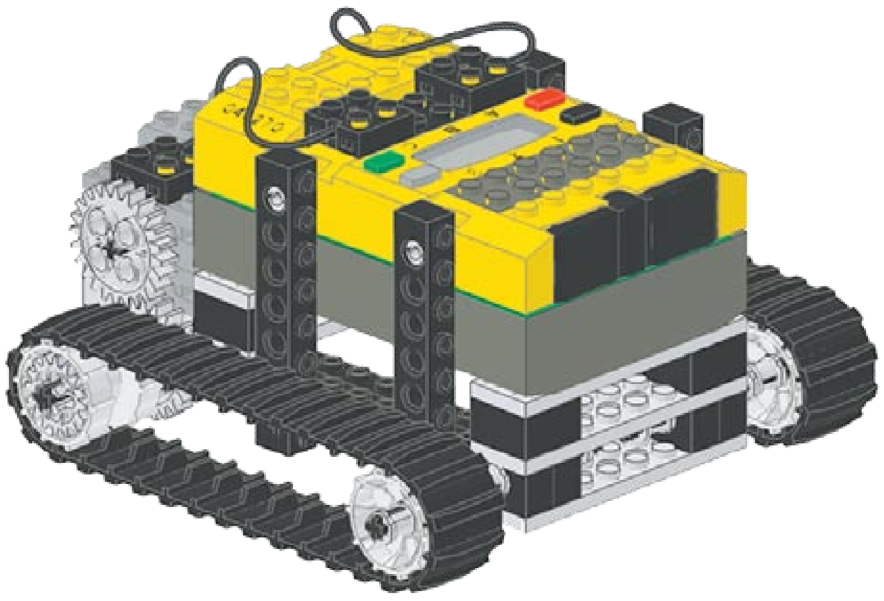
Introduction

Welcome to programming in ROBOTC.

In this lesson you will become familiar with the basic movement commands of that are available in ROBOTC. We'll be using these commands to perform behaviors such as a forward movement, turning, reverse, and other standard motor movements.

Make sure that you have the Tankbot built, and have downloaded both firmware and the sample program as shown in the videos.



Moving Forward

In this lesson you will learn about the commands used in the sample program, how to use motor commands in order to make the robot move forward, and how to control the amount of time a basic movement behavior runs.

In the last video section, you downloaded a sample program that made the robot turn. Why was the robot turning? Let's look at the code more closely, and, using the code, create a Forward behavior with the robot.

Moving Forward Program Dissection

In this section you will learn what each command does in the sample program. You will also learn how to modify the code to have your robot move forward for 3 seconds.

Below is a line by line explanation of the sample program:

Line 1: task main()

```
1  task main()
2    {
3
4      motor[motorC] = 100;
5      wait1Msec(3000);
6
7    }
```

Every ROBOTC program contains a **task main()**. The line "task main()" marks the beginning of your program's main body.

Lines 2 & 7: Braces

```
1  task main()
2    {
3
4      motor[motorC] = 100;
5      wait1Msec(3000);
6
7    }
```

Task main forms a structure which starts with an opening curly brace, and ends with a closing curly brace.

Line 3: motor[motorC]

```
1  task main()
2    {
3
4      motor[motorC] = 100;
5      wait1Msec(3000);
6
7    }
```

This is a statement, a command given to ROBOTC. This one directs the robot to turn on motorC with a power level of 100.

Example:

```
motor [motor location] = motor power level;
```

Moving Forward Program Dissection (cont.)

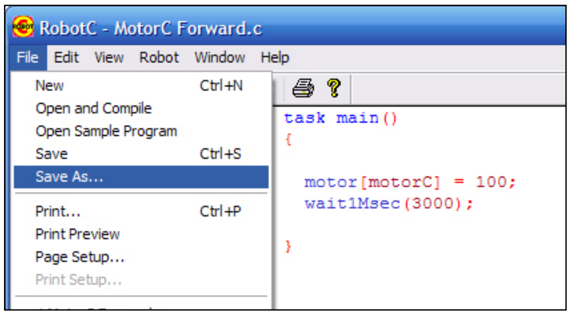
Line 4: **wait1Msec**

```
1  task main()
2    {
3
4       motor[motorC] = 100;
5       wait1Msec(3000);
6
7    }
```

Line 5 is a "Wait For" statement. Line 5 contains the ROBOTC reserved word "wait1Msec." An "msec" is a millisecond or one thousandth of a second. 3000 "msec" is 3000 thousandths or three seconds. Consequently, ROBOTC lets the motor run for three seconds.

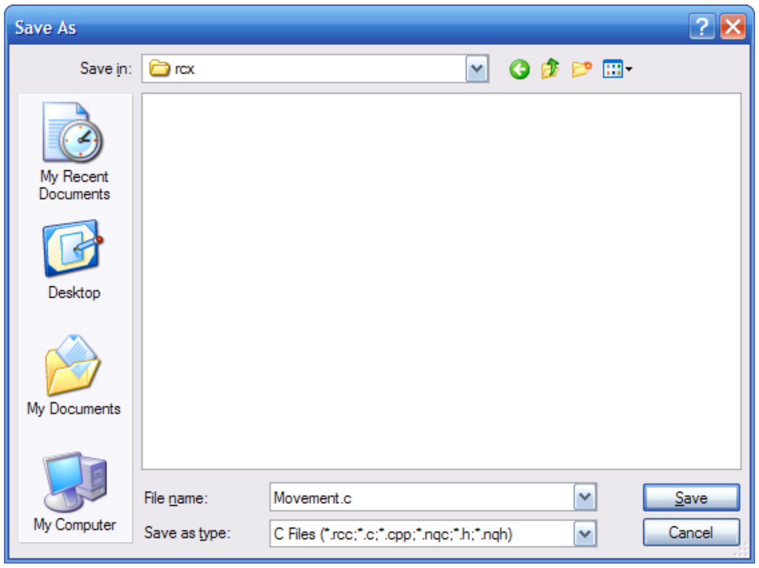
Let's modify the code to have the robot move forward for 3 seconds.

Save your program with a new name. Go to "File", "Save As".



```
task main()
{

  motor[motorC] = 100;
  wait1Msec(3000);

}
```

Save it as "Movement".



Save in: rcx

File name: Movement.c

Save as type: C Files (*.rcc;*.c;*.cpp;*.nqc;*.h;*.nqh)

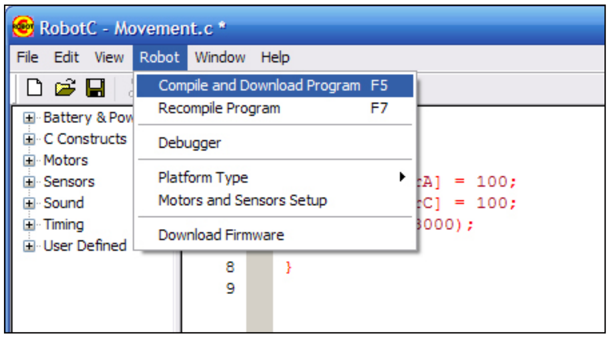
Moving Forward Program Dissection

Add this motor command to have motor A go forward with 100 power. Type here just as you would type in a word processor. ROBOTC will automatically color the words you type as it recognizes them.

```
task main()
  {

     motor[motorA] = 100;
     motor[motorC] = 100;
     wait1Msec(3000);

  }
```

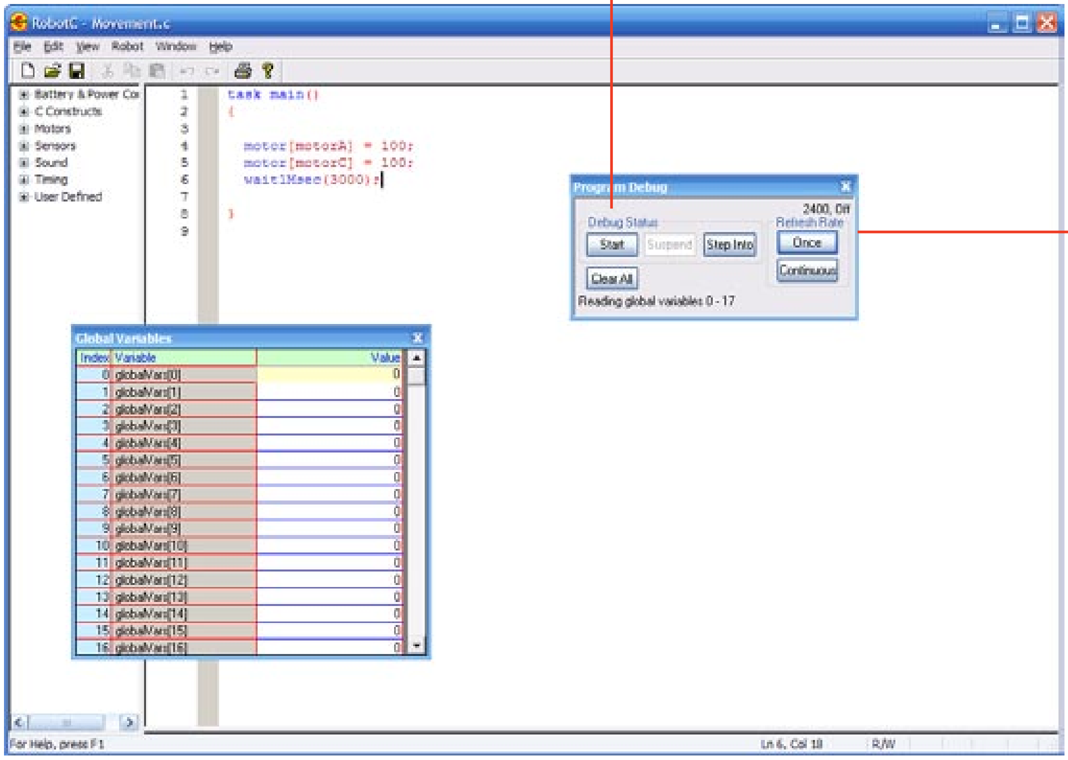
Now, download the program to the robot.

Go to "Robot", "Compile and Download Program".



Moving Forward Program Dissection

When your program is done downloading some program debug windows will appear. Press the “Start” button to run your robot.

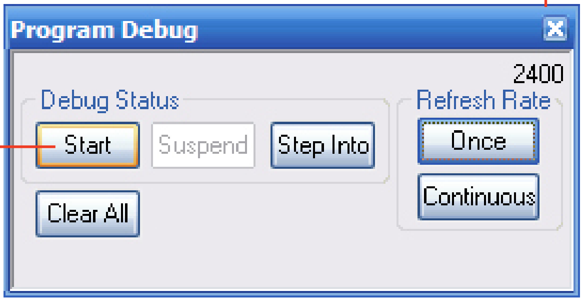


```
task main()
{

  motor[motorA] = 100;
  motor[motorC] = 100;
  wait1Msec(3000);

}
```

2400, Off

Reading global variables 0 - 17

Index	Variable	Value
0	globalVars[0]	0
1	globalVars[1]	0
2	globalVars[2]	0
3	globalVars[3]	0
4	globalVars[4]	0
5	globalVars[5]	0
6	globalVars[6]	0
7	globalVars[7]	0
8	globalVars[8]	0
9	globalVars[9]	0
10	globalVars[10]	0
11	globalVars[11]	0
12	globalVars[12]	0
13	globalVars[13]	0
14	globalVars[14]	0
15	globalVars[15]	0
16	globalVars[16]	0



Pressing the “Start” button will run the program on your robot. Your robot will begin to move immediately!

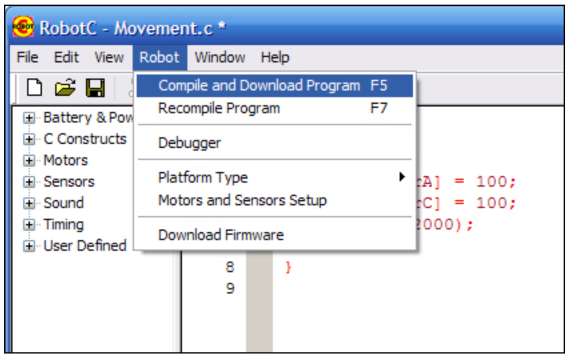
Moving Forward Timing

In this lesson you will modify the previous program to have the robot move forward for 2 seconds instead of 3.

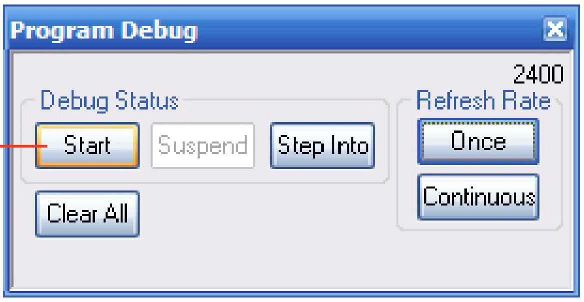
Modify line 6 so that the number inside the parentheses following "wait1Msec" is 2000 instead of 3000.

```
1  task main()
2    {
3  
4      motor[motorA] = 100;
5      motor[motorC] = 100;
6      wait1Msec(2000);
7  
8    }
```

Go to "Robot", "Compile and Download".



When the program is done downloading, click the "Start" button to run the program on your robot. Observe the robot's behavior and continue.



Speed and Direction

In this lesson you will continue to modify the "Movement" program by manipulating the motor powers, and experimenting with motor direction.

Continue on by learning how to manipulate your motor power, and consequently, motor direction. Power level allows your robot to go faster, slower, and stop by setting the percent of power that the motor is using.

Speed and Direction Braking

In this lesson, you will learn how to use a 0 power level to stop your robot.

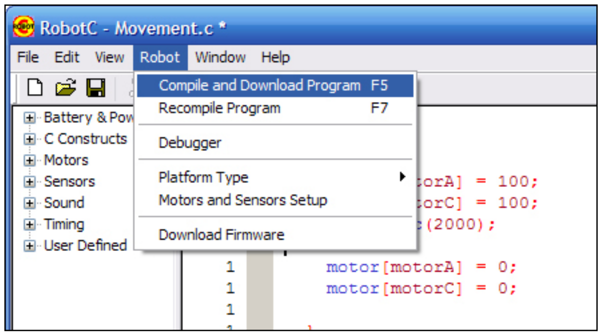
Your robot brakes automatically at the end of every program, but should you ever need to stop it in the middle of a program, simply use 0 power:

Add motor commands for both motor A and motor C. Instead of 100 power, use a power level of 0.

```
task main()
 {

    motor[motorA] = 100;
    motor[motorC] = 100;
    wait1Msec(2000);

    motor[motorA] = 0;
    motor[motorC] = 0;

 }
```

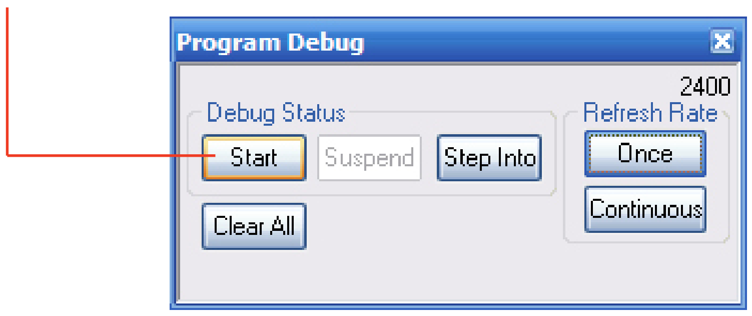
This section of code tells ROBOTC to set the power level of motorA and motorC to 0. This tells the robot to stop moving.

Go to “Robot”, “Compile and Download Program”.



```
orA] = 100;
orC] = 100;
(2000);

motor[motorA] = 0;
motor[motorC] = 0;
```

When the program is done downloading, click the “Start” button to run the program on your robot.



Speed and Direction **Half Motor Power**

In this exercise you will learn about the power level range of the RCX. You may not always want to move at full speed. Knowing how to adjust your motor's power level may prove to be very useful at times when precise movement is critical.

The power level of the motor for forward direction ranges from 0 to 100.
What power level would be half power for the RCX?

$$100 \;/\; 2 = 50$$

50 is the value that represents half motor power.

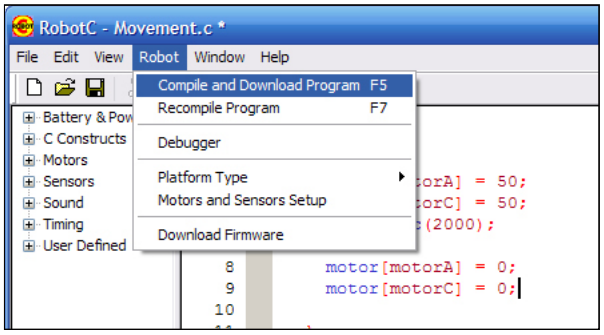
Change the motor power levels to be 50 instead of 100.

```
task main()
 {

    motor[motorA] = 50;
    motor[motorC] = 50;
    wait1Msec(2000);

    motor[motorA] = 0;
    motor[motorC] = 0;

 }
```

Run forward at half power for 2 seconds

Stop both motors (from the previous section)

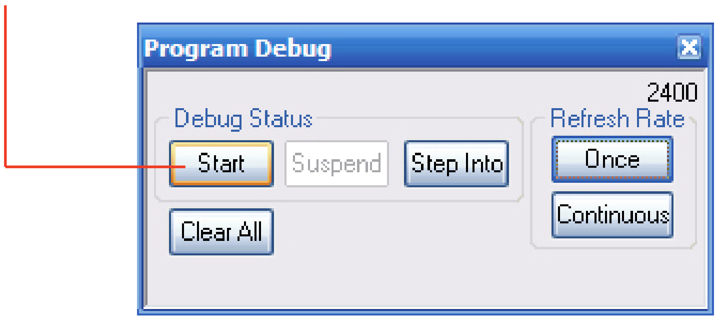
Now, compile and download the program.



```
orA] = 50;
orC] = 50;
(2000);

motor[motorA] = 0;
motor[motorC] = 0;
```

Speed and Direction Half Motor Power (cont.)

Click the "Start" button to run the program on your robot.



Notice how the robot moves forward slower than the first time it moved forward. In fact during the first foward movement, the robot moved twice as fast. You can adjust the level to be anything between 0 to 100, but what about the negative values? Move on to find out.

Speed and Direction Turn and Reverse Motor Direction

In this lesson, you will learn what negative motor powers do.

You know how to manipulate the value of the motors to go at half power and to stop, now what about negative power values? Adjust the power level so that it's back to full power forward.

```
task main()
 {

    motor[motorA] = 100;
    motor[motorC] = 100;
    wait1Msec(2000);

    motor[motorA] = 0;
    motor[motorC] = 0;

 }
```

This section of the code creates a forward movement for 2 seconds with full motor power.

Now add commands to have the power level of motor A to be -100, and motor C to be 100. Have the robot do this for .8 seconds. Stopping the motors after each movement can help them to be more precise.

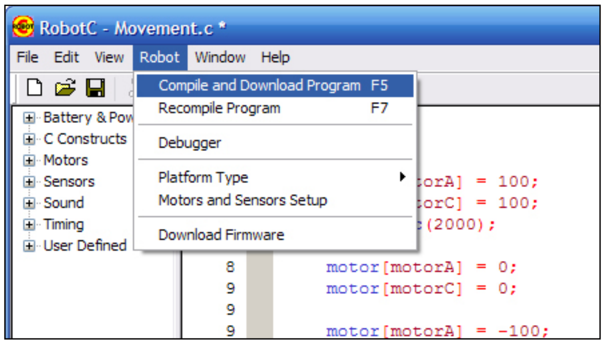
```
task main()
 {

    motor[motorA] = 100;
    motor[motorC] = 100;
    wait1Msec(2000);

    motor[motorA] = 0;
    motor[motorC] = 0;

    motor[motorA] = -100;
    motor[motorC] = 100;
    wait1Msec(800);

    motor[motorA] = 0;
    motor[motorC] = 0;

 }
```

A manual motor stop between other maneuvers can sometimes help to make those movements more defined and precise.

This section of the code tells ROBOTC to run motor A with full power reverse and motor C with full power forward.

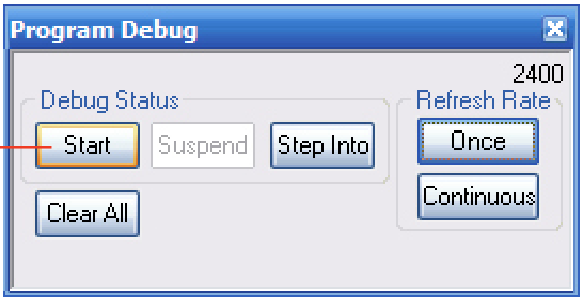
Speed and Direction Turn and Reverse Motor Direction (cont.)

Compile and download the program.



```
orA] = 100;
orC] = 100;
(2000);

motor[motorA] = 0;
motor[motorC] = 0;

motor[motorA] = -100;
```

Click the "Start" button to run the program on your robot.


Program Debug
2400
Debug Status
Start
Suspend
Step Into
Clear All
Refresh Rate
Once
Continuous