Activity 3.1.5 Variables and Functions – VEX

Introduction

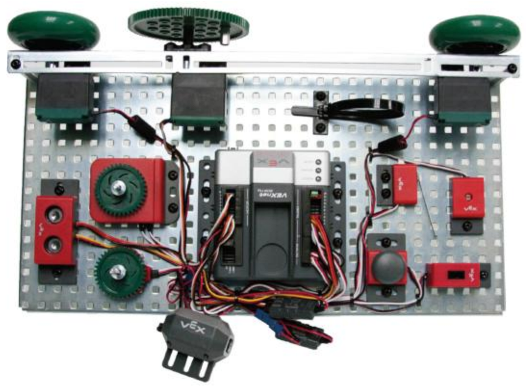
A program can accomplish a given task in any number of ways. Programs can quickly grow to an unmanageable size so variables and functions provide a technique to reduce the size of the program. A variable is a space in your robot's memory where you can store data, such as whole numbers, decimal numbers, and words. Functions group together several lines of code, which can then be referenced many times in task main, or even other functions. The use of variables and functions allows for complex control of a system.

Equipment

- Computer with ROBOTC software
- POE VEX® testbed
- PLTW ROBOTC template

Procedure

1. Form groups of four and acquire your group's POE VEX Kit under your teacher's direction.
2. Within your four-student group, form a two-student team known as Team A and a two-student team known as Team B.
    a. Team A will construct the VEX Testbed with the exception of the ultrasonic and the light sensor.
    b. Team B will construct the VEX Testbed with the exception of the servo motor and flashlight.
3. Connect the POE VEX testbed Cortex to the PC.

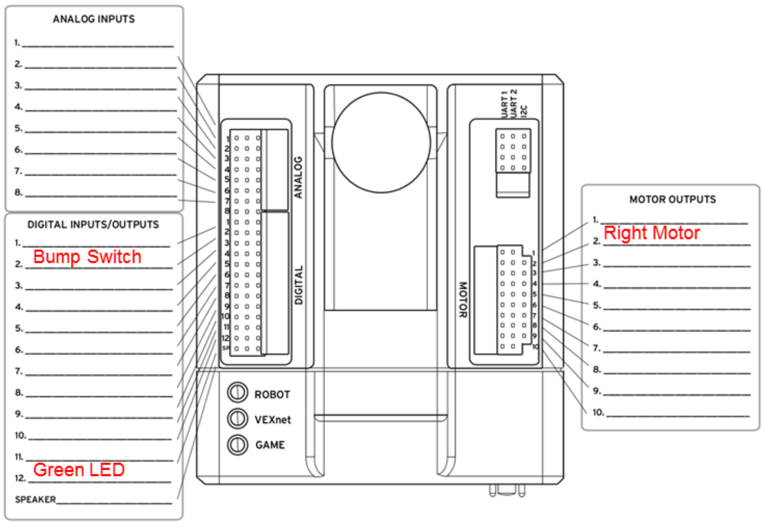


POE VEX Testbed

Part 1: Using a variable to loop n times

4. Open the PLTW ROBOTC template. Click File, Save As, select the folder that your teacher designated for you to save your ROBOTC programs in, then name the file A3_1_5_Part1.
5. In this activity you will use the green LED, right motor, and bumper switch. Leave the previously connected motors and sensors wired to the Cortex. Go to the Motors and Sensors Setup window. Configure the Motors and Sensors Setup to reflect the inputs and outputs to be used. Note that additional motors and sensors that are physically attached may be configured; however, these are not required to be configured. Click OK to close the window.

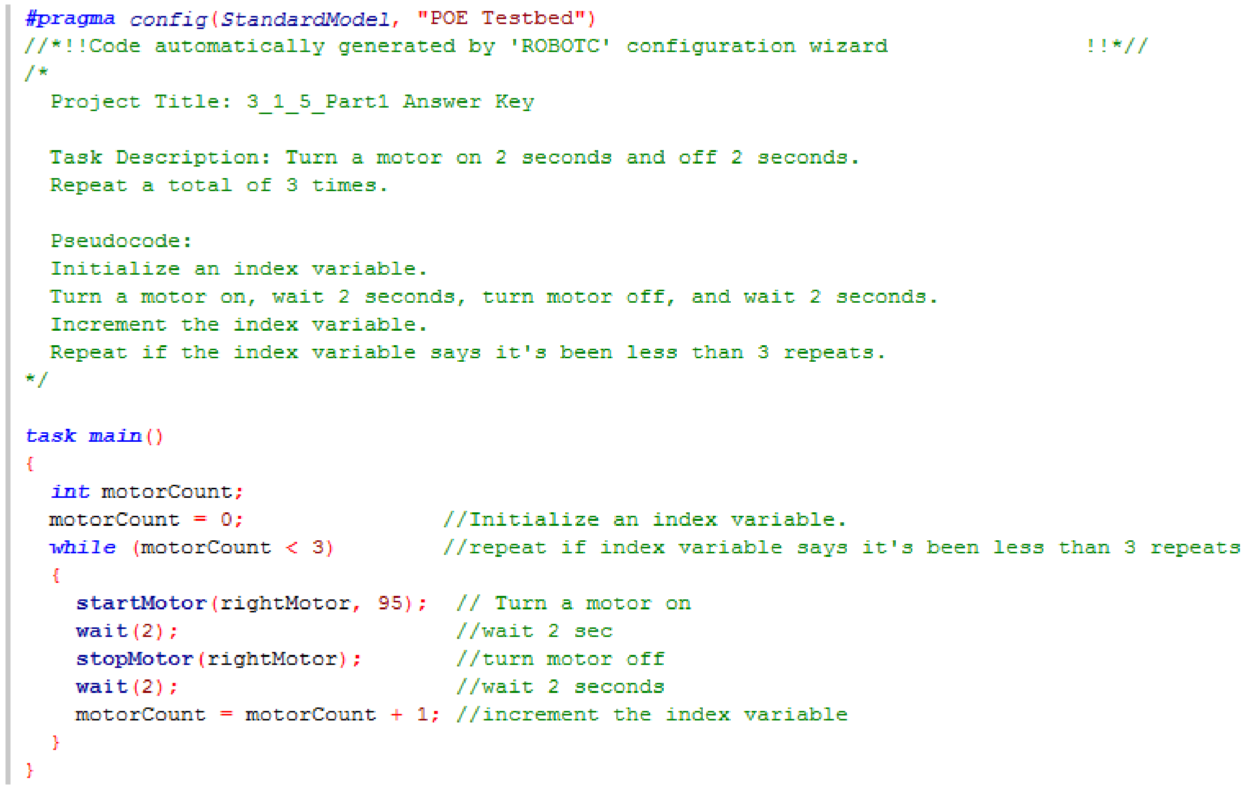
Cortex Wiring Diagram



6. Copy and paste or create the program below in the `task main()` section of the program between the curly braces.

```
int motorCount;
motorCount = 0;
while (motorCount < 3)
{
  startMotor(rightMotor, 95);
  wait(2);
  stopMotor(rightMotor);
  wait(2);
  motorCount = motorCount + 1;
}
```

7. Save the program, power on the Cortex, compile, and download the program. If you have any errors, check with your instructor to troubleshoot your program.
8. Press Start to run the program and observe the behaviors.
9. Document what this program would look like as pseudocode simple behaviors.



```
#pragma config(StandardModel, "POE Testbed")
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//
/*
  Project Title: 3_1_5_Part1 Answer Key

  Task Description: Turn a motor on 2 seconds and off 2 seconds.
  Repeat a total of 3 times.

  Pseudocode:
  Initialize an index variable.
  Turn a motor on, wait 2 seconds, turn motor off, and wait 2 seconds.
  Increment the index variable.
  Repeat if the index variable says it's been less than 3 repeats.
*/

task main()
{
  int motorCount;
  motorCount = 0;               //Initialize an index variable.
  while (motorCount < 3)        //repeat if index variable says it's been less than 3 repeats
  {
    startMotor(rightMotor, 95);  // Turn a motor on
    wait(2);                     //wait 2 sec
    stopMotor(rightMotor);       //turn motor off
    wait(2);                     //wait 2 seconds
    motorCount = motorCount + 1; //increment the index variable
  }
}
```

10. Create a program that will run a motor back and forth 20 times, 0.5 seconds each way. Test the program and troubleshoot until the expected behavior has occurred. Make sure your code is documented with a task description, pseudocode, and line-by-line comments. Save the program. You will reuse this program in Activity 3.1.6.

```
#pragma config(StandardModel, "POE Testbed")
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//
/*
  Project Title: 3_1_5_Part1 Answer Key

  Task Description: Turn a motor on 1/2 seconds and backwards 1/2 seconds.
  Repeat a total of 20 times.

  Pseudocode:
  Initialize an index variable.
  Turn a motor on, wait 1/2 seconds, turn motor on backwards, and wait 1/2 seconds.
  Increment the index variable.
  Repeat if the index variable says it's been less than 20 repeats.
*/

task main()
{
  int motorCount;
  motorCount = 0;                //Initialize an index variable.
  while (motorCount < 20)         //repeat if index variable says it's been less than 20 repeats
  {
    startMotor(rightMotor,  127);  // Turn a motor on
    wait(0.5);                     //wait 1/2 sec
    startMotor(rightMotor,-127);   //turn motor on backwards
    wait(0.5);                     //wait 1/2 seconds
    motorCount = motorCount + 1; //increment the index variable
  }
}
```

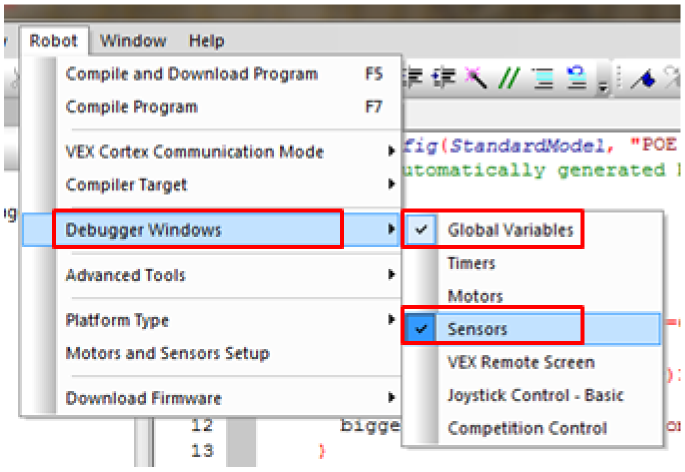
Part 2: Using a variable to remember a maximum or minimum value and Using debugger windows

11. Open the PLTW ROBOTC template. Click File, Save As, select the folder that your teacher designated, and then name the file A3_1_5_Part2.
12. Copy and paste or create the program below in the `task main()` section of the program between the curly braces.

```
int biggest;
while (1==1)
{
    biggest = 0;
    while (SensorValue(bumpSwitch)==0)
    {
        if (SensorValue(potentiometer)>biggest)
        {
            biggest=SensorValue(potentiometer);
        }
    }
}
```

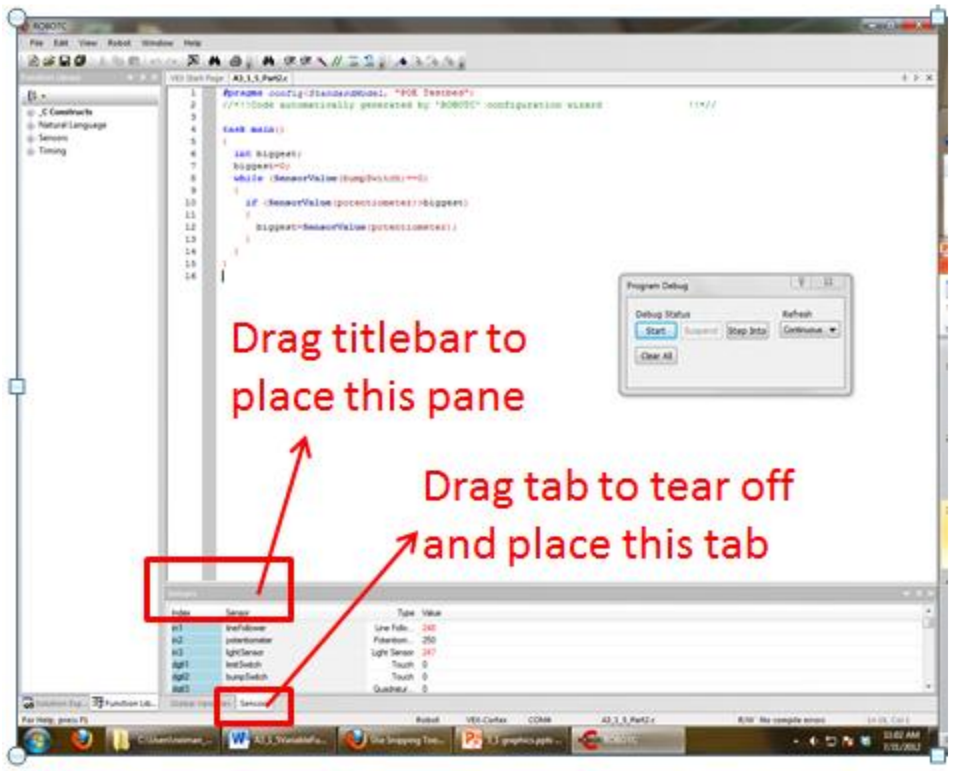
13. Compile and download the program to the Cortex.

14. Use the Robot > Debugger Windows menu to make sure that both the “Global Variables” and “Sensors” tabs are checked. See the figure for how the menu should appear.



15. These debugger windows can be placed wherever they are convenient for you on the ROBOTC screen. To read about this, choose Help > Open Help and then in the Contents tab, expand “ROBOTC Debugger” by clicking on it’s plus sign, and then click on “Docking Debugger Windows.” Steps 16-18 repeat the information from the built-in help. If you are able to learn from the built in help, just make your debugger windows look like the picture in step 17, and then continue with step 19.

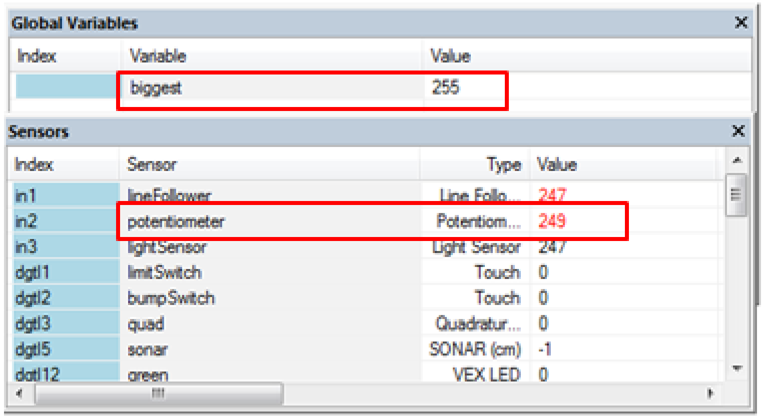
16. To move the debugger windows, click on the titlebar of the debugger window and drag it to the desired location. If the “Global Variables” and “Sensors” debugger windows are sharing a single pane with separate tabs, you can tear one of the tabs off, to make it into its own window, by clicking on the tab and dragging it to a new location. Make two separate windows, one for the “Sensors” debugger window and one for the “Global variables” window.



```
#pragma config(StandardModel, "POE Testbed")
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

task main()
{
  int biggest;
  biggest=0;
  while (SensorValue(bumpSwitch)==0)
  {
    if (SensorValue(potentiometer)>biggest)
    {
      biggest=SensorValue(potentiometer);
    }
  }
}
```

Drag titlebar to place this pane
Drag tab to tear off and place this tab
Program Debug
Start
Clear All

17. Place the "Global Variables" and "Sensors debugger windows on top of one another so that you can easily see the value of the variable "biggest" and the value of the sensor "potentiometer" at the same time. The windows might look like this:



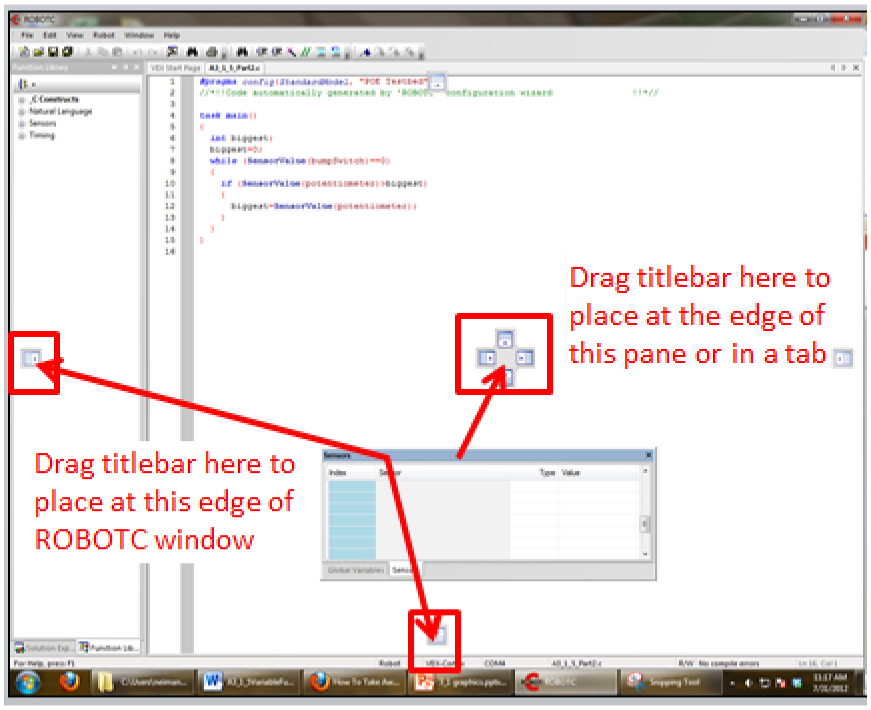
Global Variables

Index	Variable	Value
	biggest	255

Sensors

Index	Sensor	Type	Value
in1	lineFollower	Line Follo...	247
in2	potentiometer	Potentiom...	249
in3	lightSensor	Light Sensor	247
dgtl1	limitSwitch	Touch	0
dgtl2	bumpSwitch	Touch	0
dgtl3	quad	Quadratur...	0
dgtl5	sonar	SONAR (cm)	-1
dgtl12	green	VEX LED	0

18. Later, you can make the debugger windows "dock" again at the bottom of the code editing pane. Drag the debugger window's titlebar and release the mouse on one of the icons shown in the figure below. The debugger window will dock at one edge of the ROBOTC window or of the selected ROBOTC pane. A debugger window can also be placed back into a tab within another debugger window this way.


Drag titlebar here to place at the edge of this pane or in a tab
Drag titlebar here to place at this edge of ROBOTC window

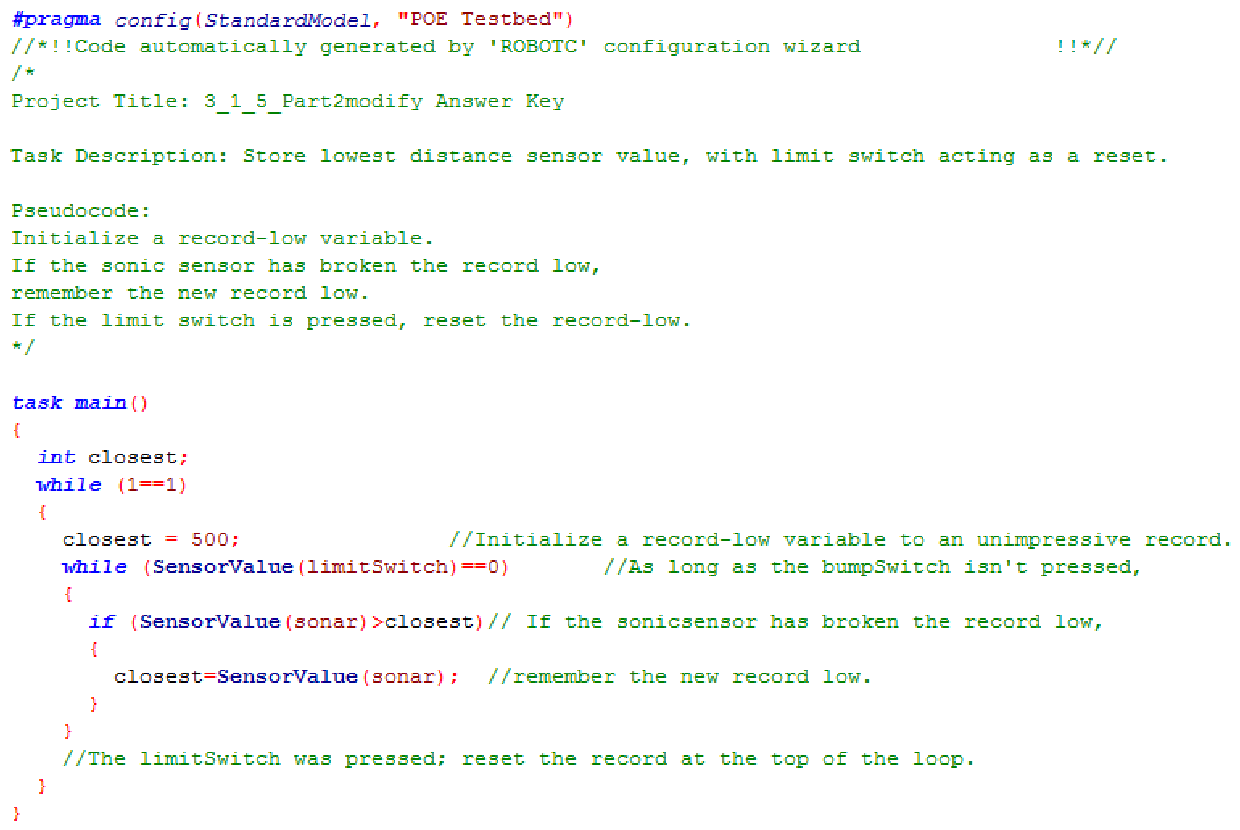
```
#pragma config(StandardModel, "POE Testbed")
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

task main()
{
  int biggest;
  biggest=0;
  while (SensorValue(bumpSwitch)==0)
  {
    if (SensorValue(potentiometer)>biggest)
    {
      biggest=SensorValue(potentiometer);
    }
  }
}
```

19. Start the program and slowly turn the potentiometer back and forth, using the debugger windows to monitor the value of the variable "biggest" and the value of the "potentiometer". Press the bump switch and note the effect. Continue to turn the potentiometer, press the bump switch, and monitor the variable and sensor values until you think you understand how the code works. What does this program do? Annotate the program with a task description, pseudocode, and line-by-line comments.

```
#pragma config(StandardModel, "POE Testbed")
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//
/*
Project Title: 3_1_5_Part2 Answer Key

Task Description: Store highest potentiometer value, with bump switch acting as a reset.

Pseudocode:
Initialize a record-high variable.
If the potentiometer has broken the record high,
remember the new record high.
If the bump switch is pressed, reset the record-high.
*/

task main()
{
  int biggest;
  while (1==1)
  {
    biggest = 0;                          //Initialize a record-high variable.
    while (SensorValue(bumpSwitch)==0)    //As long as the bumpSwitch isn't pressed,
    {
      if (SensorValue(potentiometer)>biggest)// If the potentiometer has broken the record high,
      {
        biggest=SensorValue(potentiometer);  //remember the new record high.
      }
    }
    //The bumpSwitch was pressed; reset the record at the toip of the loop.
  }
}
```

20. Click File, Save As, select the folder that your teacher designated, and then name the file A3_1_5_Part2modify.

a. If you have the ultrasonic distance sensor, modify the program so that a variable “closest” will remember the closest distance detected by the ultrasonic distance sensor. The limit switch should reset the record.



```
#pragma config(StandardModel, "POE Testbed")
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//
/*
Project Title: 3_1_5_Part2modify Answer Key

Task Description: Store lowest distance sensor value, with limit switch acting as a reset.

Pseudocode:
Initialize a record-low variable.
If the sonic sensor has broken the record low,
remember the new record low.
If the limit switch is pressed, reset the record-low.
*/

task main()
{
  int closest;
  while (1==1)
  {
    closest = 500;                //Initialize a record-low variable to an unimpressive record.
    while (SensorValue(limitSwitch)==0)       //As long as the bumpSwitch isn't pressed,
    {
      if (SensorValue(sonar)>closest)// If the sonicsensor has broken the record low,
      {
        closest=SensorValue(sonar);  //remember the new record low.
      }
    }
    //The limitSwitch was pressed; reset the record at the top of the loop.
  }
}
```

OR

b. If you have the light sensor, modify the program so that a variable “brightest” will remember the brightest light detected by the light sensor. The limit switch should reset the record.

```
#pragma config(StandardModel, "POE Testbed")
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//
/*
Project Title: 3_1_5_Part2modifyB Answer Key

Task Description: Store brightest sensor value, with limit switch acting as a reset.

Pseudocode:
Initialize a record-low variable (low=bright).
If the light sensor has broken the record low,
remember the new record low.
If the limit switch is pressed, reset the record-low.
*/

task main()
{
  int brightest;
  while (1==1)
  {
    brightest = 500;                //Initialize a record-low variable to an unimpressive record.
    while (SensorValue(limitSwitch)==0)       //As long as the limitSwitch isn't pressed,
    {
      if (SensorValue(lightSensor)>brightest)// If the light sensor has broken the record low,
      {
        brightest=SensorValue(lightSensor);  //remember the new record low.
      }
    }
    //The limitSwitch was pressed; reset the record at the top of the loop.
  }
}
```

21. Test your program and troubleshoot until the expected behavior has occurred. Make sure your code is documented with a task description, pseudocode, and line-by-line comments. Save the program.

Part 3: Defining your own function

22. Open the PLTW ROBOTC template. Click File, Save As, select the folder that your teacher designated, and then name the file A3_1_5_Part3.

23. Copy and paste or create the program below in the `task main()` section of the program between the curly braces.

```
void LEDControl();        //this is a function declaration
task main()
{
   while (1==1)
   {
      LEDControl();   //function call
   }
}
void LEDControl()   //function definition
{
   if (SensorValue(bumpSwitch)==1)
   {
```

```
            turnLEDOn(green);
        }
        else
        {
            turnLEDOff(green);
        }
    }
```

24. Test the program and troubleshoot until the expected behavior has occurred. Save the program.
25. Describe the behaviors observed.

26. Follow teacher direction and either print the programs or submit electronically with this activity.

Conclusion

1. Describe any challenges that you encountered while developing the program.

2. Describe applications for variables and functions.