

Note: This shows the motors and sensors attached to this robot

# Robots Tab

Curriculum Companion

HOME LOGOUT OPTIONS BADGES

ROBOTS MOVEMENT REMOTE CONTROL SENSING UTILITY

## ROBOTS

- VEX Squarebot
- VEX Clawbot
- VEX Swervebot
- Buggy Bot
- Mammal Bot

### VEX Clawbot

Length: 47 cm  
Width: 31 cm  
Front Left Wheel: 4.7cm radius  
Front Right Wheel: 4.7cm radius  
Rear Left Wheel: 4.7cm radius  
Rear Right Wheel: 4.7cm radius

Left Motor: Motor 1  
Claw Motor: Motor 6  
Arm Motor: Motor 7  
Right Motor: Motor 10  
Left Light Sensor: Analog 1  
Middle Light Sensor: Analog 2  
Right Light Sensor: Analog 3  
Gyro Sensor: Analog 6  
Right Shaft Encoder: Digital 1  
Left Shaft Encoder: Digital 3  
Touch Sensor: Digital 6  
Sonar Sensor: Digital 8

Curriculum Companion for VEX v4.5.0 (C) 2015 Robomatter Inc.

Different Robots Available

Physical properties of Robot

Motors and Sensors on the Robot selected.



# Challenges

The screenshot shows the 'Curriculum Companion' application window. At the top, there are navigation icons for HOME, LOGOUT, OPTIONS, and BADGES. Below these are five red tabs: ROBOTS, MOVEMENT (selected), REMOTE CONTROL, SENSING, and UTILITY. The main content area is divided into three sections:

- MOVEMENT** (left sidebar): A list of challenges with star icons indicating favorites. The 'Basketball Drills' challenge is highlighted.
- Basketball Drills** (center): A 3D perspective view of a robot on a grid floor. A yellow box highlights a starting point labeled 'A' on the left side of the grid.
- Basketball Drills** (bottom left): A text box containing a summary of the challenge: 'Use the encoders to perform a basketball-like drill. Beginning from the starting line, the robot must traverse to the first line, stop, return to the starting line, and repeat for each line. Completion of this challenge is required for the Movement Mastery badge.' Below the text is a PDF icon and the text 'Specification Document'.
- Achievements** (bottom right): A section titled 'Achievements' with a star icon and the text 'Basketball Drills Completion'. Below this, it says 'Current Robot: VEX Clawbot' and 'Fixed starting point: Point A'. At the bottom of this section is a green button labeled 'START ACTIVITY' with a right-pointing arrow.

At the bottom right of the window, the text '(C) 2015 Robomatter Inc.' is visible.

Click to Select Different Challenges

Summary of the activity with a pdf that includes details about the challenge.

Click on 'Start Activity' to begin

Achievements Some activities allow you



# We'll Test Our Program in the Utility -> Imperial Distance Utility

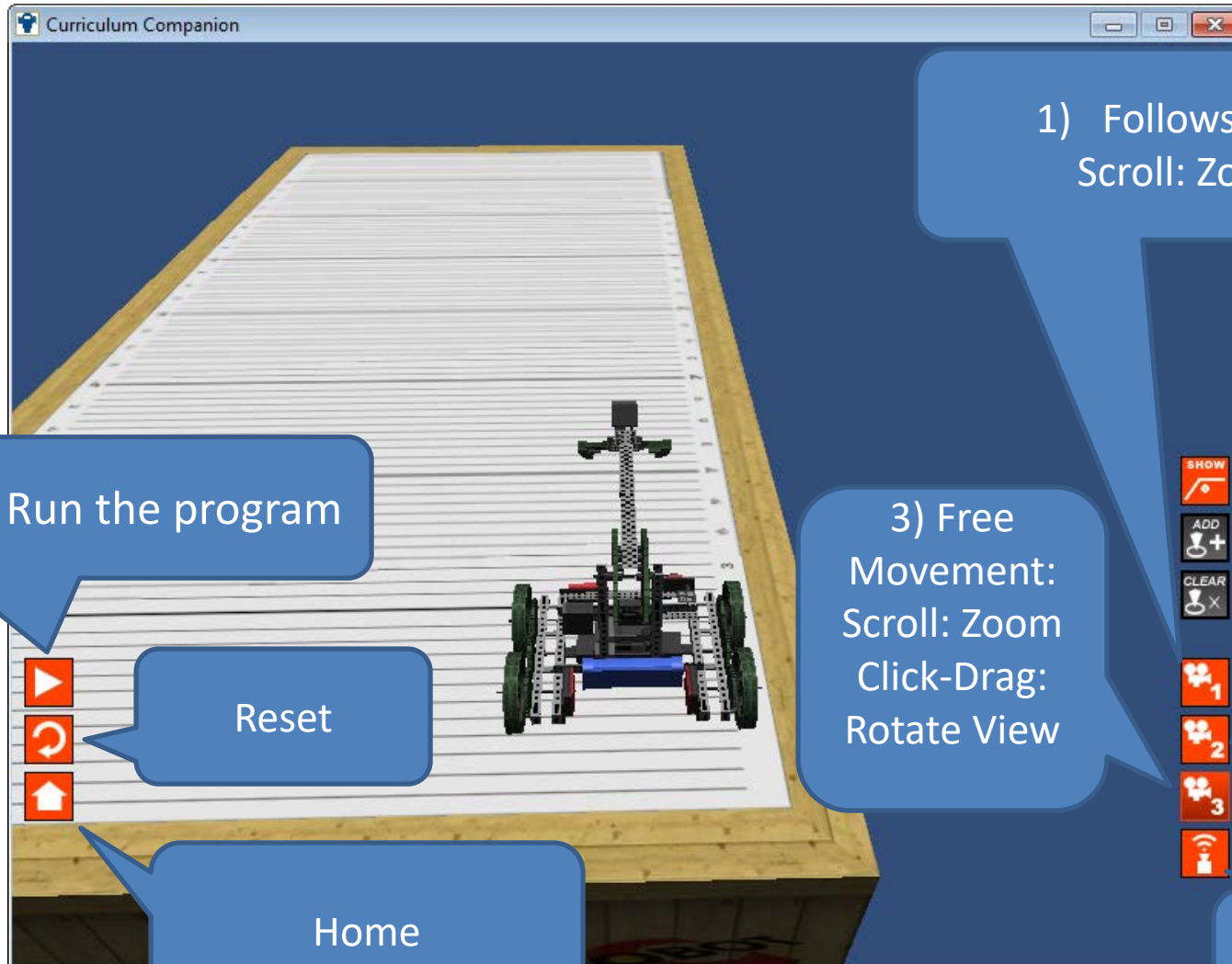
The screenshot shows the Curriculum Companion software interface. At the top, there are navigation buttons: HOME, LOGOUT, OPTIONS, and BADGES. Below these are four main tabs: ROBOTS, MOVEMENT, REMOTE CONTROL, and SENSING. The UTILITY tab is currently selected and highlighted in red. On the left side, a list of activities is shown under the UTILITY header, with 'Imperial Distance Utility' selected. The main content area displays a 3D perspective view of a table with a ruler, with callouts 'A' and 'B' indicating specific points. Below this view, the 'Imperial Distance Utility' activity page is shown, including a description, a 'Specification Document' link, and a 'START ACTIVITY' button. The footer contains the text 'Curriculum Companion for VEX v4.3.1 (C) 2015 Robomatter Inc.'

2) Imperial Distance Utility

1) Utility Tab

3) Start Activity

# Select Camera and Go



Play: Run the program

Reset

Home

1) Follows Right  
Scroll: Zoom

2) Top  
Down  
View

3) Free  
Movement:  
Scroll: Zoom  
Click-Drag:  
Rotate View

Show Sensors  
Toggle

# Your Turn

- Enter the Sample program
  - Motors and Sensors Setup
  - task main() and code
- Compile and correct errors
- Download to the virtual robot
- Run the program
- Can you modify this program to...
- Write the letter Z? S?

```
6
7 //Greg Smith
8 //Moving Robot
9 //8-4-2015
10
11 //+++++
12 task main()
13 {
14     wait1Msec(2000);
15
16     motor[rightMotor] = 127;
17     motor[leftMotor]  = 127;
18     wait1Msec(3000);
19     motor[rightMotor] = 0;
20     wait1Msec(1000);
21     motor[leftMotor]  = 0;
22 }
23 //+++++
```

Reference Website:

[http://education.rec.ri.cmu.edu/products/cortex\\_video\\_trainer/](http://education.rec.ri.cmu.edu/products/cortex_video_trainer/)

Click on Movement for much of the material covered

# Teaching Strategy: Grading Student Programming

Movement: Basketball Drills

Programming (10 Points = 100%)

- Program compiles (4 points)
- Header complete with names, description and date (2 points)
- Code is properly indented (2 points)
- Comments in the program describing the code (2 points)

Performance (10 points = 100%)

- Completed

# Online Time: Movement Challenges

- Basketball Drills
- Sentry Simulation 1
- Sumo Bot
- Labyrinth Challenge

The screenshot displays the Curriculum Companion software interface. At the top, there is a navigation bar with icons for HOME, LOGOUT, OPTIONS, and BADGES, followed by tabs for ROBOTS, MOVEMENT, REMOTE CONTROL, SENSING, and UTILITY. The MOVEMENT tab is active, showing a list of activities on the left: Basketball Drills, Driving Straight I, Driving Straight II, Labyrinth Challenge (highlighted with a star), Power Level Inv. (Encoders), Robot Acceleration, Sentry Simulation I, Simulated Acceleration, Sumo Bot, Turning Inv. (Encoders), and Wait State Inv. The main area shows a 3D perspective view of a robot on a grid floor with a 2D inset of a labyrinth maze. A red square labeled 'A' indicates the starting point. Below the 3D view, the 'Labyrinth Challenge' section provides instructions: 'Using a series of turning and movement behaviors, program the robot to reach the end of the labyrinth from the starting position. Completion of this challenge is required for the Movement Mastery badge.' A 'Specification Document' link is provided. The 'Achievements' section lists: Moving Forward, Turn Left, Turn Right, and Labyrinth Completion. The current robot is identified as 'VEX Clawbot' and the fixed starting point is 'Point A'. A green 'START ACTIVITY' button is at the bottom right. The footer contains 'Curriculum Companion for VEX v4.3.1 (C) 2015 Robomatter Inc.'

# Basketball Drill Programming Alternatives

- Using the Basketball Drills Activity to introduce:
  - Variables
  - For loop
  - Functions

# Looking at Potential Solutions to Basketball Drills

- Pseudo Code
- Go forward long enough to cross the first line
- Come back
- Go forward long enough to cross the second line
- Come back
- Go forward long enough to cross the third line
- Come back

With enough guessing and checking, you can get the correct values for the wait1Msec()

```
26 //+++++
27 task main()
28 {
29     wait1Msec(1000);
30     // Move forward to First Line
31     motor[rightMotor] = 63;
32     motor[leftMotor] = 63;
33     wait1Msec(2400);
34     //Back up
35     motor[rightMotor] = -63;
36     motor[leftMotor] = -63;
37     wait1Msec(2400);
38     //Move Forward to Second Line
39     motor[rightMotor] = 63;
40     motor[leftMotor] = 63;
41     wait1Msec(4800);
42     //Back up
43     motor[rightMotor] = -63;
44     motor[leftMotor] = -63;
45     wait1Msec(4800);
46     //Move Forward to Third Line
47     motor[rightMotor] = 63;
48     motor[leftMotor] = 63;
49     wait1Msec(7200);
50     //Back up
51     motor[rightMotor] = -63;
52     motor[leftMotor] = -63;
53     wait1Msec(7200);
54 }
```

# Using a Variable to help with changes

If only there was a tool in RobotC that would let the code repeat.

```
24
25 int timeToLine = 2400;
26 //+++++
27 task main()
28 {
29     wait1Msec(1000);
30     // Move forward to First Line
31     motor[rightMotor] = 63;
32     motor[leftMotor] = 63;
33     wait1Msec(timeToLine);
34     //Back up
35     motor[rightMotor] = -63;
36     motor[leftMotor] = -63;
37     wait1Msec(timeToLine);
38     //Move Forward to Second Line
39     motor[rightMotor] = 63;
40     motor[leftMotor] = 63;
41     wait1Msec(2*timeToLine);
42     //Back up
43     motor[rightMotor] = -63;
44     motor[leftMotor] = -63;
45     wait1Msec(2*timeToLine);
46     //Move Forward to Third Line
47     motor[rightMotor] = 63;
48     motor[leftMotor] = 63;
49     wait1Msec(3*timeToLine);
50     //Back up
51     motor[rightMotor] = -63;
52     motor[leftMotor] = -63;
53     wait1Msec(3*timeToLine);
54 }
55 //+++++
```



# For loop in RobotC

- When to use it
  - When you want to repeat something a set number of times

- Syntax

Declares an integer variable called line and gives it an initial value of 1

If the line variable is less than or equal to 3 when it reaches this, it will do the loop another time.

After completing the loop, it will add 1 to the variable line.

```
for(int line = 1; line<=3; line++)  
{  
    //Code repeated  
}
```

In this example it will repeat the code inside the {} three times.  
Once when line = 1  
Once when line = 2  
And  
Once when line = 3

# No loop vs. for loop

```
24
25 int timeToLine = 2400;
26 //+++++
27 task main()
28 {
29     wait1Msec(1000);
30     // Move forward to First Line
31     motor[rightMotor] = 63;
32     motor[leftMotor] = 63;
33     wait1Msec(timeToLine);
34     //Back up
35     motor[rightMotor] = -63;
36     motor[leftMotor] = -63;
37     wait1Msec(timeToLine);
38     //Move Forward to Second Line
39     motor[rightMotor] = 63;
40     motor[leftMotor] = 63;
41     wait1Msec(2*timeToLine);
42     //Back up
43     motor[rightMotor] = -63;
44     motor[leftMotor] = -63;
45     wait1Msec(2*timeToLine);
46     //Move Forward to Third Line
47     motor[rightMotor] = 63;
48     motor[leftMotor] = 63;
49     wait1Msec(3*timeToLine);
50     //Back up
51     motor[rightMotor] = -63;
52     motor[leftMotor] = -63;
53     wait1Msec(3*timeToLine);
54 }
55 //+++++
```

```
int timeToLine = 2400;
//+++++
task main()
{
    for(int line = 1; line<=3; line++)
    {
        //Go Forward
        motor[rightMotor] = 63;
        motor[leftMotor] = 63;
        wait1Msec(line*timeToLine);
        //Back up
        motor[rightMotor] = -63;
        motor[leftMotor] = -63;
        wait1Msec(line*timeToLine);
    }
}
```

// P1

# For loop example

Since line = 1 the first time through this loop

**line\*timeToLine** is the same as  
**1\*2400 = 2400**

the first time through this loop.

Then

**2\*2400 = 4800**

the second time and

**3\*2400 = 7200**

the third time.

```
// P
```

RobotC does the math inside the () before executing the wait1Msec() command

```
int timeToLine = 2400;
//+++++
task main()
{
  for(int line = 1; line<=3; line++)
  {
    //Go Forward
    motor[rightMotor] = 63;
    motor[leftMotor] = 63;
    wait1Msec(line*timeToLine);
    //Back up
    motor[rightMotor] = -63;
    motor[leftMotor] = -63;
    wait1Msec(line*timeToLine);
  }
}
```

# Using Functions to make the **main body** easier to read

```
int timeToLine = 2400;
//+++++ moveForward +++++
void moveForward(int timeToMove)
{
    motor[rightMotor] = 63;
    motor[leftMotor] = 63;
    wait1Msec(timeToMove);
}
//+++++ moveBackward +++++
void moveBackward(int timeToMove)
{
    motor[rightMotor] = -63;
    motor[leftMotor] = -63;
    wait1Msec(timeToMove);
}
//+++++| MAIN |+++++
task main()
{
    for(int line = 1; line<=3; line++)
    {
        moveForward(line*timeToLine);
        moveBackward(line*timeToLine);
    }
}
```

Define the Functions  
above the main body.

Main Body

# Function Details

The function 'Header'  
**void** – It will not return a value  
**moveForward** – The name of this function. You get to pick the name of you function as long as:  
-Starts with a letter  
-No spaces or punctuation  
-Not a reserved Word  
And it should describe what it is doing.

```
//+++++  
moveForward
```

Comments added to make the program easier to read. You can add details, ...

```
int timeToMove
```

**int** – Sets an integer variable  
**timeToMove** – An integer variable that will store the value sent to the function in the call statement.

```
//+++++ moveForward ++++++  
void moveForward(int timeToMove)  
{  
  motor[rightMotor] = 63;  
  motor[leftMotor] = 63;  
  wait1Msec(timeToMove);  
}
```

The code for the function goes between {}. When the function is finished the program will return to the line after the call statement.

# Dry Run: Reading the Program

```
int timeToLine = 2400;
//+++++ moveForward +++++
void moveForward(int timeToMove)
{
    motor[rightMotor] = 63;
    motor[leftMotor] = 63;
    wait1Msec(timeToMove);
}
//+++++ moveBackward +++++
void moveBackward(int timeToMove)
{
    motor[rightMotor] = -63;
    motor[leftMotor] = -63;
    wait1Msec(timeToMove);
}

//+++++| MAIN |+++++
task main()
{
    for(int line = 1; line<=3; line++)
    {
        moveForward(line*timeToLine);
        moveBackward(line*timeToLine);
    }
}
```

timeToMove

timeToMove

line



Main  
Body

# Online Time: Movement Challenges

- Basketball Drills
- Sentry Simulation 1
- Sumo Bot
- Labyrinth Challenge

When you complete the activities, incorporate variables, loops, and functions

The screenshot displays the Curriculum Companion software interface. At the top, there are navigation tabs: HOME, LOGOUT, OPTIONS, BADGES, ROBOTS, MOVEMENT, REMOTE CONTROL, SENSING, and UTILITY. The 'MOVEMENT' tab is selected. Below the navigation, there is a list of activities under the 'MOVEMENT' section, including Basketball Drills, Driving Straight I, Driving Straight II, Labyrinth Challenge (highlighted with a star), Power Level Inv. (Encoders), Robot Acceleration, Sentry Simulation I, Simulated Acceleration, Sumo Bot, Turning Inv. (Encoders), Turning Inv., and Wait State Inv. The main area shows a 3D perspective view of a robot on a grid floor with a labyrinth pattern. An inset window shows a top-down view of the labyrinth with a starting point 'A' marked. Below the 3D view, there is a text box for the 'Labyrinth Challenge' with instructions: 'Using a series of turning and movement behaviors, program the robot to reach the end of the labyrinth from the starting position. Completion of this challenge is required for the Movement Mastery badge.' There is a link for 'Specification Document'. To the right, under 'Achievements', there is a list: Moving Forward, Turn Left, Turn Right, and Labyrinth Completion. Below the achievements, it says 'Current Robot: VEX Clawbot' and 'Fixed starting point: Point A'. A green 'START ACTIVITY' button is at the bottom right. The footer of the software reads 'Curriculum Companion for VEX v4.3.1 (C) 2015 Robomatter Inc.'

# References

- [http://education.rec.ri.cmu.edu/products/cortex\\_video\\_trainer/](http://education.rec.ri.cmu.edu/products/cortex_video_trainer/)