WEEK 2
Experiment # 6

Remembering Long Lists Using EEPROM

EEPROM stands for Electrically Erasable Programmable Read Only Memory. It is a small black chip on the BASIC Stamp II module labeled “24LC16B”. It is used to store the program and data temporarily. When you don’t need the program / data inside the EEPROM, you can replace it with another one. EEPROM can accept a finite number of write cycles, around 10 million writes. The EEPROM chip on BASIC Stamp II (24LC16B) can hold up to 2048 bytes (2 kB) of information.

Acitivity:

Type in the following codes onto Stamp Editor.

```
‘ Program Listing 2.5, EEPROM Navigation.
‘ {$Stamp bs2}                      ‘ Stamp Directive

‘ ------Declarations -----
‘ Label for the declarations routine
pulse_count var word                ‘ Declare a var. named pulse_count
EE_address var byte                 ‘ Stores & increments EEPROM addr.
instruction var byte                ‘ Stores instruction from EEPROM

data     “FFFBBLFFRFFQ"             ‘ List of Boe-Bot nav. instruction

‘ ----- Initialization -----
output 2                            ‘ Set P2 as output pin
freqout 2, 2000, 3000           ‘ Send the prog start/low battery tone
low 12                              ‘ Set P12 and P13 to output-low
low 13

‘ ----- Main routine -----
main:                               ‘ Main routine label
   read EE_address, instruction ‘ Read at EE_addr, store in instruction
   EE_address = EE_address + 1      ‘ Increment EE_addr for next read

   if instruction = “F” then forward    ‘ Check for forward command
   if instruction = “B” then backward   ‘ Check for backward command
   if instruction = “R” then right_turn ‘ Check for right turn command
   if instruction = “L” then left_turn  ‘ Check for left turn command

   stop                            ‘ Stop executing commands until reset

‘ ---- Navigation Routine ----

forward:                            ‘ Forward routine
   for pulse_count = 1 to 75        ‘ Sends 75 forward pulses
        pulsout 12, 500             ‘ 1.0 ms pulse to right servo
        pulsout 13,1000             ‘ 2.0 ms pulse to left servo
        pause 20                    ‘ Pause for 20 ms
   next
```

```
goto main                           ‘ Send program back to the main

backward:                           ‘ Backward routine
   for pulse_count = 1 to 75        ‘ Sends 75 backward pulses
        pulsout 12, 1000            ‘ 2.0 ms pulse to right servo
        pulsout 13, 500             ‘ 1.0 ms pulse to left servo
        pause 20                    ‘ Pause for 20 ms
   next
goto main                           ‘ Send program back to the main

left_turn:                          ‘ Left turn routine
   for pulse_count = 1 to 35        ‘ Sends 35 left rotate pulses
        pulsout 12, 500             ‘ 1.0 ms pulse to right servo
        pulsout 13, 500             ‘ 1.0 ms pulse to left servo
        pause 20                    ‘ Pause for 20 ms
   next
goto main                           ‘ Send program back to the main

right_turn:                         ‘ Right turn routine
   for pulse_count = 1 to 35        ‘ Sends 35 right rotate pulses
        pulsout 12, 1000            ‘ 2.0 ms pulse to right servo
        pulsout 13, 1000            ‘ 2.0 ms pulse to left servo
        pause 20                    ‘ Pause for 20 ms
   next
goto main                           ‘ Send program back to the main
```

Program Explanation:

Both the "*EE_address*" and "*instruction*" variables are in byte, which means that they can store numbers between 0 and 255. The "*EE_address*" variable is used for specifying the EEPROM address to read a direction instruction from EEPROM. The "*instruction*" variable is used to store the instruction character read from EEPROM. The next declaration ("*data*") is the actual data to be stored in EEPROM. This data is stored as a string of characters.

The *main* routine first reads EEPROM address 0, and stores it in the *instruction* variable. Then, *EE_address* is incremented so that the next read cycle will look at address 1. A series of *if … then* statements is used to decide what to do based on the character retrieved from EEPROM and stored in the *instruction* variable. The *if … then* statements check to see if it is one of four known instruction characters: "F", "B", "R", and "L". For example, if the character is an "R", the first two *if … then* statements are skipped because neither of them is true. Since the third *if … then* statement is true, the program skips to the *right_turn* routine and executes it.

When the program gets into *goto main* command, it will execute the *main* routine, then the next EEPROM instruction is fetched and the instruction is checked by the four *if … then* statements again. The process is repeated until the quit character "Q" is read from EEPROM. When "Q" is loaded into the *instruction* variable, it fails all four *if … then* tests. So, the program does not go to any of the navigation routines. Instead, the program goes to the command that follows the series of *if … then* statements, which is the *stop* command.

Save the program as "Prog2_5.bs2" and run it.

Your Boe-Bot should remain still and plays the tone for 2 seconds. Then it is moving with respect to the navigation direction data, which is moving forward, backward, turn a quarter turn to the left, moving forward, turn a quarter turn to the right, and then moving forward again and finally stops.

Task:

1. Try changing, adding, and deleting characters in the *data* directive. Remember that the last character in the *data* directive should always be a "Q".
2. Try adding a second *data* directive. Remember to remove the "Q" from the end of the first *data* directive and add it to the end of the second. Otherwise, the program will execute only the commands in the first *data* directive.

Simplify Navigation with Subroutines

A subroutine is a segment of code that does a particular job. To make the subroutine does its job, a command is used in the main routine that "calls" the subroutine. The command for calling a subroutine is the *gosub* command, and it is similar to the *goto* command. A *goto* command tells the program to go to a label and then start executing instructions. The *gosub* command tells the program to go to a label and start executing instructions, but come back when finished. A subroutine is finished when the *return* command is encountered.

Activity:

Type the following codes onto the Stamp Editor.

```
‘ Program Listing 2.6, Subroutine Navigation.
‘ {$Stamp bs2}                     ‘ Stamp Directive

‘ ------Declarations ----
loop_count var word                ‘ For … next loop counter
right_width var word               ‘ Variable stores right pulse width
left_width var word                ‘ Variable stores left pulse width
pulse_count var word               ‘ Used to set # of pulses delivered

‘ ----- Initialization -----
output 2                            ‘ Set P2 as output pin
freqout 2, 2000, 3000               ‘ Signal program is (re)starting
low 12                              ‘ Set P12 and P13 to output-low
low 13

‘ ---- Main routine -----
main:
      forward:                      ‘ Forward routine
            pulse_count = 75        ‘ Set pulse_count for 75 pulses
            right_width = 500       ‘ Set right pulse width to 1.0 ms
            left_width = 1000       ‘ Set left pulse width to 2.0 ms
            gosub pulses            ‘ Call the pulses subroutine
            pause 500               ‘ Pause for 0.5 s

      backward:                     ‘ Backward routine
            pulse_count = 75        ‘ Set pulse_count for 75 pulses
            right_width = 1000      ‘ Set right pulse width to 2.0 ms
            left_width = 500        ‘ Set left pulse width to 1.0 ms
            gosub pulses            ‘ Call the pulses subroutine
            pause 500               ‘ Pause for 0.5 s

      stop                      ‘ Stop executing commands until reset


‘ ---- Subroutines ---
pulses:                             ‘ Pulses subroutine
   for loop_count=1 to pulse_count ‘ Use pulse_count for # of pulses
```

```
        pulsout 12, right_width  ‘ Use right_width for right pulse width
        pulsout 13, left_width    ‘ Use left_width for left pulse width
        pause 20                     ‘ Pause 20 ms
      next
return                            ‘ Return from subroutine
```

Program Explanation:

In the *forward* and *backward* routine, when it gets into *gosub pulses* command, it will jump to the *pulses* subroutine. The commands in the *pulses* subroutine are executed until the program gets to the *return* command. The *return* command sends the program back to the command just after *gosub pulses* command, whish is *pause 500* in this case.

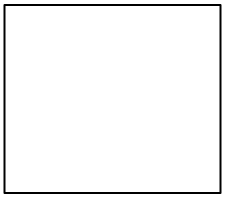
Save this program as “Prog2_6.bs2” and run it.

Your Boe-Bot should remain still and plays the tone for 2 seconds. Then it starts moving forward. Stops for about 0.5 second and then it should move backward and stops on its initial position.

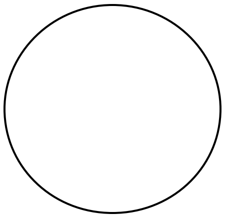
Task:

1. Add routines to the main routine that set the values for a quarter of right turn and left turn. Save this program with the same name (“Prog2_6.bs2”) and run it.
2. Create a source code for one of the following movement patterns:

40 cm/side



20 cm radius



40 cm equilateral