Written by: Elizabeth Mabrey, Director of Storming Robots

Last update: July, 2017
Before you use this document:

Unless otherwise noted, Storming Robots retains an “All Rights Reserved” copyright, pursuant from the day this document was published by Storming Robots. You may copy, distribute and transmit the work IF AND ONLY IF appropriate credit is provided, and indicate if changes were made.

Scope of this document:

1. Very fundamental programming structure, and control structure with loop and if in ANSI-C
2. Fundamentals in basic robotics intrinsic motor functions and variables commonly used in RobotC.
3. Fundamentals in Usage of sensors from the Mindstorms kits in various applications.
4. Fundamentals in Modularization of functions.
5. Simple Gear Math to prepare you for triangulation.
6. Simple button operations

Assumptions:

This tutorial assumes:

- You know the very basic operation using NXT/EV3, such as how to select turn on/off, scroll through menu to select, etc.
- You have access to a basic NXT/EV3 kit to perform all the exercises in this packet.
- You must have a workable copy of RobotC. You may download the trial version online. If you choose to purchase it, you may want to go through our website to purchase it, as it will entitle you 20% refund (off from the full retail cost) from Storming Robots.
- You should have felt somewhat comfortable to use a computer, start a program, and basic navigation in your system.

Best practice:

- Examine all samples. **DO NOT** just copy, compile and run.
- Stay curious! Try out various actions.
- Use the help panel (on the left of RobotC)
- Practice K.I.S.S. Principle. “Complicated” does not necessarily mean “cool”.
- Often elegant and efficient means simplicity.
- Write “comment” for your code

How to use this tutorial:

1. Mainly meant for Storming Robots robotics club students. This is not meant for typical hand-holding style learning.
2. Chapters are organized in the following manner:
3. Major concepts you need to know before doing the exercises
   a) Download all samples from [http://learn.stormingrobots.com](http://learn.stormingrobots.com)
   b) Look at “BEST PRACTICE FOR LEARNING FROM THE SAMPLES” section.
c) Do the challenges

d) Challenges be listed in 3 levels of complexity – B, I & II, and I & II-Analytics.
## Contents

**SECTION I - BASIC C FUNDAMENTALS IN ROBOTC**  8

I - 1) Using RobotC the first time 8
    I - 1.A) Know about The IDE (Integrated Development Environment)  8
    I - 1.B) Download firmware  9

I - 2) Your First Program 10
    I - 2.A) Write your first program  10
    I - 2.B) About display:  11
    I - 2.C) Syntax Errors 11
    I - 2.D) Learn From Samples  13
    I - 2.E) Practice Exercise (B)  13

I - 3) Introduction to Variables  16
    I - 3.A) Variable/Data Types  16
    I - 3.B) To display variables  16
    I - 3.C) Local vs. Global Variables  17
    I - 3.D) RobotC Intrinsic Variables  18

I - 4) Basics in conditional / Boolean structure 18
    I - 4.A) Logical/Boolean Operators  18
    I - 4.B) Logical/Boolean Expressions  19

I - 5) Introduction to Loop Control Structure  21
    I - 5.A) "while" loop  21
    I - 5.B) "for" loop  21
    I - 5.C) Introduction to Unary Operation  22
    I - 5.D) switch... case... statements  22

I - 6) To look up available commands:  23
I - 7) - Debugger is your GREAT friend 23
    I - 7.A) Slow Debugging Techniques without debugger24
    I - 7.B) 1.9.2 Real-time Debugger 24
    I - 7.C) 1.9.3 Setting breakpoints 25

I - 8) Best practice for Learning from the samples 26
    I - 8.A) Learn From Samples 26

I - 9) MIni-challenge Exercises 27

SECTION II – MORE ABOUT STRUCTURE DESIGN IN ROBOTC30

II - 1) Primitive C vs. RobotC intrinsic types 30

II - 2) Bare basics in Array 31
    II - 2.A) Practice Exercise31

II - 3) More in control structure 32
    II - 3.A) Tenary Operation 32
    II - 3.B) Others 32

II - 4) RobotC Intrinsic Functions 33

II - 5) Writing Pseudo-Code 34
    II - 5.A) Why is it important? 34

II - 6) Learn From Samples 34

II - 7) Mini-challenge Exercises 35

SECTION III – MOTION NAVIGATION 37

III - 1) Access and Modify Motors Data37
    III - 1.A) Move the motors with simple movements 37
    III - 1.B) 3.1.1a Learning from the samples37

III - 2) 3.2 Mini-challenge Exercises 38

SECTION IV – BASIC SENSOR OPERATION 39

4.1) How to access basic sensors data 39
IV - 1.A) Steps that you should do in order to get the data from the basic sensors: 40

IV - 2) Touch Sensor 41
   IV - 2.A) Know About Your Sensor 41
   IV - 2.B) Learn from the samples 41

IV - 3) Light Sensor 42
   IV - 3.A) Know about your sensor 42
   IV - 3.B) Practice Calculating Thresholds 43
   IV - 3.C) Learn from the samples 43

IV - 4) Ultra Sonic Sensor 44
   IV - 4.A) Know About Your Sensor 44
   IV - 4.B) To access the sensor feedback 45
   IV - 4.C) Learn from The Samples 46

IV - 5) Motor Encoder 46
   IV - 5.A) Know about your sensor 46
   IV - 5.B) Move with Encoder Feedback 47
   IV - 5.C) Learn from the sample codes: 47

IV - 6) Mini-challenge Exercises 47

SECTION V – APPLIED PHYSICS: GEAR MATH 50

V - 1) Intro to Basic Gears Transmission 50

V - 2) Calculating Gear Ratio 51

V - 3) Convert Distance Traveled to Encoder degrees 51
   V - 3.A) To find EncPerCM for Gear Ratio 1:1 52
   V - 3.B) To find EncPerCM for Gear Ratio not 1:1 53
   V - 3.C) So the Formula 53

V - 4) Convert Body Rotation Degrees to Encoder Degrees 54
V - 4.A) Learn From Samples 55
V - 5) Mini-challenge Exercises 55

SECTION VI - MOTOR FEEDBACK CONTROL 56

VI - 1) Power vs. Speeds 56
VI - 2) Sync motors 56
VI - 3) Minimize overshooting 57
   VI - 3.A) Learn From Samples 59
VI - 4) Mini-challenge Exercises 59

SECTION VII – FUNCTIONS 60

VII - 2) Learn From Samples 62
VII - 3) Mini-challenge Exercises 62

SECTION VIII - BUTTONS CONTROL 64

VIII - 1) Learn From Samples 64
VIII - 2) Mini-challenge Exercises 64

SECTION IX - SOME DIFFERENCES IN EV3 65

SECTION X ERRATA 66
SECTION I - BASIC C FUNDAMENTALS IN ROBOTC

I - 1) USING ROBOTC THE FIRST TIME

I - 1.A) KNOW ABOUT THE IDE (INTEGRATED DEVELOPMENT ENVIRONMENT)

Using the software is quite intuitive. The following is the IDE window.

Basic operations: Here lists some commonly used menu selections:

- Create new file
- Always expand the Text Functions to allow quick look up intrinsic functions/variables.
- Should format your file properly.
- Compile first to check for any error.
- Download program to the robot for testing.
I - 1.B) **DOWNLOAD FIRMWARE**

1. You must download firmware before you can start using your NXT controller.
I - 2) YOUR FIRST PROGRAM

I - 2.A) WRITE YOUR FIRST PROGRAM

--- Entry point: task main()

| This is an entry point when a program is executed. |
| One program must have only a single “task main()” |
| A task must have its own “{” and “}” pair. |
| RobotC function to display some data on the LCD screen. |
| This one will display “Hello You!” on line 2. |
| You can display maximum 7 lines. |
| RobotC function to play a beep sound. |
| Tell the system to wait for 5 seconds. |

Type this program in:

```c
int main() {
    displayTextLine(2, "Hello You!");
    PlaySound(soundBeepBeep);
    delay(500);
}
```

---

For every {, there should have its matching closing }.

For every [, there should have its matching closing ].

For every (, there should have its matching closing ).

All expressions must end with ;, except for Boolean expressions and loop structure. We’ll cover this in the later section.
I - 2.B) ABOUT DISPLAY:

<table>
<thead>
<tr>
<th></th>
<th>EV3</th>
<th>NXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixels</td>
<td>178 x 128</td>
<td>100 x 60</td>
</tr>
<tr>
<td>Regular Text</td>
<td>16 lines (0 to 15)</td>
<td>8 lines</td>
</tr>
<tr>
<td>Big Text</td>
<td>8 lines 15 characters across. But, line # must be every other one, such as:</td>
<td>3 lines 8 characters across</td>
</tr>
</tbody>
</table>

I - 2.C) SYNTAX ERRORS

What if your code generates a lot of errors after you click on “compile”? This means your code is not compliant to the grammatical rules.

Compiler is a complex pattern matcher. It looks at your code more or less like a spoken language; which imposes a strict set of grammatical rules. Once it finds mismatch of an expected pattern, it generates an error.

You will need to know how to systematically find the syntax errors. The scope of this packet will not cover the diagnostic method extensively. However, here are a few tips that you may use to help you locate the errors.

Example:

Steps:
1. To locate the error: double click on the error line. The system will locate the line which prompts the compiler to generate the error.
2. Should start with the first error.
3. Read the error message.

4. If you have missing “brace”, use the to help you to find missing ‘{ ’ } ‘( ’ or ’) ’.
I - 2.D) LEARN FROM SAMPLES

1. Download all the sample codes. - [Download Chapter 1 Samples](http://learn.stormingrobots.com) from [http://learn.stormingrobots.com](http://learn.stormingrobots.com)
   – under RobotC Packet I.

2. **Important note**: All samples were written to interface with NXT, not EV3. The only changes you should need:

   ```
   nxtDisplay.... changed to  display....
   nxtDraw... changed to  draw....
   wait10MSec .. changed to  wait1Msec.,
   ```

   Better yet, look them up from the TextFunctions Pallette.

   3. Read, understand, and make your own copy.
   4. Compile.
   5. Correct any syntax errors.
   6. Download and Run.

Important : Be curious. After you fully understand and try out each of the exercises. Question yourself “what-ifs”.

I - 2.E) PRACTICE EXERCISE  (B)

1) Let’s have some fun drawing something on the screen.

Before you can do this, you must learn a bit about the NXT LCD screen.

   - Measured 100 wide by 64 pixels high display.

2) Draw a line and rectangle.
a) Type the code in the window.
b) Compile and Download your code.
c) Run your code at your NXT.

```
task main()
{
    nxtDisplayTextLine(1, "My name is Blah!");
    nxtDrawLine(1, 1, 100, 64); // draw line from (1,1) to (100,64)
    nxtDrawRect(10,60, 50, 5); // draw a rectangle with top left pt(10,60) and bottom right pt(50,5)
    wait1Msec(500);
}
```

Notice the “//”. It prefixes a line of comment. This is a great way to make your code easier to understand and for trouble shooting as well.

To comment a block, use “/*” and “*/”.

e.g.

```
/* program name: abc.c
   Date: such and such
   Author: Smiley Smith */
```

3) Create a rectangle with 10 pixels base, 5 pixels height.

e.g. `drawRect(10,50,20,45);`

4) Animate the rectangle that you created, e.g. moving it down 5 pixels at time until it disappears off the screen.

```
e.g.     drawRect(10,50,20,45);
          wait1Msec(500);
          eraseRect(10,50,20,45);
          drawRect(10,45,20,40);
          wait1Msec(500);
          eraseRect(10,45,20,40);
```

5) Create a Smiley face.

a) Before you do this, you should explore other Draw functions by looking up available Draw functions, and create a circle with 30 pixels as its diameter. If you do not see it on your left “Text Functions” Palette on your left, you should do one of the followings:

   a) Change yourself to Super User.
b) Select “Function Library (Text)”
I - 3) INTRODUCTION TO VARIABLES

I - 3.A) VARIABLE/DATA TYPES

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
<th>Sample expressions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>a data type only has two value value range: true or false</td>
<td>bool done;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bool tooLong=false;</td>
</tr>
<tr>
<td>int</td>
<td>a data type can hold 2 bytes long integer value, value range: An integer value with the range from -32767 to 32767.</td>
<td>int x;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int y=98;</td>
</tr>
<tr>
<td>float</td>
<td>Float : a data type can hold 2 bytes long float value allowing decimal places value range: An decimal value with the range from -32767.0 to 32767.0.</td>
<td>float fraction;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>float amount = 9.7512;</td>
</tr>
<tr>
<td>string</td>
<td>String : a data type can hold sequence of characters. The value must be represented in &quot; &quot;; e.g. string str = “Storming Robots”;</td>
<td>string name;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>string city=“Branchburg”</td>
</tr>
</tbody>
</table>

e.g.:

```
int xyz;       // this means “declaring” variable xyz as an integer type.
int abc=10;    // this means “declaring” as an integer type, and
               // “initializing” the abc with the value of 10.
```

I - 3.B) TO DISPLAY VARIABLES

Sample expressions:

```
int i=10, j=20;

displayTextLine(1, “this is robotc!”);  // display at line 1 with “this
                                          // is robotc"

displayTextLine(2, “i = %d; j = %d”, i, j); // display value of “i” and “j”.

i = 10; j = 20
```

// display value of “i” and “j” with each one in a column of 4 spaces, right justified.

```
displayTextLine(2, “i = <%4d; j = %4d>”, i, j);
```

```
i = < 10; j = 20>
```

// display value of “i” and “j” with each one in a column of 4 spaces, left justified.

```
displayTextLine(2, “i = <%-4d; j = %-4d>”, i, j);
```

```
i = <10 ; j = 20 >
```
I - 3.C) LOCAL VS. GLOBAL VARIABLES

- Understanding of ideas of "scope"

==============
task main()
{
    int i=4;  // this is a local
    i++;
}
==============

int i = 4;     // this is a global
task main()
{
    i++;
    ...
}
==============
task main()
{
    int i=29;
    {
        int i=4;
        displayTextLine(2, "%d", i); // this display 4
    }
    displayTextLine(2, "%d", i);  // this display 29
}
I - 3.D) **ROBOTC INTRINSIC VARIABLES**

There are many variables facilitated by RobotC. You should never name your own variables using RobotC intrinsic variables. You may see most of them from the “Text Functions” Pallette.

---

**e.g.**

<table>
<thead>
<tr>
<th>Math</th>
<th>Sensors</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands</td>
<td>Variables</td>
<td>Commands</td>
</tr>
<tr>
<td>PI</td>
<td>rand()</td>
<td>SensorRaw[]</td>
</tr>
<tr>
<td>random()</td>
<td>rand()</td>
<td>SensorValueFloat[]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

- int nPgmTime
- int nSysTime;

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>› The lower 16-bits of the internal 1-millisecond clock.</td>
<td>› The lower 16-bits of the internal 1-millisecond clock.</td>
</tr>
<tr>
<td>› Contains the value of the user program elapsed time.</td>
<td>› Contains the value of the system clock elapsed Time since NXT is powered on.</td>
</tr>
<tr>
<td>› Is reset to 0 when user program starts running.</td>
<td>› Is reset to 0 when NXT is first powered on.</td>
</tr>
<tr>
<td>› Does not increment during even debugging mode.</td>
<td>› Increments during even debugging mode.</td>
</tr>
</tbody>
</table>

---

I - 4) **BASICS IN CONDITIONAL / BOOLEAN STRUCTURE**

I - 4.A) **LOGICAL/BOOLEAN OPERATORS**

---

<table>
<thead>
<tr>
<th>Basic logical operators:</th>
</tr>
</thead>
<tbody>
<tr>
<td>= = Equal (the same)</td>
</tr>
<tr>
<td>( do not use “=” as “=” means assignment)</td>
</tr>
<tr>
<td>!= Not Equal</td>
</tr>
<tr>
<td>&lt; Less than</td>
</tr>
<tr>
<td>&gt; Greater than</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>&amp; &amp; And</td>
</tr>
</tbody>
</table>

---

Sample Expressions Boolean variables:

```python
bool incomplete = false;
```
bool done = false;

I - 4.B) LOGICAL/BOOLEAN EXPRESSIONS

a) if ... else if ... else ----------------------------

```
if (boolean expressions)
{
    .... if block
}
```

Where <boolean expressions> will contain a single or complex expressions to be evaluated.

The <if block> means a block of codes which will be executed ONLY if the <boolean expressions> are evaluated to be true.
### Sample Expressions: (code segment only)

```c
int x = 0;
x = x + 5;
if (x == 0)
{
    PlaySound(soundBeepBeep);
}
else if (x < 0)
{
    PlaySoundFile("Woops.rso");
}
else
{
    displayTextLine(3, "Ha! x = %d", x);
}
```

### Caution!
Watch your "{" and it's matching "}". To search for the matching "{" and "}", start with the innermost set.
We often refer this to conditional block.

#### More Boolean expressions samples:

<table>
<thead>
<tr>
<th>Boolean expression</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(x == 10)</code></td>
<td>if x equal to 10</td>
</tr>
<tr>
<td><code>(x &lt;= 10)</code></td>
<td>if x less than and equal to 10</td>
</tr>
<tr>
<td>`(x &gt; 10</td>
<td></td>
</tr>
<tr>
<td><code>(x &lt;= 10 &amp;&amp; y &lt;= 20)</code></td>
<td>if x &lt;= 10 and y &lt;= 20</td>
</tr>
<tr>
<td><code>(x &gt; 10 &amp;&amp; y &gt; 20)</code></td>
<td>if x &gt; 10 and y &gt; 20</td>
</tr>
<tr>
<td>`(x &lt;= 10</td>
<td></td>
</tr>
<tr>
<td>`(x &lt;= 10</td>
<td></td>
</tr>
</tbody>
</table>

Notice that these statements express the same set of conditions.
I - 5) INTRODUCTION TO LOOP CONTROL STRUCTURE

What is a loop: to repeat a set of instructions.

I - 5.A) “WHILE” LOOP

Expression format:

```
while ( conditional expression )
{
    ... while block
    ... this may contain one to many instructions
}
```

Sample expressions:

```
int x=0; // create variable x and initialize it to 0
while ( x< 20 ) // you should try this out
{
    displayTextLine(i, "x =%d", x);
    x = x+1;  // increments i by one after each iteration
}
```

I - 5.B) “FOR” LOOP

Expression format:

```
for ( <initial setup> ; <conditional expression >; <increment or decrement> ) {
    ... for block
}
```

Sample expression:

```
int i;
for (i = 0; i< 20; i=i+1)
{
    displayTextLine(i, "i =%d", i);
}
```
I - 5.C) INTRODUCTION TO UNARY OPERATION

A unary operation is an operation with only one operand, i.e. a single input. They are evaluated before other operations containing them.

In programming:

- ++i  same as  i = i + 1  same as  i += 1
- --i  same as  i = i - 1same as  i -= 1
- i /=2;  same as  i = i / 2
- i *=2;  same as  i = i * 2
- i %=2;  same as  i = i % 2

I - 5.D) SWITCH... CASE... STATEMENTS

(skip this section if you are absolute beginner)

```
switch ( x )
{
    case 1 :
    case block ... 
    displayTextLine(2, "x==1");
    break;
    case 2: 
    displayTextLine(2, "x==2");
    break;
    Default:
    displayTextLine(2, "x==%d",x);
    break;
}
```

The following structure will provide the same logic as the “switch” structure on the left.

```
if (x == 1) {
    displayTextLine(2, "x==1");
}
else if (x==2) {
    displayTextLine(2, "x==2");
}else{
    displayTextLine(2, "x==%d", x);
}
```
I - 6) TO LOOK UP AVAILABLE COMMANDS:

To look up available commands, go to the RobotC function library and select the desired command from the list. This will provide you with the syntax and usage information for that command.

I - 7) DEBUGGER IS YOUR GREAT FRIEND

Being able to debug a robotics program real-time is extremely helpful to locate logical errors in your code. RobotC’s debugger feature makes RobotC more superior than most other robotics development environment available. It is not common to find a robotics IDE which also comes with a user-friendly and easy to use debugger. This debugger does not support multi-tasking though. Here is a simple overview:

- Press F5 to compile and download.
- Click on “Step into”. You will see an arrow pointing at the current execute expression.
- Look at the “Output” dialog windows at the bottom and watch the variables that you are interested to watch.
- If the output windows did not show up, click on the “Robot” tab to select the “Global Variables”.
- You may even select the “NXT Remote Screen” to view the LCD screen display as well.
- Do not use others as they are for more advanced usage.
I - 7.A) SLOW DEBUGGING TECHNIQUES WITHOUT DEBUGGER

For diagnostic purpose:

- play different tones/sounds in a suspicious segment of code;
- add “display / print” statements to your program code at various points in your program; such as display some critical variables;

Both of the above techniques are available in most robotics programming language. However, a real-time debugger eliminates the need to resort to them. There’s no need to add code for debugging to your program. A built-in debugger provides better functionality without ever having to modify your source code!

There is also a built-in Debug Stream that you can use to keep track of your program from behind the scenes. For example, you could print a message to the Debug Stream when you enter and exit loops, functions, etc. Then you can view the cached Debug Stream to help in the debugging process.

I - 7.B) 1.9.2 REAL-TIME DEBUGGER

Debugging capability enables interactive real-time access to the robot as your program is running.

This process is extremely valuable as it may significantly reduce the time it takes to find and correct errors in your programs.

With ROBOTC's debugger you can:

- Start and stop your program execution from the PC
- "Single step" through your program executing one line of code at a time and examining the results (i.e. the values of variables) and the flow of execution.
- Define "breakpoints" in your code that will cause program execution to be suspended when they are reached during program execution
- Read and write the values of all the variables defined in your program
- Read the write the values of motors and sensors.
I - 7.C) 1.9.3 SETTING BREAKPOINTS

Breakpoints are a useful debugging tool. Breakpoints can be placed on any line of code and tell the controller to stop executing the program when that line is reached.

At the breakpoint, you can then check the status of the robot, variables, etc.

Set Breakpoints:

- Put cursor on the line you want the debugger to pause at.
- Click at grey bar or
- Click on line number. A context menu will appear. Select “Insert Breakpoint” to place a breakpoint at that line.

A red dot shows up to indicate a breakpoint is set for that line.

Stops at Breakpoints:

The highlighted line is where the execution pauses at.
I - 8) BEST PRACTICE FOR LEARNING FROM THE SAMPLES

If you have yet done all the sample codes from Chapter 1 downloaded from from http://robotc.stormingrobots.com, do it now.

IMPORTANT STEPS TO FOLLOW FOR BEGINNERS:

- If you have trouble remember the basic syntax, you should go back to read every single line of codes to understand the very basic and simple grammar/syntax.

I - 8.A) LEARN FROM SAMPLES

There are 15 sample files, which you learn from. You should download them, and work on each of them in the order they are listed. Following the guideline below:

—How to work with the samples:

1. **Understand each single expression!** Read the sample code to ensure you understand every single expression.
2. **Use Debugger** to view how values are being changed. The debugger is truly your BEST friend.
3. **Hand type the samples!** Especially for beginners, type in the sample. DO NOT use the computer “copy and paste”. Type them manually.
4. **Use your time wisely!** Compile, Run them and see result makes sense to you. If you simply copy and paste without understanding every single line of expression, you are wasting your time.
5. **Be inquisitive and Experiment!** Look into each single line of expression, make a couple of meaningful changes, then compile and download it again to your NXT.
6. **Be resourceful!** Use the “Function Library Panel” often to find available functions(commands/system variables.
7. **Again, Use Debugger** to view how values are being changed.
I - 9) Mini-challenge Exercises

Challenge 1: (B) Write a single program to do the following: (B)

- Display your name.
- Play a sound. Wait for one second.
- Display your school name on the next line.
- Play another sound. Wait for one second.
- Display your favorite event, followed by another sound. Wait for another one second.
- End the program.

Challenge 2: (B) Write a program to create a variable called “ct”. Create a loop so that your screen will display the following:

1
2
3
4
5
6
7
8

Challenge 3: (B)

1) Draw a small rectangle with coordinates with
7. top left point at (10, 20) and bottom right point at (25, 5)
8. e.g. drawRect(10, 20, 25, 5);
9.
2) Draw a rectangle with base 10 pixels and height 5 pixels with the top left point at (10, 20).
3) Draw a rectangle with base 10 pixels and height 5 pixels. This rectangle must be situated right at the center of your screen.

Challenge 4: (B) Download the sample code for performing factorial, simpleFactorialError.c. There is a syntax error and one logic error. Correct the error, compile, download and execute.
**CHALLENGE 5:** \( \text{(I)} \) Given the radius. Generate Circumference and Area of a circle. Only 2 decimal places. Do it in one’s increment four times. E.g.

<table>
<thead>
<tr>
<th>R</th>
<th>Cir</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>43.98</td>
<td>153.94</td>
</tr>
<tr>
<td>14</td>
<td>87.96</td>
<td>615.75</td>
</tr>
<tr>
<td>21</td>
<td>131.95</td>
<td>1385.44</td>
</tr>
<tr>
<td>28</td>
<td>175.93</td>
<td>2463.01</td>
</tr>
</tbody>
</table>

**CHALLENGE 6:** \( \text{(I)} \) Given the width and length of a rectangle. Generate the area of the biggest circle to fit inside this rectangle. E.g.

<table>
<thead>
<tr>
<th>W</th>
<th>L</th>
<th>CirA</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>3.14</td>
</tr>
</tbody>
</table>

**CHALLENGE 7:** \( \text{(I)} \) Draw the rectangle, as well as the painted complete circle fit inside the rectangle.
Challenge 8: (I) Given the measurement of base and height of a right angle triangle, find out the measurement of the hypotenuse of the triangle. (Pythagorean Theorem)

If you are a beginner, you should skip Chapter 2, and go right Chapter 3.
SECTION II – MORE ABOUT STRUCTURE DESIGN IN ROBOTC

This chapter is designed to strengthen your analytical and debugging skill in programming – usually for Level II. You may skip this chapter and proceed to Chapter 3. After you have completed this Packet I tutorial, you may return back to Chapter 2 to test your analytical skill.

Steps:

- Review each section.
- After you have read the section, Download the sample codes as recommended in each section.
- Always review and understand the purpose of the sample program. DO NOT just copy and paste.
- Compile, and test them with the controller to see if it runs as you have expected.
- For chapter II, you only need to work with the controller (the programming brick along), without building the robot.

II - 1) PRIMITIVE C VS. ROBOTC INTRINSIC TYPES

—Primitive C types
int : integer (2 bytes)
short : short integer (1 byte)
float : number with decimal places (2 bytes)
bool : true or false (1 byte)
char : a single character field (1 byte)

—RobotC specific data types
RobotC provides a rich set of its own data types. The most common data types:
tMotor : motor type
tSensors : sensor type
string : maximum 17 characters field
II - 2) BARE BASICS IN ARRAY

- systematic representation for a sequence of same kind of data.

Example 1:

```c
char grades[5]; // allocate 5 elements for "grades". Each grade takes up one byte (size of "char" type)
```

<table>
<thead>
<tr>
<th>bytes</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
</table>

```c
grades[0] = 'A'; grades[1] = 'B'; ... etc.
```

Example 2:

```c
int age[10]; // allocate 10 elements for "age". Each age takes up 4 bytes (size of "int" type)
```

<table>
<thead>
<tr>
<th>bytes</th>
<th>0...3</th>
<th>4...7</th>
<th>8...11</th>
<th>12...15</th>
<th>16...19</th>
<th>20...23</th>
<th>24...27</th>
<th>28...31</th>
<th>32...35</th>
<th>36...39</th>
</tr>
</thead>
</table>

```c
age[0] = 50; grades[1] = 60; ... etc.
```

```c
short motor[3]; // allocate 3 elements for "grades". Each grade takes up one byte (size of "char" type)
```

<table>
<thead>
<tr>
<th>bytes</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
</table>

```c
```

Beware, do NOT use more than you allocate. It will crash your program or generate unpredictable results.

II - 2.A) PRACTICE EXERCISE

Exercises pertaining array will be provide in later chapters when you use Array for motors and sensors.

More about array data structure will be covered in Tutorial Packet II.
II - 3) **MORE IN CONTROL STRUCTURE**

II - 3.A) **TENARY OPERATION**

![Ternary Operation Example](image)

```c
result = (a > b ? x : y);
```

```c
if (a > b) {
    result = x;
}
else {
    result = y;
}
```

II - 3.B) **OTHERS**

There are many others pertaining to:

- File I/O Access (will be covered in RobotC Tutorial Packet II)
- Bluetooth access (will be covered in Bluetooth Tutorial Packet)
- LCD Graphics types (will not be covered in our tutorial. However, you may review RobotC samples codes.)
- Many others (not covered as they are for platforms other than NXT)
II - 4) **ROBOTC INTRINSIC FUNCTIONS**

RobotC provide a rich set of robotics functions.

To view them all:

---

**Functions template:**

```c
<Data type>  <function name> ( parameters )
{
   Instructions...
}
```

- **e.g.** `void wait1Msec ( 5000 )`
  - `wait1Msec`: is the function name
  - `5000`: parameter
  - `void`: return data type is void. This means it does not return anything.

- **e.g.** `void drawRect (10, 20, 35, 6)`
  - `drawRect`: is the function name
  - `10, 20, 35, 6`: four parameters which are all “int” type
  - `void`: return data type is void. This means it does not return anything.

- **e.g.** `int strcmp ( "aaaa", "aaab")`
  - `strcmp`: is the function name
  - "aaaa", "aaab": two string parameters
  - `int`: return data type is int. This means it returns integer value which equals value difference between “aaaa” and “aaab”.

---

In this section, you will do more practice to reinforce your basic programming skill before using the actual motor navigation and sensors feedback.
II - 5) **WRITING PSEUDO-CODE**

Pseudo Code is:

- natural language constructs modeled to look like statements available in many programming languages
- design a sequence of instructions to perform some task(s)
- should always do this before you start programming
- e.g. calculating factorial for \( N! \)

```
set result to 1
for I from N to 1
    result = result * I
```

- e.g. Euclidean algorithm to calculate G.C.D. for number \( N \), \& \( M \)

```
if M > N
    swap M and N
while N != 0
    remainder = N % M
    M = remainder
    N = M
end while
Answer = M
```

II - 5.A) **WHY IS IT IMPORTANT?**

- Demonstrate the detailed steps to perform an algorithm.
- Break down the abstract of an algorithm
- Allow us to focus on developing the algorithm without worrying about the actual coding semantics.
- Is very useful for error tracking, trouble-shooting, etc.
- Help to modularize or dissemble a complex task into small tasks

II - 6) **LEARN FROM SAMPLES**

This exercises are only for those who have the proper math level skill and want to take on more programming challenge. You may skip this.

Remember the Best Practice rules thumb stated at the beginning of chapter 1.

- 01Factorial.c: change the factorial number and run it again
- 02FactorialError.c: find the error and correct it.
- 03Triangle.c: practice you loop structure. Involving nested loops. Need design skill.
- 04DrawSpiral.c : draw spiral. This requires knowledge in trigonometry, design skill, and loops structure.
II - 7) MINI-CHALLENGE Exercises

Note: If you have absolutely no idea how to get started with the following projects, you should do the following:

- go back the samples, study the codes
- be inquisitive and make relevant changes
- do not be afraid of errors
- step through the codes with the debugger window on.
- Look at the variables value and see they behave as you expected.

**Challenge 1: Change the sound and delay.**

a) Modify p1PlaySimpleSound.c to play Beep sound 5 times with 1 beep in-between.
b) Modify the p1Display02.c to display the following using variable like this:

```
I   I*I
---
2   4
3   9
12.
```
c) Modify p1Display02.c using loop.

Hint:
Display... (num+2, "%d %d", num, num*num);

**Challenge 2: Write a program to calculate factorial starting with value 3 and display like the following:**

At program starts up After 1 second After 1 second After 1 second
The screen shows... The screen shows... The screen shows... the screen shows...

<table>
<thead>
<tr>
<th>X</th>
<th>X!</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>X</td>
<td>X!</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>120</td>
</tr>
<tr>
<td>X</td>
<td>X!</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>120</td>
</tr>
<tr>
<td>6</td>
<td>720</td>
</tr>
</tbody>
</table>

**Pseudo code:**

set x to 1
while (x <= 6)
perform X! (i.e. factorial of ‘X’)
update the display with both value of X and X! (factorial of X)
wait 1 sec

**Challenge 3: Write the Euclidean algorithm**

- to calculate G.C.D. for number 36 & 96 based on the pseudo code provided in section 2.3.
SECTION III – MOTION NAVIGATION

III - 1) ACCESS AND MODIFY MOTORS DATA

III - 1.A) MOVE THE MOTORS WITH SIMPLE MOVEMENTS

(note: NXT controller supports 3 motor ports, while EV3 supports 4. This makes no difference in programming concepts.)

RobotC variables for motor ports: motorA, motorB, motorC, motorD (EV3 only)

RobotC data type: tMotor

Valid data range: -100 to 100

reverse to forward direction of -100% to 100% power level)

0 == stop

For example:

motor[motorA] = 50; // start motor A with 50% power level

III - 1.B) 3.1.1A LEARNING FROM THE SAMPLES

Download the chapter 3 samples from the SR learning site – http://learn.stormingrobots.com. All samples focus on basic movements with sample control structure.
III - 2) 3.2 MINI-CHALLENGE EXERCISES

1) Write a program to make your robot run in a square using Wait1Msec(...) or delay(...). (B)

2) Write a program to make your robot in Ladder style using Wait1Msec(...) or delay(...). (B-I)

3) (B-I) Write a program to wiggle left and right for 5 times and back to the starting orientation.

   ...repeat this 5 times... and end up back to the starting orientation.

   Sample Pseudo code:

   tm = 500
   Left for tm milliseconds
   While ct < 5
   right for tm*2 msec
   left for tm*2 msec
   ct + 1
   left for tm sec

4) (I-II) Program your robot to run to a target distance as fast as possible, but be able to stop almost right at the target without overshooting. Set target to be 50cm, then try 100cm, then try 200 cm.

   Hint: think about gradual deceleration.

Note: You may wonder what encoder value should be used for 50cm, or 100cm, or 200cm. For now, you will just use any intuitive number(s) with trial and errors method. You will learn how to perform the gear math to convert the distance to encoder value.
SECTION IV – BASIC SENSOR OPERATION

4.1) HOW TO ACCESS BASIC SENSORS DATA

MUST READ before you start working with sensors.

There are three things you should always start with:

1- Know which sensor port you are using

   **Sensor ports:** S1, S2, S3, S4

   e.g.  `tSensors myLeftEye = S2;`

2- Know RobotC Sensor Types for the sensor

   This allows the RobotC to generate normalized (or commonly known/cooked) value of a sensor

   You must use a value sensor type provided by RobotC. Some commonly used RobotC “sensors type” constants:

   ```
   sensorTouch // for both RCX & NXT touch sensors
   sensorReflection // for RCX light sensors
   sensorRotation // for RCX rotation sensors
   sensorLightActive // NXT light sensor only; sensor LED light on
   sensorLightInactive // NXT light sensor only; sensor LED light off
   sensorColorNxtFULL // NXT color sensor
   sensorEV3_Touch // EV3 touch
   sensorEV3_Color // EV3 color sensor
   sensorEV3_Ultrasonic // EV3
   ```

   You should be able to find many more at the **TSensorTypes.h** under the RobotC installation folder.

   e.g.  `SensorType[myLeftEye] = sensorLightActive;`

   Note:

   `sensorI2CCustomFast // for other I2Cbase sensors (will be covered in advanced Packets)`

3- Know valid data range. Various sensors have different valid range.
IV - 1.A) STEPS THAT YOU SHOULD DO IN ORDER TO GET THE DATA FROM THE BASIC SENSORS:

1st step: SET the sensor type.

2nd step: ACCESS/GET the value

In the following sections, you will learn how to program to utilize feedback from the basic NXT sensors available from the NXT package. They are:

- Touch sensor
- Light sensor
- Ultrasonic sensor
- Rotation sensor
  (This is embedded inside each motor. We should call it as Encoder. This will be covered more extensively in the next couple of chapters.)
IV - 2) TOUCH SENSOR

IV - 2.A) KNOW ABOUT YOUR SENSOR

Sensor Type:  

sensorTouch

Valid data range:  

0 means released  
1 means pushed in

—Examples:

Sample 1:

```
sensorType[S1] = sensorTouch;
while (SensorValue[S1]==0) {
    displayTextLine(4, "Released!");
}
displayTextLine(4, "Pushed!");
```

Sample 2:

```
const tSensor leftBumper = S1;
int value=0;
sensorType[leftBumper] = sensorTouch;
while (value==0) {
    displayTextLine(4, "Released!");
    value = SensorValue[leftBumper];
}
displayTextLine(4, "Pushed!");
```

Sample 3:

May express the while loop as following:

```
while ( (value=SensorValue[leftBumper]) == 0)
    displayTextLine(4, "Released!");
displayTextLine(4, "Pushed!");
```

IV - 2.B) LEARN FROM THE SAMPLES

Again, you should review the code to understand the actions.

Some samples do not use motors in order to demonstrate the “debouncing” issue. Again, It is important to analyze and understand by utilizing the debugger.

Do note that the link will download all samples for Chapter.
IV - 3) LIGHT SENSOR

IV - 3.A) KNOW ABOUT YOUR SENSOR

The light sensor measures the amount of light that it sees. It reports the amount of “reflection” represented by a number between 0 (total darkness) and 100 (very bright).

The light sensor uses its own light source, a red light emitting diode (LED), to illuminate a small area in front of its receiver, called Photo-transistor.

Sensor Type: sensorLightActive

Valid data range: 0 <= value <=100; 0==darkest and 100==brightest

Examples:

1) const tSensors Eye = S1;
   SensorType[Eye] = sensorLightActive;
   displayTextLine(4, “See %d“, sensorValue[Eye]);

2) If S1 sees darker than
e.g.
   if (SensorValue[S1] < 43) { ...
   }

3) while both S1 and S2 see brighter than
e.g.
   while (SensorValue[S1] > 43 && SensorValue[S2] > 45) { ...;
   ...
   }

4) If see brighter than
e.g.
   if (SensorValue[S1] > 43) { ...;
   ...
   }

5) while see brighter than
IV - 3.B) PRACTICE CALCULATING THRESHOLDS

Before you program your light sensor, you must find out the proper threshold. One simple way is to take the average of the darkest and the brightest.

value of black: ______  value of white: ________  average : _______

Think About this:

The light value can easily be affected by the lighting around the robot.

Try the following 3 things:

1) Read the light value of the black tape in well-lit classroom
2) Turn off the light, read the light value of the same black area.
3) Take your robot and this paper outside under the sun, read the light value of the same black area again.

One simple rule of thumb using light sensor is to locate your sensor in such a way that:

- Light source is consistent
- Distance from it to your target surface consistent.

IV - 3.C) LEARN FROM THE SAMPLES

Work on the 8 samples downloaded from http://learn.stormingrobots.com. Remember the Best Practice guideline. One of the samples show you how to have your own robot to calibrate the threshold.

e.g.

if (SensorValue[S1] > 43) { ...;

... }


IV - 4) ULTRA SONIC SENSOR

IV - 4.A) KNOW ABOUT YOUR SENSOR

—Special thing you should know about ultrasonic wave

✦ Detect angles

It uses sound propagation technology, the same scientific principle as bats: it measures distance by calculating the time it takes for a sound wave to hit an object and come back – just like an echo.

Do note that the sound wave spans out. For Mindstorms ultra-sonic sensor, it spans out about 25 degrees. That means it will detect objects within that spanning degrees. You use this to detect the proximity, not what is right in front of you such as laser sensor.

Actual distance most likely will not equate exactly to the perpendicular distance from the sensor; especially when the object is farther away from the sensor. The closer it is to the sensor, the more accurate straight ahead distance you will get.

✦ Object Shape

Flat objects with hard surfaces provide the best readings. Curved objects (e.g. a ball), or from very thin and small objects can be difficult for the sensor to produce reading reasonable to us.

✦ Observe the following data

Sensor is mounted on a station which will pivot 180 degrees. It records the sensor feedback from 0° to 179°
✧ **Surface Material**

Objects covered with soft fabrics, such as carpet, will absorb the sound wave.

✧ **May intersect noise**

If you choose to use two+ ultrasonic sensor, you will need to put them far apart enough so that one won’t intersect with another.

Do some simple trigonometry work to calculate the minimum distance between the 2 sensors vs. your target max and min distance.

✧ **I²C**

This is actually an I2C sensor. Click here to view the official LEGO Mindstorms Ultrasonic Sensor I2C Communication Protocol Specification. If you decide to interface this with other controller such as Arduino, you will need to access the Mindstorms SDC to see how to wire them.

IV - 4.B) **To access the sensor feedback**

**Sensor Type:** SensorSONAR

**Valid data range:** 7 <= value <= 60 ; 255 for anything farther than

Note that battery level may affect the reliability, even within 50. We have had experienced that I could detect only up to 35 when the battery is running low.

Caution: Sound wave will not work on surface which absorbs sound waves, such as carpet. In addition, it works better if the target surface is flat instead of curve such as a cylinder.

**Important things you should know before using any ultrasonic sensors**

The following applies to any sensors using ultrasonic wave, not just pertaining to Mindstorms sensors.

1) const tSensors distEye = S1;
   SensorType[distEye] = sensorSONAR;
   displayTextLine(4, “See %d”, sensorValue[DistEye]);

2) If sensor detects an object closer than 7cm
   e.g.
   if (SensorValue[S1] < 7) {
       ...
   }
IV - 4.C) LEARN FROM THE SAMPLES

If you have worked on the samples before this section, you should have downloaded the encoder samples as well. Work on the sensors samples.

IV - 5) MOTOR ENCODER

IV - 5.A) KNOW ABOUT YOUR SENSOR

Sensor Type: N/A

Valid data range: -32767 <= value <= 32767 (can be in centimeters or Inches)

The NXT motor has a built in tachometer that keeps track of the current angle (in degrees) of the motor axle. Or, people usually called it rotation sensor or encoder. The terms are interchangeable.

Tachometer is the technical mechanic name.

RobotC uses a variable nMotorEncoder to hold the rotation degree for the motor.

e.g.

```plaintext
nMotorEncoder[motorA] = 0;  // reset encoder for motor A
nMotorEncoder[motorB] = 0;  // reset encoder for motor B

if(nMotorEncoder[motorA] < 1800)
{
    ... // if encoder rotates < 1800, do step(s) in the "{}"
}
while(nMotorEncoder[motorA] < 1800)  // while encoder A rotates < 1800,
{
    ... // if encoder rotates < 1800, do step(s) in the "{}",
        Go back to checking if encoder still rotates < 1800
}
    // while encoder A < 1800 or encoder B > -1800)
{
    ... // if the condition is true, do step(s) in the "{}",
        Go back to check whether the condition still true
}
```
Important: The encoder embedded inside the motor is to detect the rotation of the motor itself, “not” the robot’s physical body itself.

IV - 5.B) MOVE WITH ENCODER FEEDBACK

This will allow you to tell how many degrees your motor turns. This will be covered in the next chapter when we cover sensors.

RobotC intrinsics variables: nMotorEncoder[ ... ]

e.g. nMotorEncoder[motorA] = 0; // reset the motor encoder count

if ( nMotorEncoder[motorA] < 360) // if the motor A rotates less than 360 encoders.

*** Note that this just so happens to be == 360 degrees of motor rotation. It was made that way for youngsters to conceptualize the rotational degrees of the motor. Most encoders for motors do not have such a nice 360 encoder equating one full motor revolution.

IV - 5.C) LEARN FROM THE SAMPLE CODES:

Again, if you have worked on the samples before this section, you should download the encoder samples as well. ???

IV - 6) MINI-CHALLENGE EXERCISES

Challenge 1: (B)

1. goBackWEnc.c : run backward for any specific distance based on the conversion using target control.
2. goRightWEnc.c : make right turn for any specific distance based on the conversion using target control.

Challenge 2: (B)

1. modify all the forward/backward/right/left into individual functions and integrated into a single program.
2. runSquare.c : program your robot to run in a square using the 4 functions stated above. Use conversion routines
3. rewrite the forward/backward functions to a single goStraight function which can do either forward or backward
4. rewrite the right/left turn functions to a single turn function which can do either right or left turn.

Challenge 3: (B-I)

Line tracking until see an obstacle. Then, it should stop at the obstacle.

Challenge 4: (I-II)
Line tracking until see an obstacle. Then, it should stop at the obstacle. After that, it should turn around and line trace back to the startup.

**Challenge 5: (I-II)**

Like (4), except:

There will be a silver tape at the startup.

Your robot must stop at the silver tape on the returing path i.e. after it sees the obect.

**Challenge 5: (I-II - ANALYTICS)**

- Line tracking until see an obstacle. Then, it should stop at the obstacle.
- get around it and get back on the line behind the obstacle.

**Pseudo Code:**

Modularize your tasks!

**Step 1) Write a high level analysis:**

```pseudo-code
While true {
    If sonar > 15cm (t1)
        Line trace
        Continue
    (t2) Back up a few cm
    (t3) Turn bot to face to the side to prepare to go around the object (assuming around left side of the object)
    (t4) Circle around the object until it does not see line
}
Where t# == subtasks
```

**Step 2) Broke down the large task into small ones. Sample as following:**

(t1) you have done line trace moves in the past exercises..

(t2) while encoder [ ... ] < N
    Moves back

(t3) while sonar < 255
    LeftMotor -20% , RightMotor 20% // Left turn until bot does not see object
    Left turn a bit more to ensure bot will have enough room for your robot to manoeuvre around.

(t4) while not see line
    LeftMotor 20% , RightMotor 70% // sample only : 2:7 ( L : R ) == turn around left side of the object.
Note: The power levels may very likely change depending on your gears system, tire, chassis sides, etc.

The purpose of tracing the object instead of “dead reckoning” because you would never know the size of the object. However, just for simplicity sake, you can use a fixed sized object.

**Challenge 6:** (II - ANALYTICS)

Like (5), except:
- You should calculate the ratio of the left vs right motor based on the size of the object.
- Modify your subtask t4 (see above). You might have to modify t3 as well.

**Challenge 7:** (II-III - ANALYTICS)

Like (6), except you will use feedback control (P.I.D.) to trace along the object instead.
SECTION V – APPLIED PHYSICS: GEAR MATH

V - 1) INTRO TO BASIC GEARS TRANSMISSION

— BASICS TERMS
A gear is a set of toothed wheels (gear wheels or cog-wheels) that work together to transmit movement.

Many of the everyday mechanisms and devices we commonly use contain gear wheels. These include bicycles, cars, and can-openers.

— FUNCTIONS OF GEARS:
1. To change the position of a rotating movement. (This is sometimes called applying the rotation at a distance.)
2. To change the direction of rotation.
3. To increase or decrease speed of rotation.
4. To increase turning force (This is sometimes called torque.)

— DRIVER/INPUT
The name for a gear wheel that is turned by an outside force (such as that from a motor or from a person turning a handle) and that also turns at least one other gear wheel.

— DRIVEN/FOLLOWER/OUTPUT
The name for a gear wheel that is turned by another gear wheel.

— GEAR RATIO
A proportion used to compare how two meshed gear wheels move relative to each other. For gears, use the number of teeth for calculation. For pulleys, use its diameter for calculation.

— GEARING DOWN
An arrangement in which a small driver turns a large follower, resulting in a slowing down of the turning. Gearing down produces a powerful turning force (torque).

— GEARING UP
As arrangement in which a large driver turns a small follower, resulting in a speeding up of the turning. Gearing up reduces the turning force.

— IDLER GEAR
The name for a gear wheel that is meshed between a driver and a follower. It does not mean it does not move. It is called idler gear because it does not affect the final gear ratio.
**WARM UP INFORMATION**

- Two meshed gear wheels turn in opposite directions.
- When two gear wheels are mounted on the same axle, they both turn at the same speed, regardless of their sizes.
- Gears have a trade-off with turning force (torque) and turning speed.
- In general, torque ↑ speed ↓ when torque ↓ speed ↑

**V - 2) CALCULATING GEAR RATIO**

Torque (twisting or turning force) is inversely proportional to speed.

In order to determine both the speed and force of rotating axles, we need to calculate the Gear Ratio.

The gear ratio is the ratio of the number of teeth on each gear. Here is a gear with 8 teeth meshed with a gear with 40 teeth.

What does this gear ratio 5:1 tell us?

- The Input-driver gear will rotate 5X when the output-follower gear rotates 1X
- The Input-driver gear will rotate 5X faster than the output-follower gear.
- This contraption is meant to increase torque 20.

**V - 3) CONVERT DISTANCE TRAVELED TO ENCODER DEGREES**

In order to convert a target distance, e.g. 10cm, to a required encoder value, you need to know:
“How many encoder degrees to run 1cm? ”. Let call it EncPerCM.

**V - 3.A) To find EncPerCM for Gear Ratio 1:1**

Step 1: Find the tire’s circumference:

Let \( tD = \) diameter of the tire in centimeter

One tire revolution = \( tD \times \pi \) cm

Step 2: Find the Gear Ratio (GR)

Gear Ratio = \( \frac{\text{follower gear}}{\text{driver gear}} \)

See the sample setup on the right:

GR = 1 : 1 or 1/1

Step 3: Find encoder degrees per 1 unit of distance

Say, we are interested in \( \text{“cm”} \), centimeter. Thus, you need find out:

\[
1 \text{ cm} = \text{?} \text{ encoder degrees} \quad \text{. Let’s call it EncDegreesPerCM}
\]

\[
\begin{align*}
&\cdot \cdot \cdot \text{One tire revolution} = tD \times \pi \text{ cm} \\
&= 360^\circ \\
&\cdot \cdot \cdot tD \times \pi \text{ cm} = 360^\circ \\
&1 \text{ cm} = \frac{360^\circ}{tD \times \pi}
\end{align*}
\]

so in your program, you need to create a variable to hold this value. For example:

```c
float tireD = 8; // if the tire diameter = 8 cm
int EncoderPerCM = 360 / (tireD * PI);
```

So, In order to travel 20 cm, total encoder value = 20 * EncDegreePerCM;
V - 3.B) TO FIND ENCPerCM FOR GEAR RATIO NOT 1:1

GR = 1/2
Motor rotates 2X == Tire rotates 1X

\[
360^\circ = \text{tireD} \pi \times \text{GR cm}
\]

\[
\text{EncoderPerCM} = \frac{360^\circ \times \text{GR}}{\text{tireD}\pi}
\]

In order to travel 20 cm, rotational degrees = \(20 \times 360^\circ \times 2 / \pi \)

V - 3.C) SO THE FORMULA

The formula to convert distance traveled to encoder value (rotational degrees in this case):

Postulate 1: Encoder pre CM = \(360 \times \text{GearRatio} / (\text{tireD}\pi)\)

To travel \(x\) cm, motor requires to turn \(x \times 360 \times \text{GearRatio} / (\text{tireD}\pi)\) encoder
V - 4) CONVERT BODY ROTATION DEGREES TO ENCODER DEGREES

Remember: It is all about “distance” it needs to travel, no matter what shape the travel pattern is.

Therefore, the 1st step is to find out the distance for Chassis Turning.

Steps:

1) Find out the traveled distance:
   a) Measure the wheel base (W).
   b) Calculate the distance traveled by one full chassis revolution,
      - circumference with the wheel base as the diameter of the Circle it travels.
      - \( W \pi \) (let’s create a variable called, \( \text{FullTurnCM} \))
   c) Get the fraction of the body turns out of a 360 full body rotation, e.g. 90° = \( \frac{1}{4} \) of 360°
      - i.e. \( \text{fraction} = \text{target degree of turn} / 360 \)
   d) \( \text{FullTurnCM} \times \text{fraction} \)

2) Calculate Encoder degrees per cm, (let’s call it \( \text{EncoderPerCM} \)).
   - \( 360 \times \text{GR} / (\text{tireD} \times \pi) \)

3) Total degrees (encoder value) needed to travel for 90 degrees
   - \( \text{FullTurnCM} \times \text{fraction} \times \text{EncoderPerCM} \)

You can easily generate another formula:

One full body point turn rotation = \( W \pi \) cm (refer to figure 3)
\[
1 \text{ cm } = 360 \times \text{GearRatio} / (\text{tireD} \times \pi) \text{ (from postulate 1)}
\]
\[
\therefore \text{ Full body point turn rotation } = W \times \pi \text{ cm}
\]
\[
\therefore = W \times \pi \times 360 \times \text{GearRatio} / (\text{tireD} \times \pi)
\]
\[
= W \times 360 \times \text{GearRatio} / \text{tireD}
\]
\[
1 \text{ body point turn rotation } = (W \times 360 \times \text{GearRatio} / \text{tireD}) / 360
\]
\[
= W \times \text{GearRatio} / \text{tireD}
\]

The formula to convert body rotation to encoder value:

Postulate 2: 1° of body point turn rotation = \( W \times \text{GearRatio} / \text{tireD} \) encoder degrees
So for \( X^\circ = X \times W / (D \times \text{GearRatio}) \) encoder degrees
To turn $\theta^\circ$ turn left turn:

- Right motor requires to turn $\theta \times W \times \text{GearRatio} / \text{tireD}$ encoder degrees
- Left motor requires to turn $-1 \times \theta \times W \times \text{GearRatio} / \text{tireD}$ encoder degrees

V - 4.A) LEARN FROM SAMPLES

There are 6 sample files for you to learn from. File names are very self-explanatory as far as what the file is for. You should work on them in the order it is listed. Again, be inquisitive, and experiment. You will need a ruler for all exercises for this chapter.

V - 5) MINI-CHALLENGE EXERCISES

Now, write 2 functions: (II - ANALYTICS)

- goBackwardWMath()
- goRightWMath()

Hints:

- You may write separate programs or functions within a single program.
- Make a function to be capable of handle both going forward and backward
SECTION VI - MOTOR FEEDBACK CONTROL

VI - 1) POWER VS. SPEEDS

You can set the Motor power with motor[], not speeds.

Not all motors are created equal. Various factors in the robot’s construction, and the manufacturing process for the motors themselves cause different amounts of energy to be lost to friction in each motor.

This means that even though both motors start with the same power at the plug, the amount of power that reaches the wheel to move the robot can vary quite a bit. Even with the same power being applied, speeds may differ.

In order to make two motors run the same speeds, you need to do some work, that is called motor feedback control.

VI - 2) SYNC MOTORS

be able to travel a straight line. This is because both motors are not rotating at exactly the same speed at all time. It requires a motor feedback controller process to make this happen.

RobotC provides such feature to synchronize the power levels between the two motors.

Important note:

▷ Once you activate the feedback control, you must not modify the secondary motor.
▷ Do set master motor to “100” power level. If it is already 100, the sync process will not be able to give the secondary motor higher power level.

Steps:

1. Reset both motor encoder values
2. To activate the feedback control:

   e.g. nSyncedMotors = synchAC;    // Activate the feedback control.
       // Motor ‘A’ is the master, ‘C’ is the secondary.

3. To set “nSyncedTurnRatio”:

   e.g.
   nSyncedTurnRatio=100;
       // means secondary motor power level = 100% of master motor power level;
       // i.e. go forward
   nSyncedTurnRatio=-100;
       // means secondary motor power level = -100% of master motor power level;
4. Start your motor for a specific # of encoder value.
   - **Important:** by activating the feedback control, the process assumes duty to adjust the power of the secondary motor. Therefore, you MUST not make any modification to the secondary encoder, nor its direction.
5. Stop ONLY the master motor.
6. To deactivate:

   \[ \text{NSyncedMotors} = \text{synchNone}; // No motor synchronization } \]

### VI - 3) MINIMIZE OVERSHOOTING

This will allow you to predefine the target encoder values and minimize overshooting.

RobotC intrinsic: \( \text{nMotorEncoderTarget[...]} \)

**e.g.**

\[
\text{nMotorEncoderTarget[motorA] = 360; // preset the target encoder value to 360} \\
\text{    } // however, this does not start the motor \\
\text{motor[motorA] = 50; // make sure you will start the motor} \\
\text{    } // \text{NXT will decelerate and stop your robot for you robot} \\
\text{    } // right about at 360 encoder value \\
\text{nMotorRunState[ motorA ] // the current status of the motor, such as ramping up, or full speed, stop, etc.} \\
\text{runStateldile // system value to state the motor has stopped} \\
\]

<table>
<thead>
<tr>
<th>Motor Feedback Control basics:</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. LM = motorA (the left motor) and RM = motorC (the right motor)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NXT</th>
<th>EV3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{nMotorEncoderTarget[LM] = 2880;} )</td>
<td>( \text{setMotorTarget(LM, 2880, 40);} )</td>
</tr>
<tr>
<td>( \text{nMotorEncoderTarget[RM] = 2880;} )</td>
<td>( \text{setMotorTarget(RM, 2880, 40);} )</td>
</tr>
<tr>
<td>( \text{motor[LM] = motor[RM] = 40;} )</td>
<td></td>
</tr>
</tbody>
</table>

```c
while( \( \text{!}\text{nMotorRunState[LM]} == \text{runStateldile} \) \&\& \( \text{nMotorRunState[RM]} == \text{runStateldile} \) )
    delay(10);
```

```c
while ( \( \text{getMotorRunning(LM)} \) \|\| \( \text{getMotorRunning(RM)} \) )
    delay(10);
```
By now, you should have noticed that there is always a certain amount of overshooting at the complete stop. In order to minimize this, you will need to perform the proper deceleration by adjusting the power based on the remaining distance and a few features about the mechanic aspects of the motor.

This is part of the motor feedback control process.

In mechanical engineering, you will come across a topics called “Proportion, Integral, and Derivative – P.I.D.”. Put in a much simpler form, P. is adjustment based on current data, I. is for adjustment based on past data, and D. is adjustment based on prediction.

Again, RobotC provides this feature for you with intrinsic variable: `nMotorEncoderTarget`.

Note that you do not need to stop the motor. The internal feedback control process will stop it for you at the specific encoder value you specified.

```
e.g.  nMotorEncoderTarget[motorA] = 1000;  // move 1000 encoder counts and stop
      motor[motorA] = 75;                      // start your motor
      // motor[motorA] = 0; is not necessary.
```

**Possible problem:**

- `nMotorEncoderTarget` (for NXT) does not start the motor.
- The controller decreases the power level a lot at the end of the few degrees of rotation. If your battery level is not at its optimum condition, e.g. 7.3V+, the motor may seem to stop before it reaches the target.

If battery level is low...

<table>
<thead>
<tr>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refer to the sample on the right: Motor seems to stop at, say, 998. Thus, if you wait until it finishes to get done, it won’t; and your robot will appear to be stuck.</td>
</tr>
</tbody>
</table>

```
nMotorEncoderTarget[motorA] = 1000;  // move 1000 encoder counts and stop
       while (nMotorEncoder[motorA]< 1000)
            {  motor[motorA] = 50;
            }
       PlaySound(BeepBeep);
       // even the robot stops, it never beeps because the process is stuck at the conditional expression.
```

Workaround to avoid being stuck.

```
nMotorEncoderTarget[motorA] = 1000;
      motor[motorA] = 75;
      if ( nMotorRunState[motorA]==runStateIdle ||
            nMotorRunState[motorA]!= runStateIdle )
            wait1Msec(10);
      PlaySound(BeepBeep);
      // the robot stops and plays beeps.
```
**VI - 3.A) LEARN FROM SAMPLES**

1. 01-GoForward W Target Control.c  
2. 02-GoLeft W Target Control.c  
3. 03-GoForwardWSyncTargetControl.c  
4. 04-GoLeftWSyncTargetControl.c

[Download samples](tutorial samples: robotc.stormingrobots.com).

**VI - 4) MINI-CHALLENGE EXERCISES**

Now, write 2 functions: (II - ANALYTICS)

- goBackWSyncTargetControl.c  
- goRightWSyncTargetControl.c

**Challenge 1:** (II - ANALYTICS)

- goBackWEnc.c : run backward for any specific distance based on the conversion using target control.  
- goRightWEnc.c : make right turn for any specific distance based on the conversion using target control.
SECTION VII – FUNCTIONS

A function is a segment of codes which capsculate a set of actions. Its characteristics:

- Reusable
- Self-contained, if designed properly
- Perform a coherent set of actions for achieving a particular functionality
- May take in variables / arguments which go through the same of instructions
- May return a specific value or none.

__Structure of a function:

A function takes in 2 arguments, and return a value. In this example, it takes in two individual “int” and return another “int”. Note that you can any data types, such char, byte, string, etc.

```c
int largerNum( int x, int y)
{
    if (x > y)
        return x;
    return y;
}
```

A function runs the robot forward until it sees dark and stop.

```c
void goUntilSeeDark()
{
    while(SensorValue[S1]>35)
    { // bot on white
        motor[motorC] = 50;
        motor[motorB] = 50;
    }
    motor[motorC] = 0;
    motor[motorB] = 0;
}
```
For example: to calculate factorial of a number:

```c
int factorial(int number)
{
    int i, result=1;
    for (i=number; i>1 ; i--)
    {
        result = result * i;
    }
    return result;
}

task main()
{
    int answer=0;
    Answer = Factorial (6);
    displayTextLine(2, "6! = %d", answer);
    Answer = Factorial (5);
    displayTextLine(4, "%5! = %d", answer);
}
```
VII - 2) LEARN FROM SAMPLES

00-drawCircle.c
01-FactorialFunc.
02-Simplefuncs W Light Sensor.c
03-GoStraightTargetControl.c
04-GoStraightSyncTargetControl.c
05-GoTurnTargetControl.c
06-GoTurnSyncTargetControl.c
07-sampleHeader.h
07-movementsFuncs.c
07-simpleDriver.c

You may download from this link.

VII - 3) MINI-CHALLENGE EXERCISES

Challenge 1: (II - ANALYTICS)

Modify the 4 sample projects in Ch 6.4 to become 4 separate functions in a single program.

Sample skeleton code:

```c
void goForward(int distInCM )
{
    ....
}

void goBackward(int distInCM )
{
    ....
}

void goLeft(float chassisDegrees)
{
    ....
}

void goRight(float chassisDegrees)
{
    ....
}
```
task main()
{
  ....
}

1. Create a "goStraight(int distance)" function which can perform either forward/backward.

2. Create a "goTurn(int degrees)" function which can perform either right or left turn.

3. Write runSquare.c: program your robot to run in a square using the 2 functions done in (2) and (3) stated above. Note: For accurate navigation, you should use do the gear math conversion, as well as motor feedback control introduced in chapter 6.

4. Instead of running in a square. Program your robot to alternate its turn 4 x.

Hint: Write a flowchart for this first, and you will see you will need only a very few lines of code addition to the previous square program.
SECTION VIII - BUTTONS CONTROL

VIII - 1) LEARN FROM SAMPLES

You can hijack the buttons for specific purpose.

Intrinsic variables:

- nNxtButtonPressed can be one of the following values:
  - kNoButtons = nothing is pressed
  - kLeftButton = left button is pressed
  - kRightButton = right button is pressed
  - kEnterButton = enter button is pressed
  - kExitButton = exit button is pressed

You will find 2 button samples for NXT, and one for EV3.

You may download from this link.

VIII - 2) MINI-CHALLENGE EXERCISES

**Challenge 1:** Change the sound and delay.

a) Modify 09-PlaySoundWhile.c from chapter 1 to play Upward Tone when right button is pressed, but downward Tone when left button is pressed.

b) Modify the 03-Display.c from chapter 1 to display the following using variable like this:

```
I   I*I
-------------
2   4
3   9
4   16
5   25
```

For this exercise, the next number shows only when the right button is pressed. Program ends after the sixth times when a button is pressed.

**Challenge 2:**

Play “Do Re Mi Fa So La Ti ” using the buttons... Right buttons go from Do to Re, from Re to Mi, etc.

Left button goes backward, such as Ti to La, La to So, etc.
SECTION IX - SOME DIFFERENCES IN EV3

This chapter covers some commonly used APIs which are different in EV3 from NXT.

Some minor changes in data types changed, such as from "word" to "short".

You should always consult either the RobotC Helper and/or the “right mouse click” on the API itself to make sure the data types matches.

Since these are just small changes, I do not re-run all samples for version 4.X. You should have enough skill set to make these slight changes yourself.

—Samples to show the main changes on the topics covered in this packet.

<table>
<thead>
<tr>
<th>NXT</th>
<th>EV3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic (out of the box) Sensors Types:</td>
<td></td>
</tr>
<tr>
<td>Light: sensorLightActive</td>
<td>sensorEV3_Color</td>
</tr>
<tr>
<td>Sonar: sensorSONAR</td>
<td>sensorEV3_Ultrasonic</td>
</tr>
<tr>
<td>Touch: sensorTouch</td>
<td>sensorEV3_Touch</td>
</tr>
<tr>
<td>Color: sensorColorNxtFULL</td>
<td>sensorEV3_Color</td>
</tr>
<tr>
<td>Display</td>
<td></td>
</tr>
<tr>
<td>nxtDisplayTextLine( . )</td>
<td>displayTextLine( . )</td>
</tr>
<tr>
<td>ntxtXXX...</td>
<td>XXX...</td>
</tr>
<tr>
<td>Timing</td>
<td></td>
</tr>
<tr>
<td>wait... ( … )</td>
<td>delay( milliseconds )</td>
</tr>
<tr>
<td>Motor Feedback Control:</td>
<td></td>
</tr>
<tr>
<td>nMotorEncoderTarget[motorC] = 2880; motor[motorC] = 40;</td>
<td>setMotorTarget(motorA, 2880, 40);</td>
</tr>
<tr>
<td>while( !(nMotorRunState[motorA] == runStateIdle &amp;&amp; nMotorRunState[motorC] == runStateIdle) ) delay(10);</td>
<td>while ( getMotorRunning(motorA)</td>
</tr>
</tbody>
</table>

Again, be resourceful! You may use the “Help”, or the samples from RobotC, or samples from [http://learn.stormingrobots.com](http://learn.stormingrobots.com). Or better yet, if you love to look further into the gut of RobotC, look at the "RobotCIntrinsics.c".
SECTION X ERRATA

Please email to office@stormingrobots.com

☞ The End of Packet I ☜