RobotC Language Guide

Mr. Michaud
Marist School

Elements of RobotC

- Comments
- Configuration
- Motor Commands
- Timing Commands
- Task main()
- Data Types
- Assigning Variables
- Reading Sensor Values
- Conditionals
- While Loops
- Functions
- Operators
- For Loops

Comments

- Lines of text with purpose to inform human reader of program content and intent
- Vital for collaboration (and grade!)

```
// Comments are denonted with double slash marks

/* Multi Line comments are
placed between a start slash startButton
and star slash */
```

Configuration

- Motors and Sensors must be configured with #pragma commands. This is done using the Motors and Sensors Setup option.

```
#pragma config(Sensor, dgtl1,  touch,          sensorTouch)
#pragma config(Sensor, dgtl2,  startButton,    sensorTouch)
#pragma config(Sensor, dgtl3,  sonar,          sensorSONAR_cm)
#pragma config(Motor,  port1,           right,         tmotorVex269, openLoop)
#pragma config(Motor,  port10,          left,          tmotorVex269, openLoop, reversed)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//
```

Motor Commands

Motor Commands:

```
motor[<motor name or port>] = power;
```

Motors are assigned values to start motion. Power ranges from -127 to 127.

-127 is full speed backwards

0 is Stop

127 is full speed forward

Examples:

```
motor[port1] = 100;

motor[right] = -50;
```

Timing Commands

```
wait1MSec(milliseconds);

wait10MSec(hundredth of a second);
```

Tells program to wait for the number of milliseconds. 1000 milliseconds equals 1 second.

Example:

```
motor[left] = 50;      // Motors On
motor[right] = 50;
wait1Msec(1000);       // Wait for 2 Seconds
motor[left] = 0;       // Motors Off
motor[right] = 0;
```

Task Main Function

- All RobotC programs must have a task main() function. The robot software will begin in the task main() to run commands.

Example:

```
task main() {
  motor[left] = 50;       // Motors On
  motor[right] = 50;
  wait1Msec(1000);        // Wait for 2 Seconds
  motor[left] = 0;        // Motors Off
  motor[right] = 0;
}
```

Data Types and Variables

bool	0	0 or 1 || true or false
byte	0000 0000	Whole number: -128 to 127
char	0000 0000	Can be a character value (ie "a")
float	0000 0000 0000 0000 0000 0000 0000 0000	decimal point number
long	0000 0000 0000 0000 0000 0000 0000 0000	Whole number: -2,147,483,648 to 2,147,483,647
int	0000 0000 0000 0000	Whole number: -32,768 to 32,767
ubyte	0000 0000	Whole number between 0 and 255

Assigning Variables:

```
int  power = 100;  // Creates an integer equal to 100

bool touched = false;

float batteryPower = 6.78;
long clockCycles = 84234;
```

Reading Sensor Values

`SensorInput[<sensor>]` command will return the value for the sensor plugged in to a given port.

Example: Sonar Sensor (named "sonar" plugged in to Digital Port 3:)

```
int currentDistance = SensorValue[sonar];
```

Or

```
int distance = SensorValue[dgtl3];
```

Control Statements with Conditionals

```
if (condition) {
// Response
}


while (condition)  {
// Actions
}
```

Touch Stop While Statement

```
// Wait until the Sensor is touched
while (SensorValue[touch] != 1) {
  // Do Nothing - Just waiting
}
```

Encoder Wait

```
// Main Task:  go Forward until the Motor Encoder registers 670 "Clicks"
task main() {
  // Reset the Encoder to 0
  SensorValue(Encoder) = 0;

  // Turn the motors On
  motor[right] = 75;
  motor[left] = 75;

  // Wait for the Encoder to be less then -670 "Clicks"
  while(SensorValue(Encoder) > -670) {
    // Do nothing - just wait

  }
  // Turn Motors Off
  motor[right] = 0;
  motor[left] = 0;
}
```

Declaring Functions and Tasks

- Task: Set of commands that can run concurrently
- Function: Encapsulates a set of commands
  - Void Functions: Perform a set of actions, but do not return a value
  - Return Functions: Perform calculations or measurements, and then return a value

Void Function Example

```
void driveSraight(int speed) {
  motor[left] = speed;
  motor[right] = speed;
}
```

Return Function

```
int getNearDistance() {
  int currentDistance = SensorValue[sonar];
  int nearDistance = currentDistance;
  int count = 0;
  pointTurn(50);
  while (count < 10000) {
    currentDistance = SensorValue[sonar];
    if (currentDistance < nearDistance) {
      nearDistance = currentDistance;
    }
    count++;
  }
  stopMotors();
  return nearDistance;
}
```

Calling a Function

```
task main() {
   driveStraight(50);
   wait1Msec(2000);
   driveStraight(0);
}
```

Operators

=	Assigns a value
==	Compares - "is equal to"
>	Greater than
<	Less than
!	Not
||	Or
&&	And
*	Multiply
/	Divide
++	Increase by 1
+	Addition
-	Subtraction
%	Modulo

For Loop

- Executes a series of code more than once. Uses an ‘index’ variable to count the cycles

```
for (int i = 0; i < 8; i++) {
  motor[left] = 100;
  motor[right] = -100;
  wait1Msec(500);
  motor[left] = 0;
  motor[right] = 0;

  motor[left] = -100;
  motor[right] = 100;
  wait1Msec(500);
  motor[left] = 0;
  motor[right] = 0;
}
```