Lab Objectives

By the end of this lab you should be able to:

1. **Identify** the key components of the Scribbler 2 robot.
2. **Use** the Propeller Tool IDE to create and download programs to the robot.
3. **Identify** what a variable is and how one is used in a program.
4. **Create** a program that utilizes *repeat-loops* and *if/else* statements to direct the behavior of the robot.

The Scribbler 2 Robot

The Scribbler 2 (shown below) is a robot that is mobile (meaning it can move), autonomous (meaning that it can make its own decisions thanks to its embedded microcontroller and its various sensors, which allow it to perceive its environment) and programmable (meaning you can create and modify the program rendering it autonomous).

The main feature of version 2 of the Scribbler is its microcontroller, the Propeller. This is a multicore Parallax microcontroller, meaning that it can effectively run tasks in parallel, which is essential for a robot.

Programming the Scribbler 2

The Scribbler 2 can be programmed using SPIN language, a language created by Parallax specifically for the Propeller microcontroller. The Propeller Tool software is the integrated development environment (IDE) used for writing, compiling and downloading programs written in SPIN to the Scribbler 2 robot. This software is already installed on all of the lab computers.
and is free to download if you want a copy for your personal computer. A Windows version can be downloaded from: [http://www.parallax.com/downloads/propeller-tool-software](http://www.parallax.com/downloads/propeller-tool-software) and a version for Mac/Linux can be downloaded from: [http://www.fnarf Bargle.com/bst.html](http://www.fnarf Bargle.com/bst.html)

SPIN programs consist of six special declaration blocks: CON, VAR, OBJ, PUB, PRI and DAT. To help make the code more readable, the Propeller Tool represents each declaration block with a different color (example code shown below).

For this lab, we will only be using the first four declaration blocks. The CON block is the section of code that declares global constant symbols and global Propeller configurations settings. The VAR block is the section of code that declares global variable symbols and will be discussed in the next section. The OBJ block is the section of code that declares which objects are used and the object symbols that represent them. Every program for the Scribbler 2 must contain the object declaration:

```plaintext
s2 : "S2"
```

which gives us access to the robot and the functions necessary to control it. The PUB block is the section of code that performs a specific function. This is the section that contains the main part of the program.

A template containing all of the necessary declarations in each block can be found on eCommons under “Resources” on the left-hand side of the home page. This template can and should be used as a starting point for all Scribbler 2 programs.

The Scribbler 2 connects to a computer through the serial port on the front of the robot. We will be using a serial to USB adaptor to connect the Scribblers to the lab computers. In order to
download your program to the Scribbler 2, it must have its power switch in the “on” position and be connected to a computer.

Variables

In SPIN programming language, variables are declared in the VAR block with a size and a name. The size of a variable must be declared so the compiler can tell the microcontroller how much space to set aside for the variable in memory. Variables can be assigned three different sizes which are:

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits (unsigned)</td>
<td>0 to 255</td>
</tr>
<tr>
<td>word</td>
<td>16 bits (unsigned)</td>
<td>0 to 65,535</td>
</tr>
<tr>
<td>long</td>
<td>32 bits (signed)</td>
<td>-2,147,483,648 to +2,147,483,647</td>
</tr>
</tbody>
</table>

When a variable is declared in the VAR block it is not assigned a value. Variables are assigned values inside the PUB block. For example, if we declared a variable x in the VAR block using the command:

```sp
byte x
```

then we can assign it a value in the PUB block using the command:

```sp
x := 10
```

Notice that the operator “:=” is used to assign a value to a variable in SPIN where as the “=” operator is used to assign a value to a constant in the CON block.

Loops

The SPIN language uses the command repeat to create loops. There are four different ways to use the repeat command to create loops that are equivalent to for-loops and while-loops in MATLAB. To create the equivalent of a for-loop, we use the repeat command followed by the number of iterations we want the loop to perform. The following two examples show the different methods for creating this type of loop.

Example 1:

```sp
x := 0
repeat 10
  x := x + 1
```

Example 2:

```sp
x := 0
repeat n from 0 to 9
  x := x + 1
```

The loops in both examples perform ten iterations and result in x being equal to ten. It is assumed that the variables x and n were previously declared in the VAR block. Notice that we do not use an end command to terminate the loop. Instead every line of code that belongs in the loop must be indented. If a line of code comes after the repeat command and it is not indented, then it will be executed after the loop is finished.
To create the equivalent of a while-loop, we use the repeat command followed by the while command or the until command and a conditional statement. The following two examples demonstrates the syntax for creating this type of loop.

Example 3:

```
x := 0
repeat while x < 10
    x := x + 1
```

Example 4:

```
x := 0
repeat until x == 10
    x := x + 1
```

The loops in these examples are equivalent to each other and perform ten iterations and result in x being equal to ten. It is again assumed that the variable x was previously declared in the VAR block.

**if/else Statement**

The if/else statement in SPIN has a similar form to the if/else statement in MATLAB. We use the if command with a conditional statement optionally followed by multiple elseif commands with conditional statements and an optional else command without any conditions. The following example shows the proper syntax for an if/else statement in SPIN.

Example 5:

```
x := 1
if x == 1
    s2.set_leds(s2#GREEN, s2#OFF, s2#OFF, s2#NO_CHANGE)
elseif x == 2
    s2.set_leds(s2#OFF, s2#GREEN, s2#OFF, s2#NO_CHANGE)
elseif x == 3
    s2.set_leds(s2#OFF, s2#OFF, s2#GREEN, s2#NO_CHANGE)
else
    s2.set_leds(s2#OFF, s2#OFF, s2#OFF, s2#NO_CHANGE)
```

This example uses the command s2.set_leds() to turn the Scribbler 2's LEDs on and off based on the value of x. The command s2.set_leds() takes as an argument the desired state of the robot's LEDs. The LEDs are called from left to right in the order they are shown in the top view of the robot (see page 1 of this lab).

The result of this if/else statement is the left LED is turned on, the center and right LEDs are turned off, and the power LED doesn't change. This example assumes that the object s2 was previously declared in the OBJ block and the variable x was previously declared in the VAR block.
block. Notice that we use indentation again the same way we did with the repeat command to indicate which lines of code belong to each part of the if/else statement.

Relational Operators

The if/else statement and some repeat-loops in SPIN use relational operators to test the truthfulness of conditional statements. Some of the relational operators are the same as the ones used in MATLAB but some are different. The relational operators in SPIN are:

<table>
<thead>
<tr>
<th>Relational Operator</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not Equal</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater Than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less Than</td>
</tr>
<tr>
<td>=&gt;</td>
<td>Equal To or Greater Than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Equal To or Less Than</td>
</tr>
</tbody>
</table>

Modulus operator

The modulus operator (also referred to as the modulo operator) returns the remainder of division of one number by another. In SPIN, the modulus operator is “//” and it only works with integer values. The following examples demonstrate how the modulus operator works.

Example 6:  
\[
    x := 9 // 2
\]

In Example 6, x is assigned the value 1 since 9 divided by 2 is equal to 4 with a remainder of 1.

Example 7:  
\[
    x := 10 // 2
\]

In Example 7, x is assigned the value 0 since 10 divided by 2 is equal to 5 with a remainder of 0. These two examples illustrate how the modulus operator can be used to distinguish between even and odd numbers. When an odd number is divided by 2, the remainder is 1, whereas when an even number is divided by 2, the remainder is 0.

Propeller Manual, s2 Object and Serial Terminal

The Propeller manual contains everything you need to know about all of the commands in the SPIN programming language. The SPIN code for the s2 object contains descriptions of all of the commands that can be used to control the Scribbler 2 robots. The SPIN code for the FullDuplexSerial object contains descriptions of all of the commands that can be used to interact with the serial terminal. In order to use these objects, both the s2 and FullDuplexSerial codes must be placed in your working directory. The manual and both object SPIN codes should be the first place you look if you need help with programming the Scribbler 2 robots. These three resources have been uploaded to the class website on eCommons and can be found under Resources on the left-hand side of the home page.
Exercises

1. Using the repeat command, calculate the sum of all the even numbers between 1 and 21. Print the final value of the sum to the serial terminal. **Hint:** use the modulus operator to pick out the even numbers.

2. Using the repeat while command, turn on the Scribbler’s LEDs from right to left one at a time, and repeat this action five times. **Hint:** use the s2.delay_tenths() command to keep an LED on for a short amount of time before turning it off and turning on the next one.

3. Start by setting a variable x equal to zero. Using the repeat command, increment x by one ten times. Inside the repeat-loop, use an if/else statement to turn on the center LED and turn off the left and right LEDs if x is an even number, and turn off the center LED and turn on the left and right LEDs if x is an odd number. **Hint:** use the modulus operator again to tell the even numbers from the odd numbers.

**Note:** For this lab you should comment every line of your code. It shows that you know what you are doing and makes it easier for someone else (i.e., the grader) to understand your code.