PROJECT LEAD THE WAY

PLTW

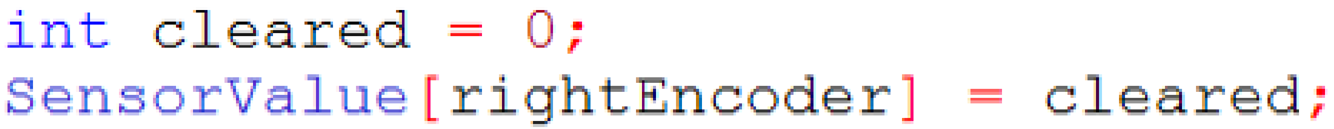
Igniting imagination and innovation through learning.

Variables and Functions

ROBOTC Software

Variables

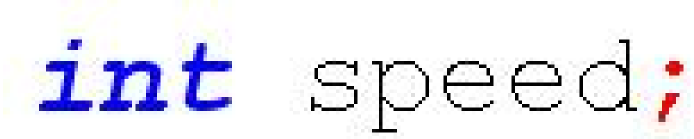
- A variable is a space in your robots memory where data can be stored, including whole numbers, decimal numbers, and words
- Variable names follow the same rules as custom motor and sensor names: capitalization, spelling, availability
- Variables can improve the **readability** and **expandability** of your programs



```
int cleared = 0;
SensorValue[rightEncoder] = cleared;
```

Creating a Variable

- Declare the variable (stating its type and its name) once at the beginning of task main:



```
int speed;
```

Type of data:

- int
- float

Name of variable:

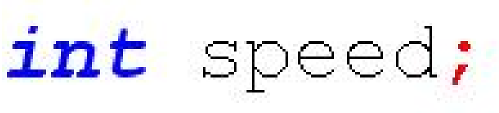
- Starts with letter
- Letters, numbers, and underscores are ok
- Not a reserved word

Variable Types

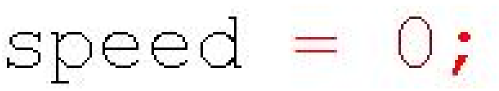
Data Type	Description	Example	Code
Integer	Positive and negative whole numbers, as well as zero	-35, -1, 0, 33, 100	**int**
Floating Point Number	Numeric values with decimal points (even if the decimal part is zero)	-.123, 0.56, 3.0, 1000.07	**float**
Boolean	True or false – Useful for expressing the outcomes of comparisons	true, false	**bool**
Character	Individual characters, placed in single quotes. Not useful with POE kits.	‘L’, ‘f’, ‘8’	**char**
String	Strings of characters, such as words and sentences placed in double quotes. Not useful with POE kits.	“Hello World!”, “asdf”	**string**

Creating a Variable

- Initialize the variable by giving it its initial value:

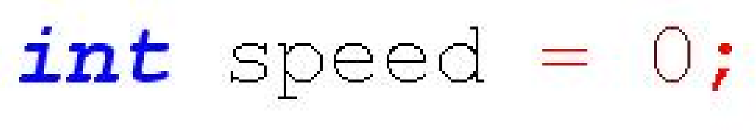


```
int speed;
```



```
speed = 0;
```

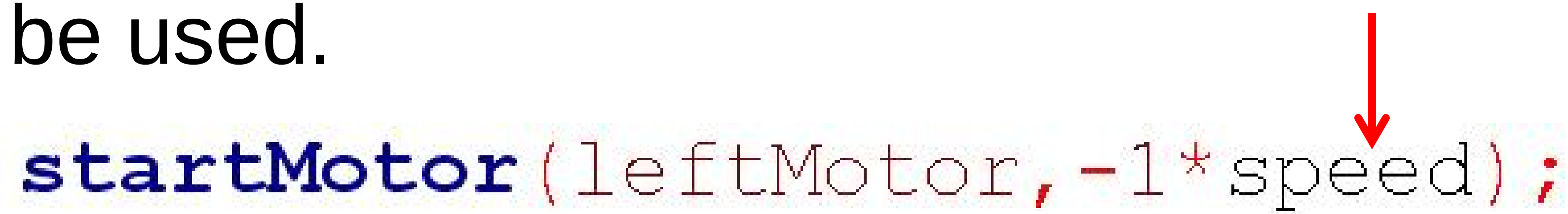
- Declaration and initialization are often contracted into a single statement:



```
int speed = 0;
```

Using the Value of a Variable

- The value stored in the variable can be referenced by using the variable’s name anywhere the value of the variable could be used.

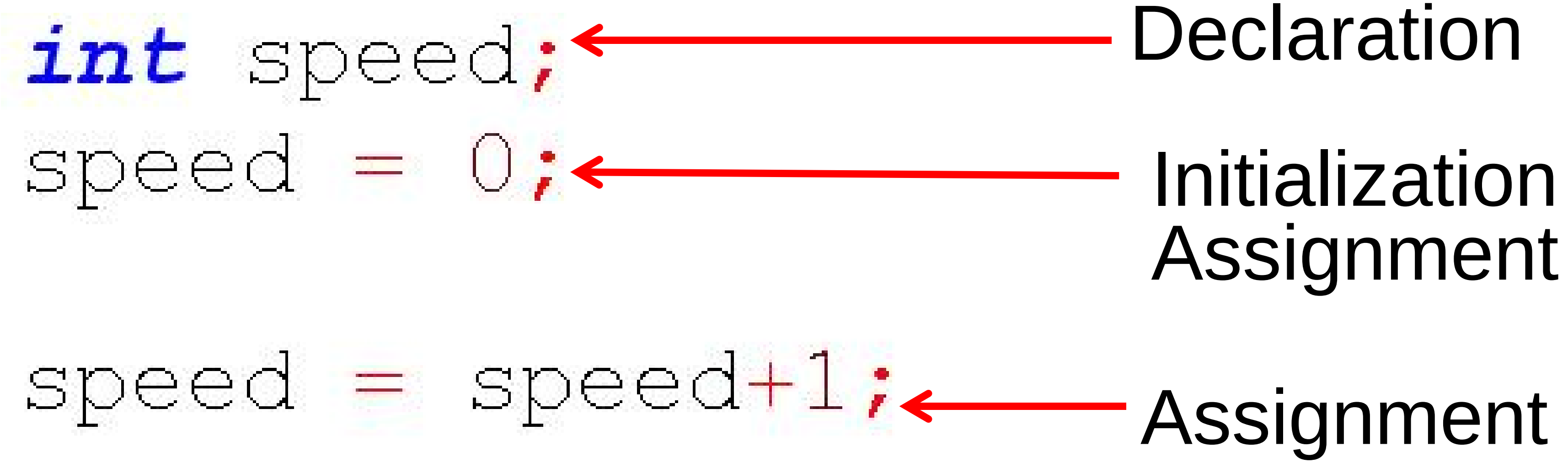


```
startMotor(leftMotor,-1*speed);
```

- This does not change the value of the variable.
- Only referenced when this line executed; in this example, if “a” changes later, it won’t automatically update the motor speed.

Assigning a Value to a Variable

- The assignment operator is the single equal sign
- The right-hand side of the equal sign is evaluated, and then the value is assigned to variable on the left-hand side
- This is not the equality from algebra!



```
int speed;
speed = 0;

speed = speed+1;
```

Declaration

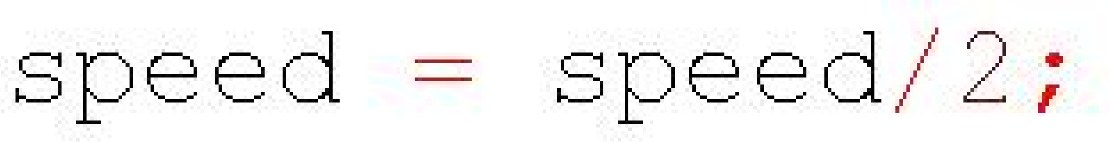
Initialization
Assignment

Assignment

Assigning a Value to a Variable

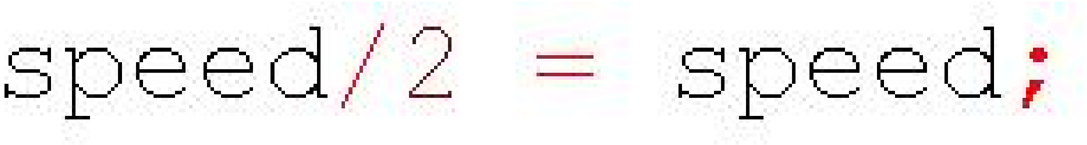
- The left-hand side of the assignment operator must be a variable.

Correct:



```
speed = speed/2;
```

Incorrect:



```
speed/2 = speed;
```

Variable Applications

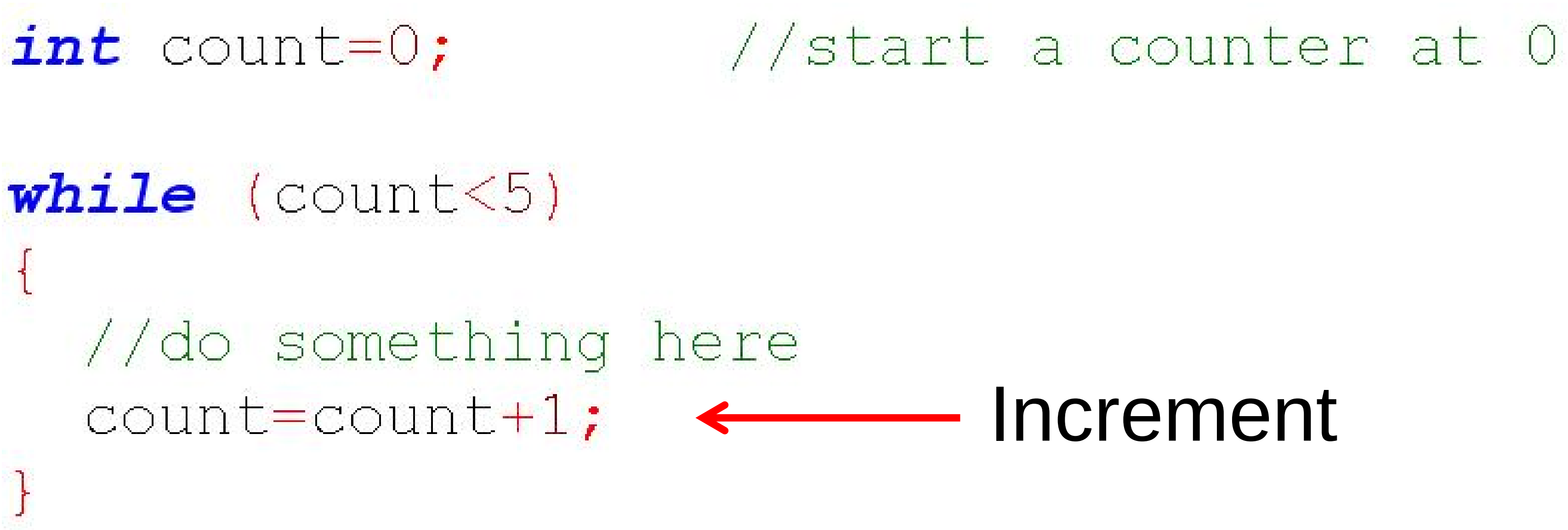
- Variables are needed for most programs. Here are some examples:
  - Example #1: Repeat code 5 times
  - Example #2: Count user’s button presses
  - Example #3: Remember if the user EVER pushed a button
  - Example #4: Remember a maximum value
  - Example #5: Debug a program by remembering which branch of code has been executed.

Variable Application #1: Loop n times

Task description: Start and stop a motor 5 times.

Instead of writing the same code multiple times, use a variable to remember how many times the code has executed so far.

Variable Application #1: Loop n times



```
int count=0;            //start a counter at 0

while (count<5)
{
   //do something here
   count=count+1;
}
```

Increment

- This loop will run five times, with a=0,1,2,3,4

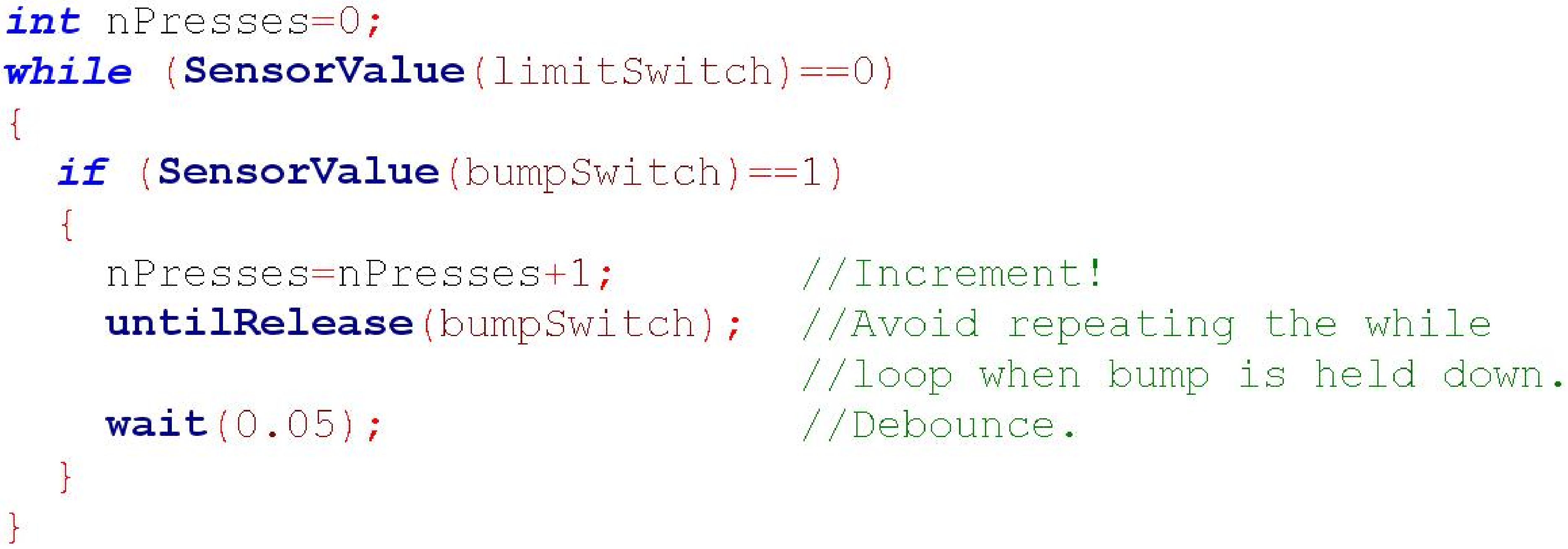
Variable Application #2: Count the user’s actions

Task description: Count the number of times a user does something.

E.g., how many times did the user press the “increase volume” button on a remote?

Use a variable to remember how many times the user performed that action so far.

Variable Application #2: Count the user’s actions



```
int nPresses=0;
while (SensorValue(limitSwitch)==0)
{
  if (SensorValue(bumpSwitch)==1)
  {
    nPresses=nPresses+1;           //Increment!
    untilRelease(bumpSwitch);      //Avoid repeating the while
                                   //loop when bump is held down.
    wait(0.05);                    //Debounce.
  }
}
```

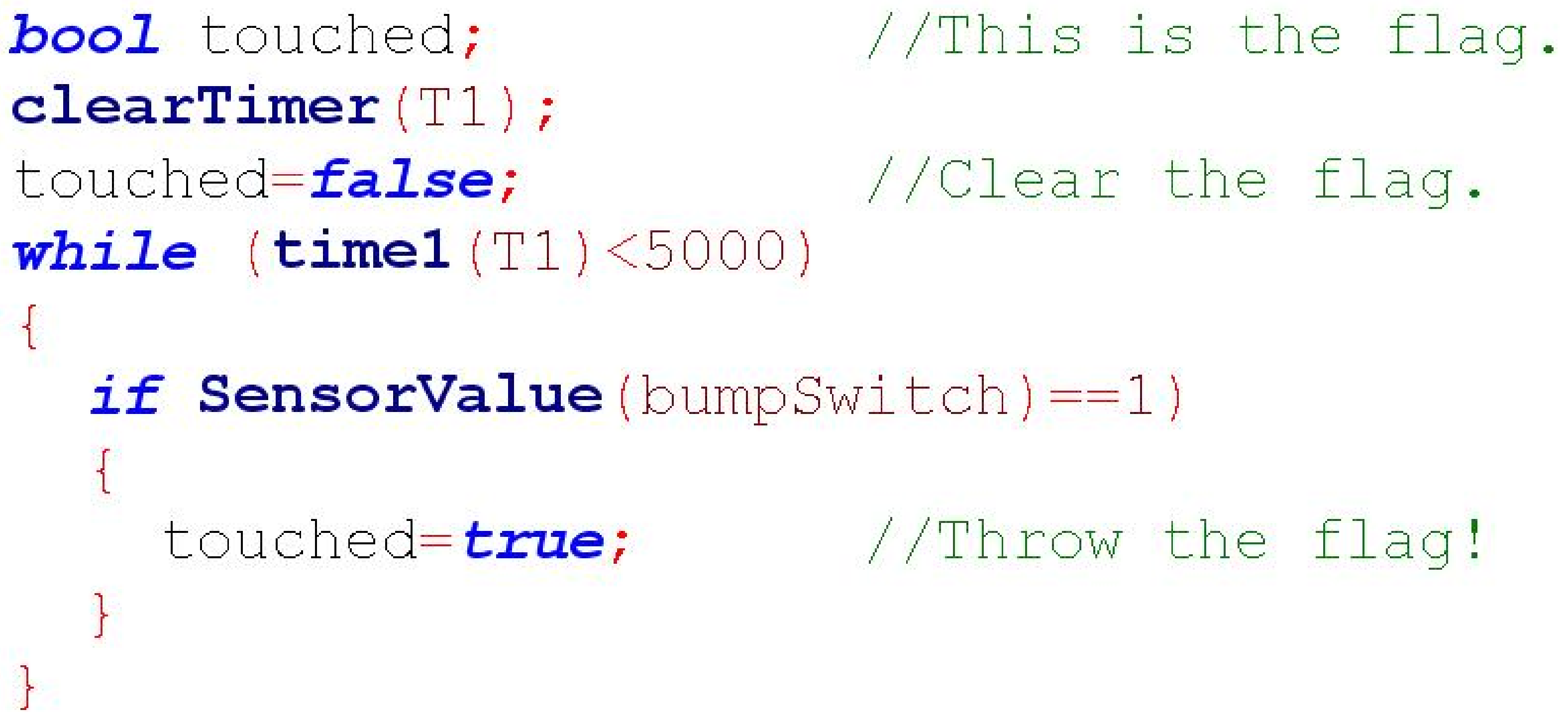
The variable nPresses remembers how many times the bump switch was pressed before the limit switch was pressed.

Variable Application #3: Remember whether an event ever happened.

Task description: Observe the user for 5 seconds and remember whether they EVER pressed a switch, even briefly.

Use a variable to remember whether the event has happened yet. This is called a flag. Once the flag is thrown, it stays thrown until cleared.

Variable Application #3: Set a “flag”



```
bool touched;                //This is the flag.
clearTimer(T1);
touched=false;               //Clear the flag.
while (time1(T1)<5000)
{
  if SensorValue(bumpSwitch)==1)
  {
    touched=true;            //Throw the flag!
  }
}
```

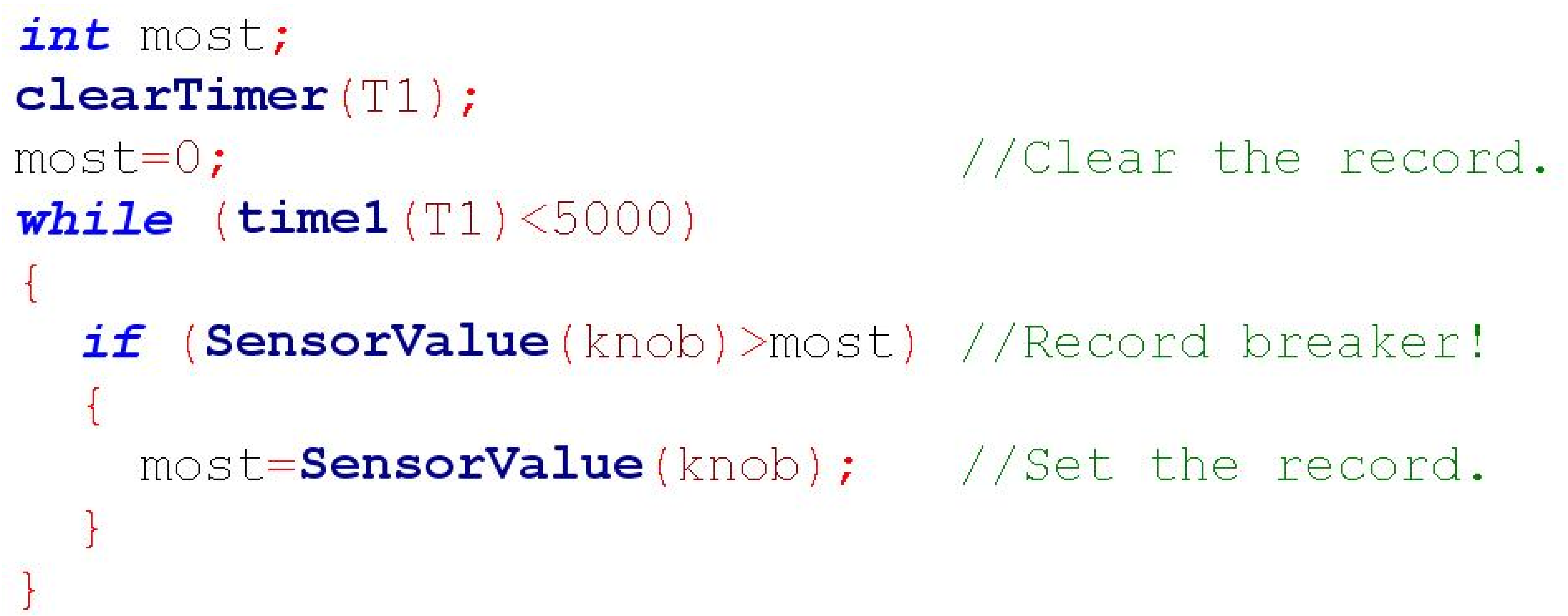
- The variable *touched* remembers if the bump switch was EVER pushed.
- After this code, *touched* will be true, even if bump was pressed and released.

Variable Application #4: Remember the maximum value

Task description: Observe a sensor for 5 seconds and remember its highest value.

Use a variable to remember the biggest value that has occurred so far.

Variable Application #4: Remember a maximum



```
int most;
clearTimer(T1);
most=0;                         //Clear the record.
while (time1(T1)<5000)
{
  if (SensorValue(knob)>most)   //Record breaker!
  {
    most=SensorValue(knob);     //Set the record.
  }
}
```

Similar to the flag, but the variable remembers an “int” instead of a “bool”.

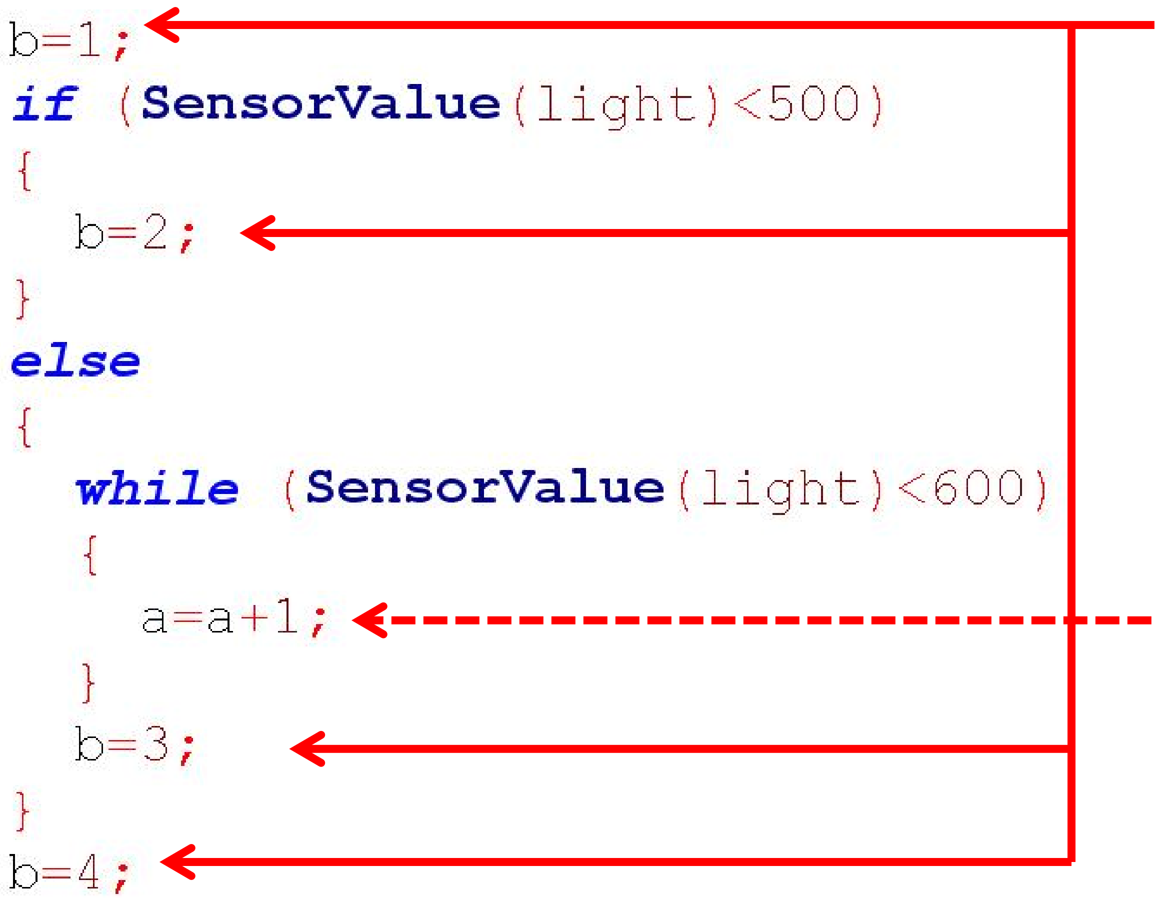
Variable Application #4: Remember what has executed

Run-time errors can be hard to figure out without knowing which parts of your program are being executed!

Sometimes slowing down a program with the step debugger is impractical.

Use a variable to remember (and report) what parts of your program executed.

Variable Application #5: Debug a program



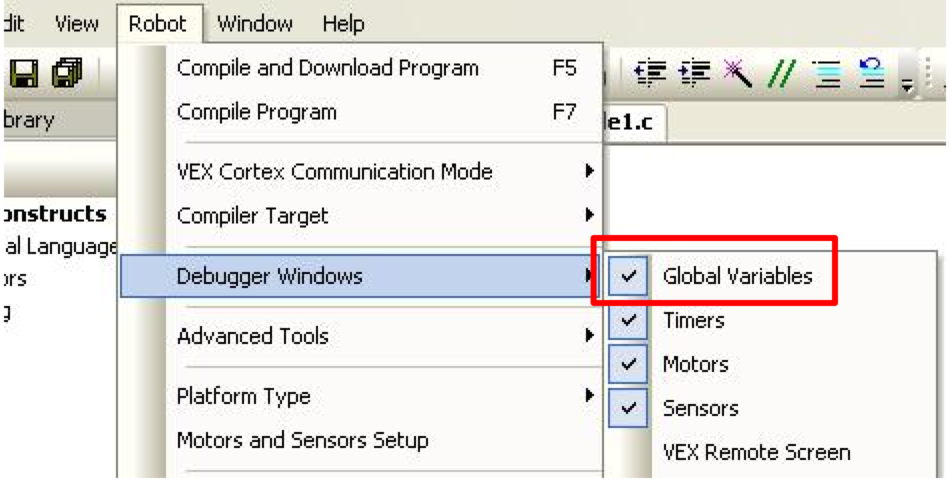
```
b=1;
if (SensorValue(light)<500)
{
   b=2;
}
else
{
   while (SensorValue(light)<600)
   {
      a=a+1;
   }
   b=3;
}
b=4;
```

Remembers the most recent code

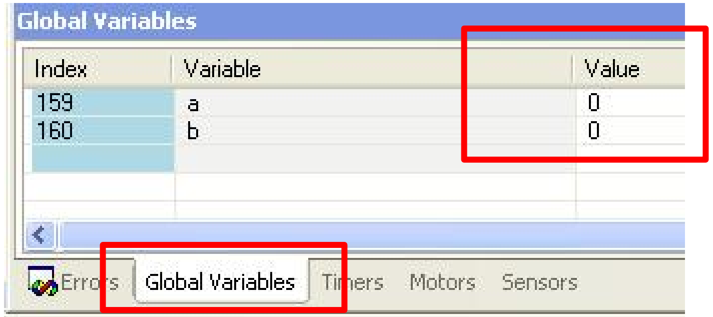
Counts the # of times loop is executed

Variable Application #5: Debug a program

- Activate “global variables” tab in the debug window.



- Variable values reported here as program runs


Global Variables

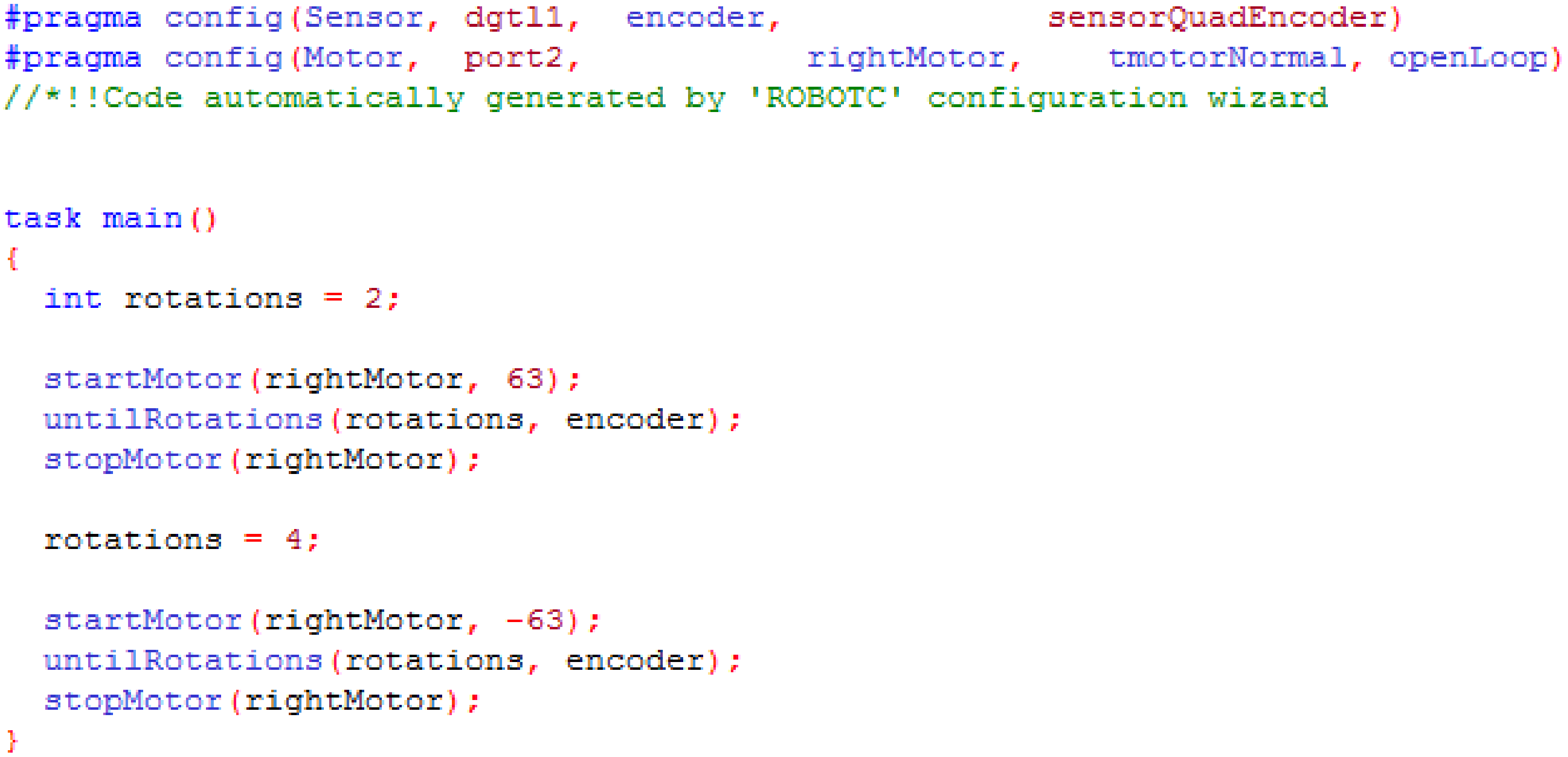
Index	Variable	Value
159	a	0
160	b	0

Global vs. Local Variables

- Variables can have either a “global” or a “local” scope.
  - **Global** variable
    - Can be read or changed from any task or function in your code.
    - Its value can be seen/read *globally*.
  - **Local** variable
    - Belongs only to the task or function in which it was created
    - Value can only be read or changed from within that task or function
    - Value can only be seen/read *locally*
    - Generally the type of variable you’ll want to use, local to “main”

Creating Local Variables (preferred)

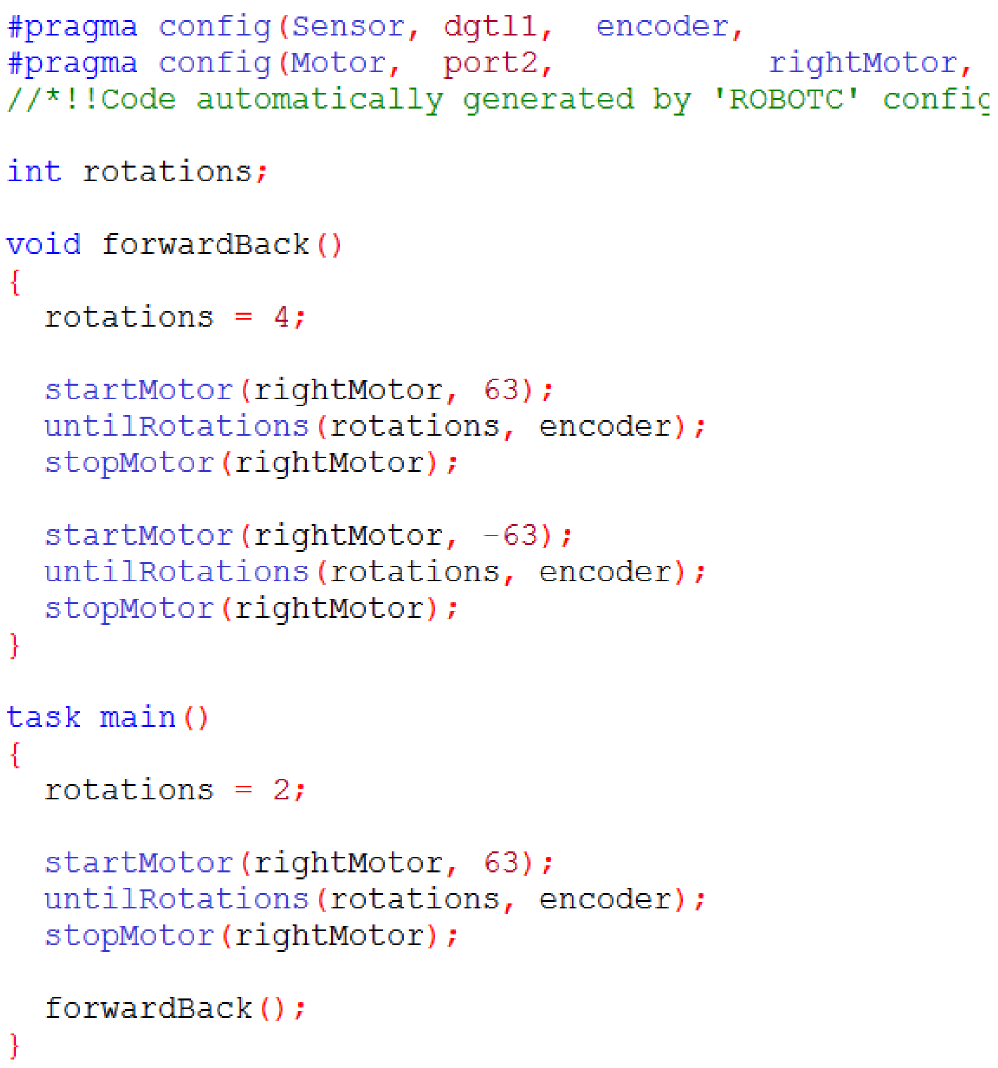
- To create a local variable, declare it within the curly braces of task main or one of your functions.
- You will only be able to change the value of this variable within its task or function.



```
#pragma config(Sensor, dgtl1,  encoder,             sensorQuadEncoder)
#pragma config(Motor,  port2,           rightMotor,    tmotorNormal, openLoop)
//*!!Code automatically generated by 'ROBOTC' configuration wizard


task main()
{
  int rotations = 2;

  startMotor(rightMotor, 63);
  untilRotations(rotations, encoder);
  stopMotor(rightMotor);

  rotations = 4;

  startMotor(rightMotor, -63);
  untilRotations(rotations, encoder);
  stopMotor(rightMotor);
}
```

Creating Global Variables

- To create a global variable, declare it after your pragma statements, but before task main or any function declarations.
- This will allow your variable to be changed by any task or function in your program.



```
#pragma config(Sensor, dgtl1,  encoder,
#pragma config(Motor,  port2,           rightMotor,
//*!!Code automatically generated by 'ROBOTC' config

int rotations;

void forwardBack()
{
  rotations = 4;

  startMotor(rightMotor, 63);
  untilRotations(rotations, encoder);
  stopMotor(rightMotor);

  startMotor(rightMotor, -63);
  untilRotations(rotations, encoder);
  stopMotor(rightMotor);
}

task main()
{
  rotations = 2;

  startMotor(rightMotor, 63);
  untilRotations(rotations, encoder);
  stopMotor(rightMotor);

  forwardBack();
}
```

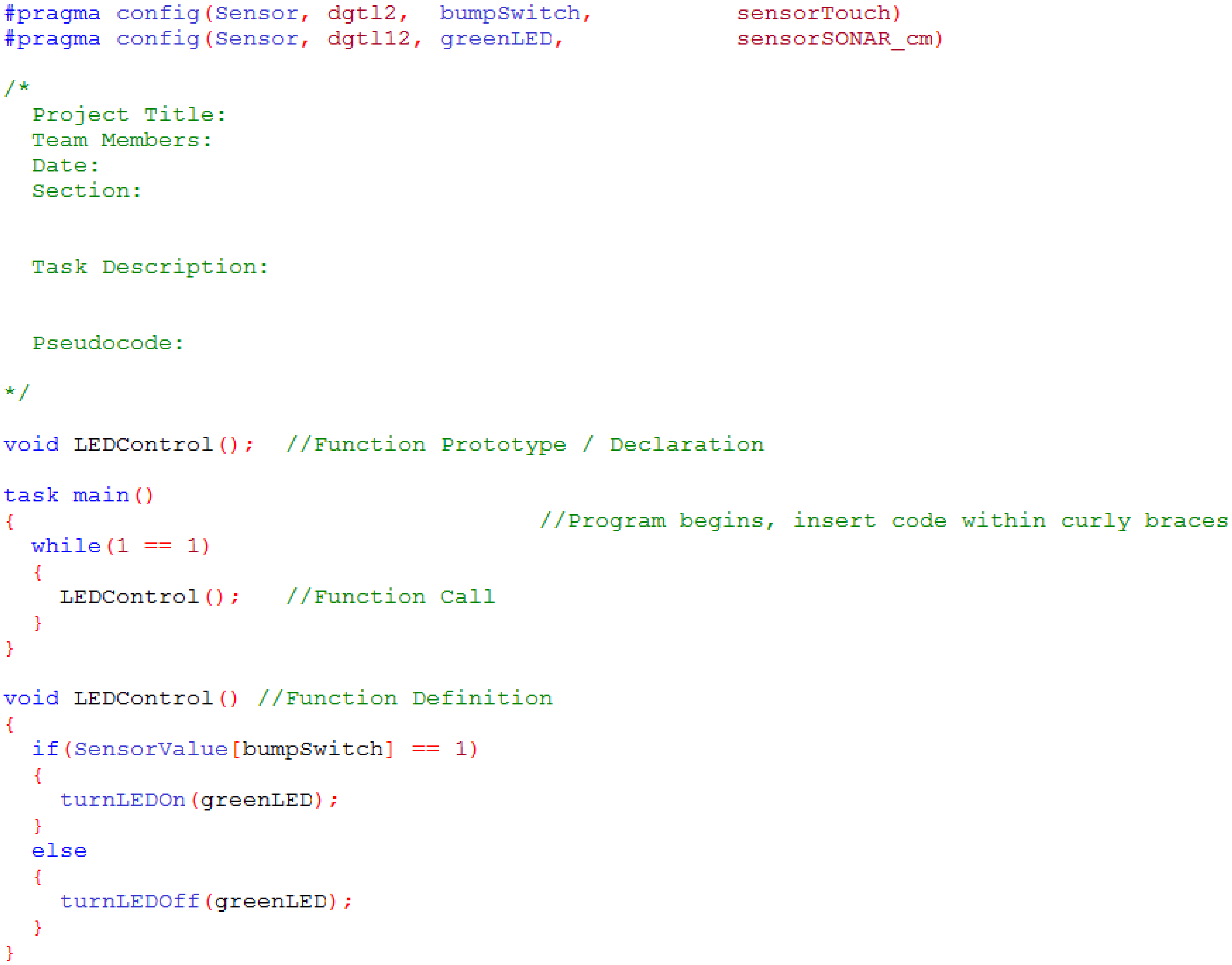
Functions

- Functions
  - Group together several lines of code
  - Referenced many times in task main or in other functions
- Creating Functions

  Example: LED on if bumper is pressed, off if released

  1. Function header (name of function)
  2. Function definition (code in the function)
  3. Function call (where function code will run)

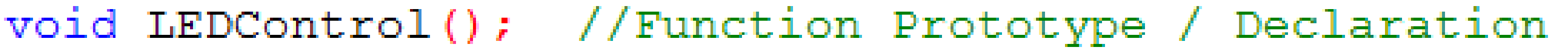
Sample Function “LEDControl()”



```
#pragma config(Sensor, dgtl2,  bumpSwitch,          sensorTouch)
#pragma config(Sensor, dgtl12, greenLED,            sensorSONAR_cm)

/*
  Project Title:
  Team Members:
  Date:
  Section:


  Task Description:


  Pseudocode:

*/

void LEDControl();  //Function Prototype / Declaration

task main()
{                                       //Program begins, insert code within curly braces
  while(1 == 1)
  {
    LEDControl();   //Function Call
  }
}

void LEDControl() //Function Definition
{
  if(SensorValue[bumpSwitch] == 1)
  {
    turnLEDOn(greenLED);
  }
  else
  {
    turnLEDOff(greenLED);
  }
}
```

Function Declaration

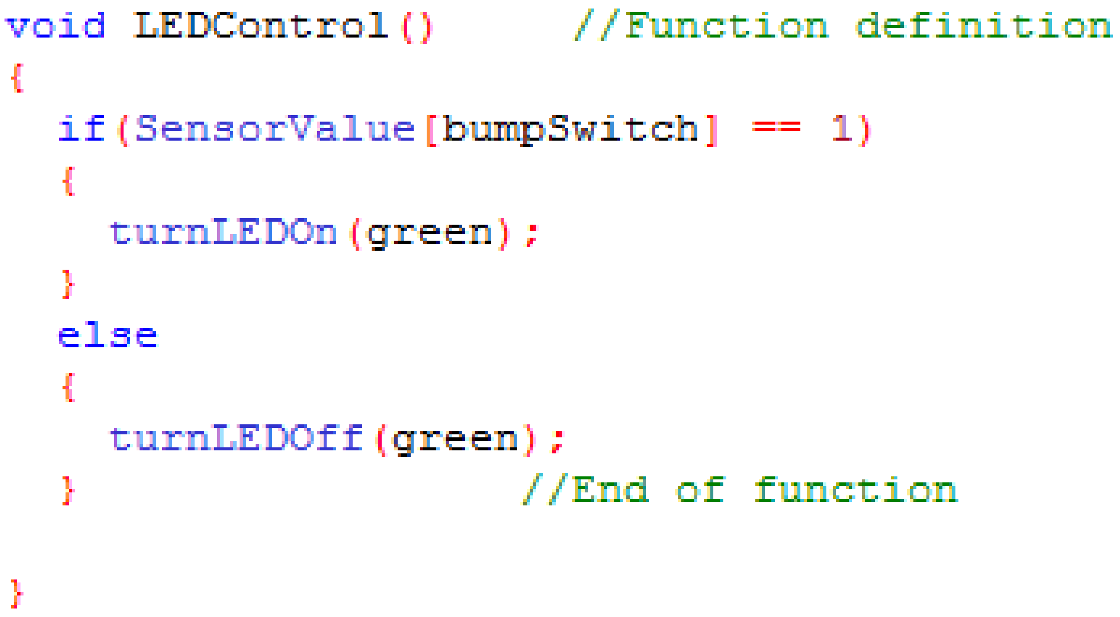
- Function declarations (or prototypes) declare that a function exists and indicates its name
- Function declarations between #pragma statements and task main
- Function declaration optional if function definition is above task main



```
void LEDControl();  //Function Prototype / Declaration
```

Function Definition

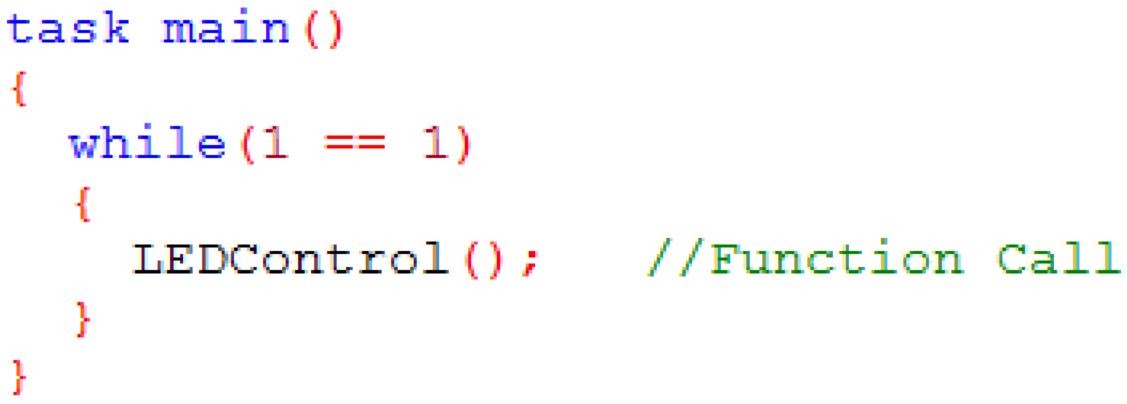
- Function definitions **define** the code that belongs to the function



```
void LEDControl()      //Function definition
{
  if(SensorValue[bumpSwitch] == 1)
  {
    turnLEDOn(green);
  }
  else
  {
    turnLEDOff(green);
  }                 //End of function

}
```

Function Call

- Function calls
  - Call and run code from function
  - Placed in task main or other functions



```
task main()
{
  while(1 == 1)
  {
    LEDControl();    //Function Call
  }
}
```

References

Carnegie Mellon Robotics Academy. (2011). ROBOTC. Retrieved from http://www.robotc.net