PROJECT LEAD THE WAY

PLTW

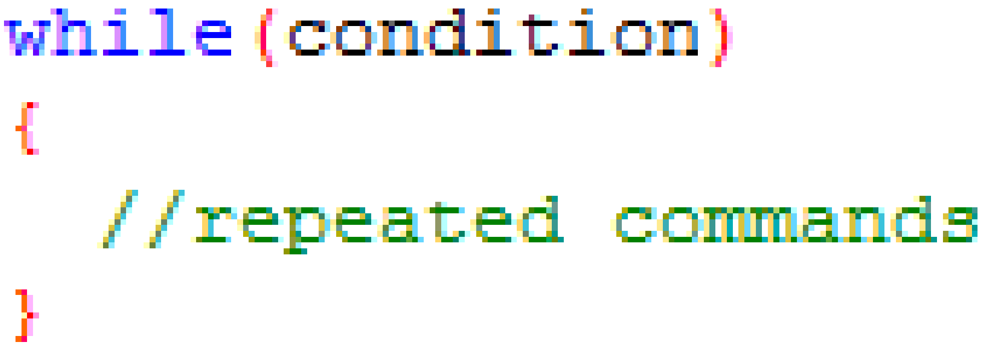
Igniting imagination and innovation through learning.

While Loops and If-Else Structures

ROBOTC Software

While Loops

- While loop is a structure within ROBOTC
- Allows a section of code to be repeated as long as a certain condition remains true

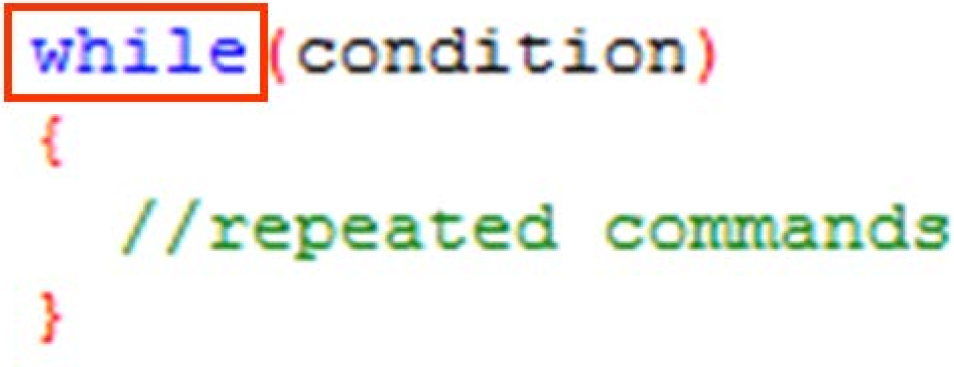


```
while(condition)
{
   //repeated commands
}
```

- Three main parts to every while loop
  1. The word “while”
  2. The condition
  3. Commands to be repeated

1. The Word While

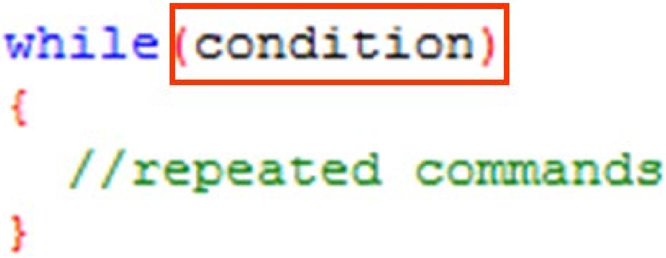
- Every *while* loop begins with the keyword while



```
while(condition)
{
   //repeated commands
}
```

2. The Condition

- Condition controls how long or how many times a *while* loop repeats
  - When condition is true, the *while* loop repeats
  - When condition is false, the *while* loop ends and the remainder of the program executes

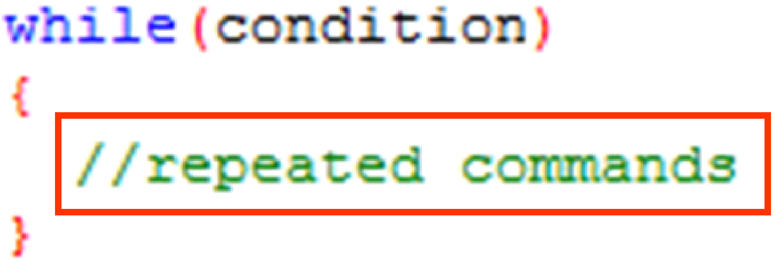


```
while(condition)
{
  //repeated commands
}
```

- Condition is checked once every time loop repeats before commands between curly braces are run

3. Commands To Be Repeated

- Commands between curly braces will repeat while condition is true
- Program checks at the beginning of each pass through the loop

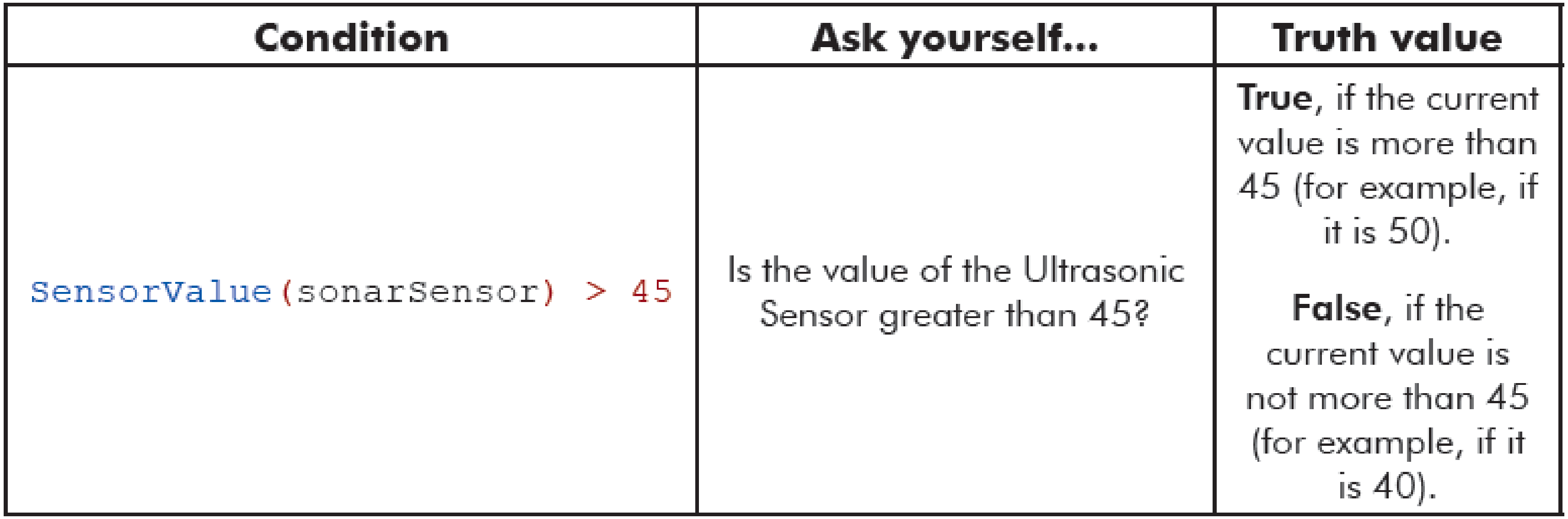


```
while(condition)
{
  //repeated commands
}
```

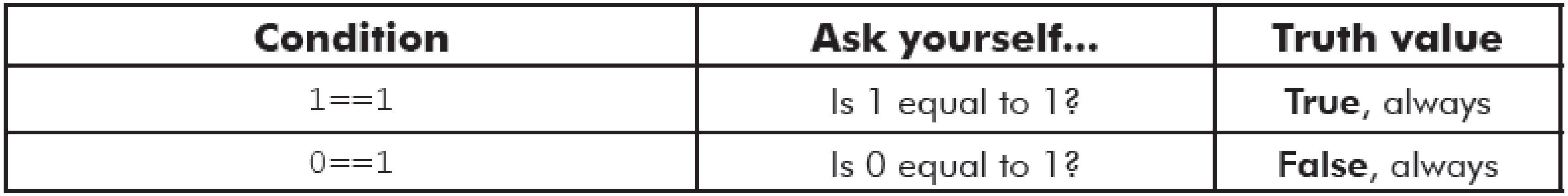
Boolean Logic

- Program decisions are always based on questions
- Only two possible answers
  - yes or no
  - true or false
- Statements that can be only true or false are called Boolean statements
- Their true-or-false value is called a truth value.

Boolean Logic

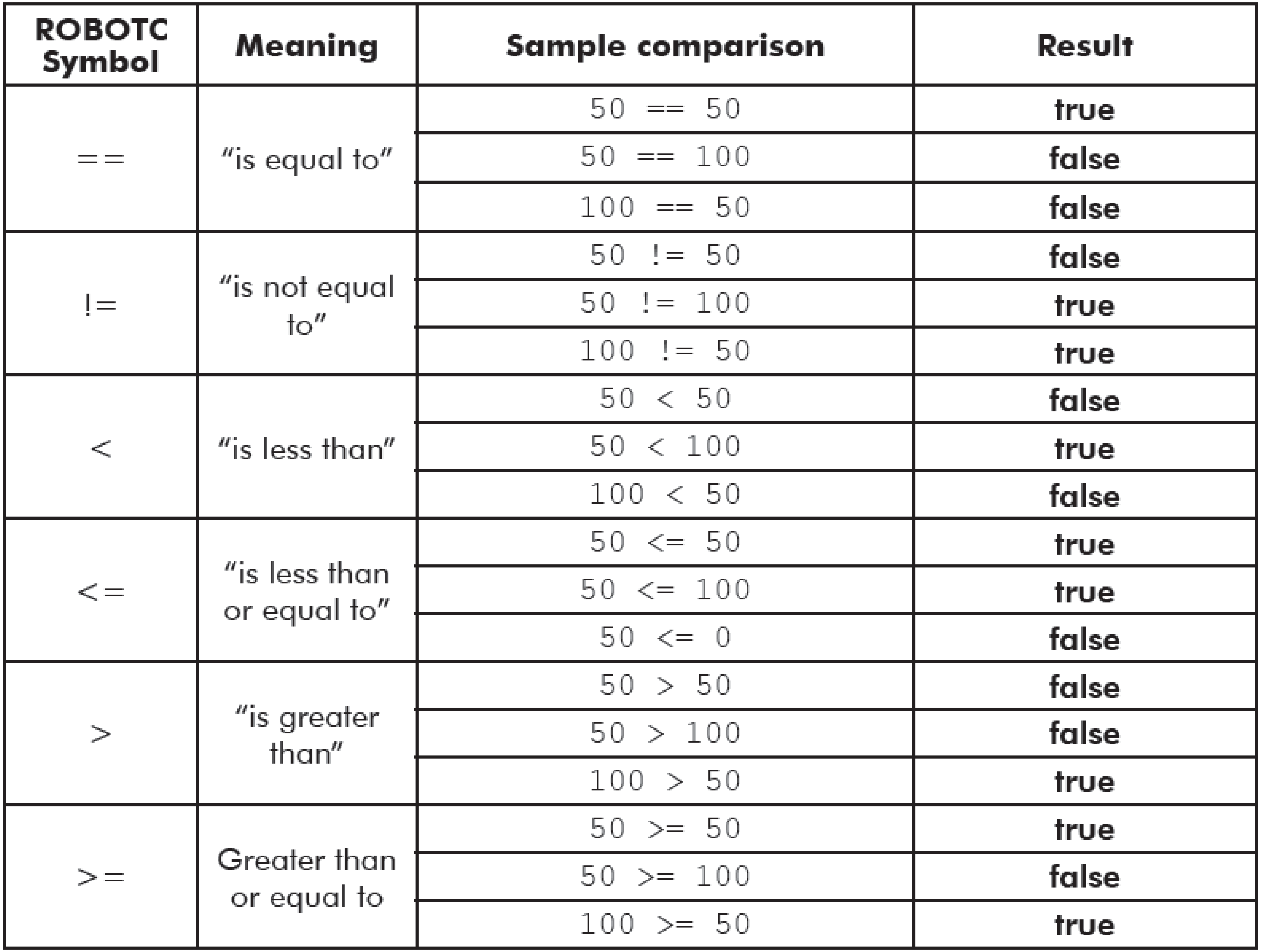


Condition	Ask yourself...	Truth value
`SensorValue(sonarSensor) > 45`	Is the value of the Ultrasonic Sensor greater than 45?	**True**, if the current value is more than 45 (for example, if it is 50). **False**, if the current value is not more than 45 (for example, if it is 40).



Condition	Ask yourself...	Truth value
`1==1`	Is 1 equal to 1?	**True**, always
`0==1`	Is 0 equal to 1?	**False**, always

Boolean Logic



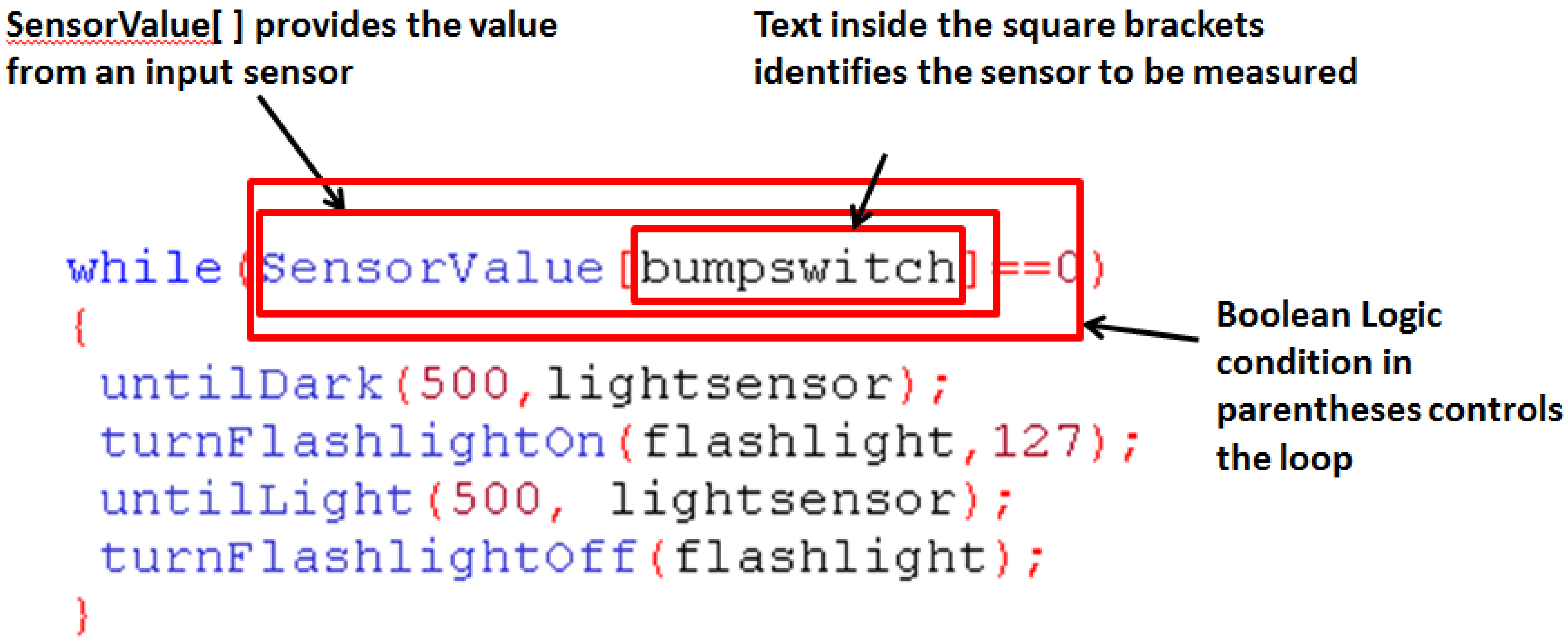
ROBOTC Symbol	Meaning	Sample comparison	Result
==	"is equal to"	50 == 50	true
		50 == 100	false
		100 == 50	false
!=	"is not equal to"	50 != 50	false
		50 != 100	true
		100 != 50	true
<	"is less than"	50 < 50	false
		50 < 100	true
		100 < 50	false
<=	"is less than or equal to"	50 <= 50	true
		50 <= 100	true
		50 <= 0	false
>	"is greater than"	50 > 50	false
		50 > 100	false
		100 > 50	true
>=	Greater than or equal to	50 >= 50	true
		50 >= 100	false
		100 >= 50	true

Writing a condition: Example

- While the bump switch is not pressed:

  wait until it’s dark, then turn on light;

  wait until it’s light, then turn off light



SensorValue[ ] provides the value from an input sensor

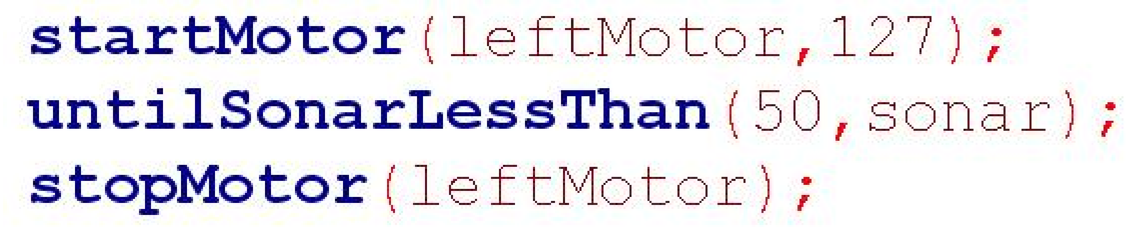
Text inside the square brackets identifies the sensor to be measured

Boolean Logic condition in parentheses controls the loop

```
while(SensorValue[bumpswitch]==0)
{
 untilDark(500,lightsensor);
 turnFlashlightOn(flashlight,127);
 untilLight(500, lightsensor);
 turnFlashlightOff(flashlight);
}
```

While loop: more flexible than an "until" statement

- In this code, a motor runs until an object is within 50 cm.
- The program can't respond to an emergency shutoff switch.
- The program can't control other outputs in response to other inputs.

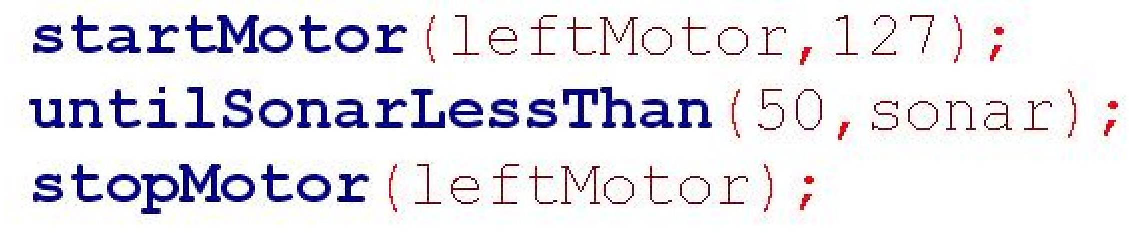


```
startMotor(leftMotor,127);
untilSonarLessThan(50,sonar);
stopMotor(leftMotor);
```

Program waits here until an object is near.

While loop: more flexible than an "until" statement

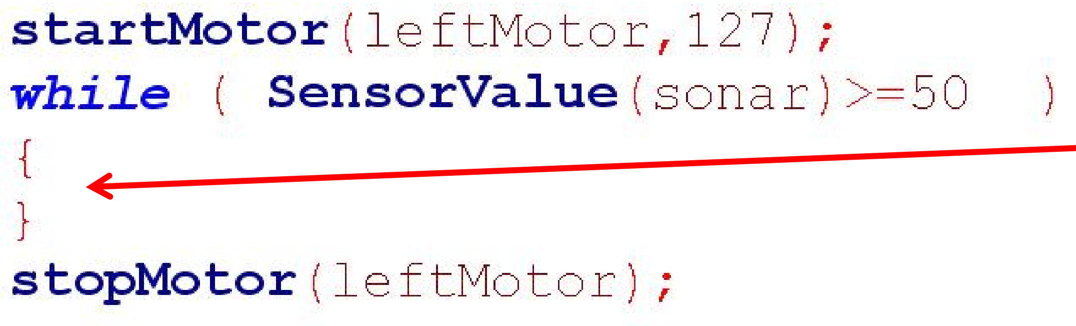
- A while loop can do the same thing as the "until" statement.
- Example code using until statement:



```
startMotor(leftMotor,127);
untilSonarLessThan(50,sonar);
stopMotor(leftMotor);
```

Program waits here until an object is near.

- While loop can do the same thing:



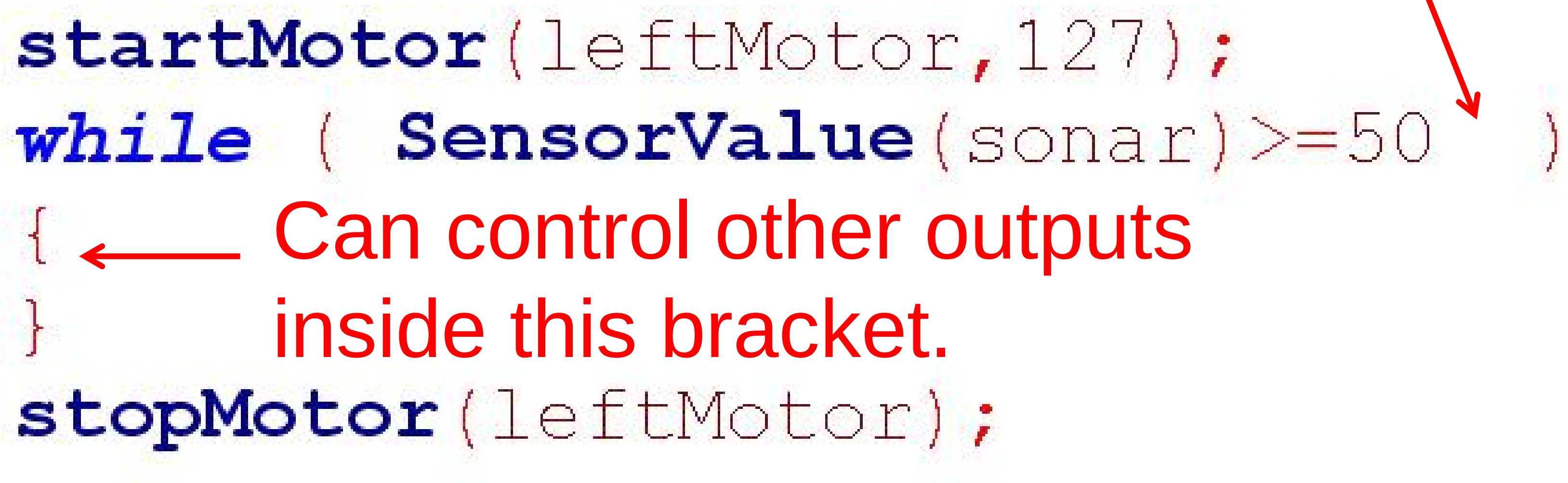
```
startMotor(leftMotor,127);
while ( SensorValue(sonar)>=50  )
{
}
stopMotor(leftMotor);
```

Program loops here until an object is near.

While loop is more flexible than an “until” statement

- Other conditions can be added to the while condition, e.g. an emergency shutoff.
- Other code can be executed in the while loop.

Can expand the condition

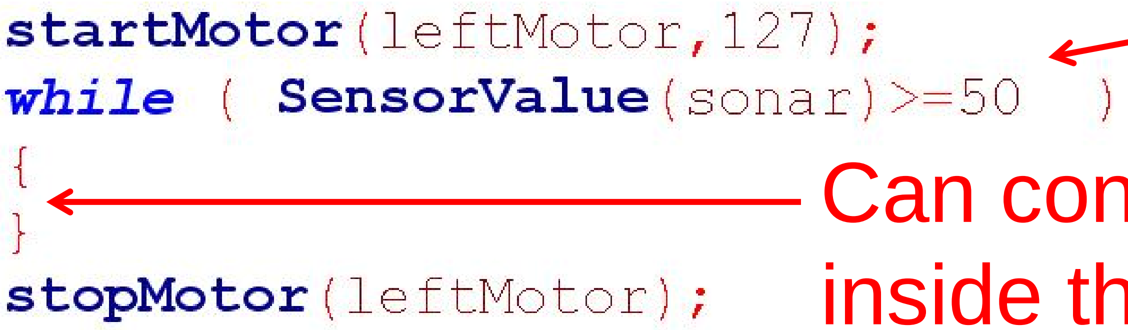


```
startMotor(leftMotor,127);
while ( SensorValue(sonar)>=50  )
{
}
stopMotor(leftMotor);
```

Can control other outputs inside this bracket.

While loop is more flexible than an “until” statement

- Example equivalent to “until”:



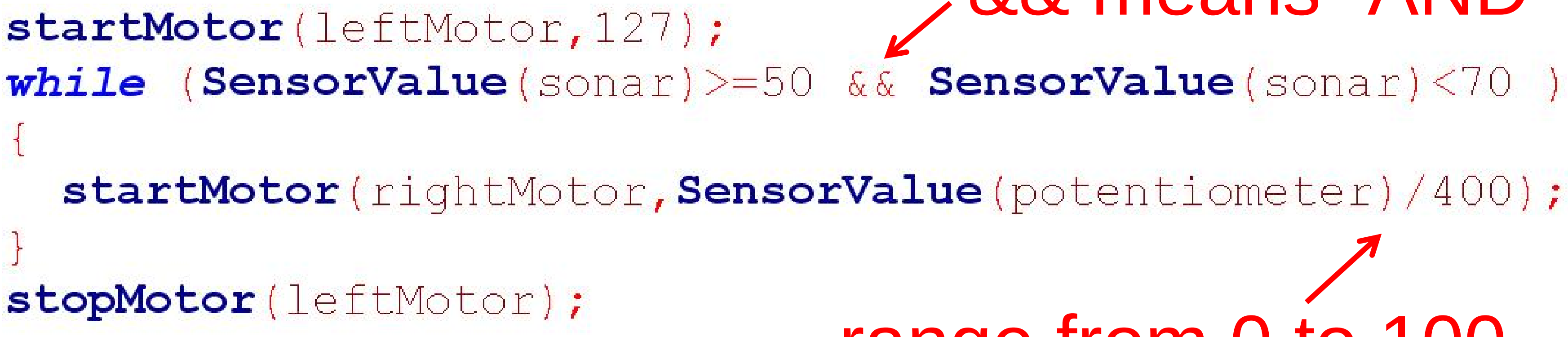
```
startMotor(leftMotor,127);
while ( SensorValue(sonar)>=50  )
{
}
stopMotor(leftMotor);
```

Can expand the condition

Can control other outputs inside this bracket.

- Example using this flexibility:

&& means “AND”



```
startMotor(leftMotor,127);
while (SensorValue(sonar)>=50 && SensorValue(sonar)<70 )
{
  startMotor(rightMotor,SensorValue(potentiometer)/400);
}
stopMotor(leftMotor);
```

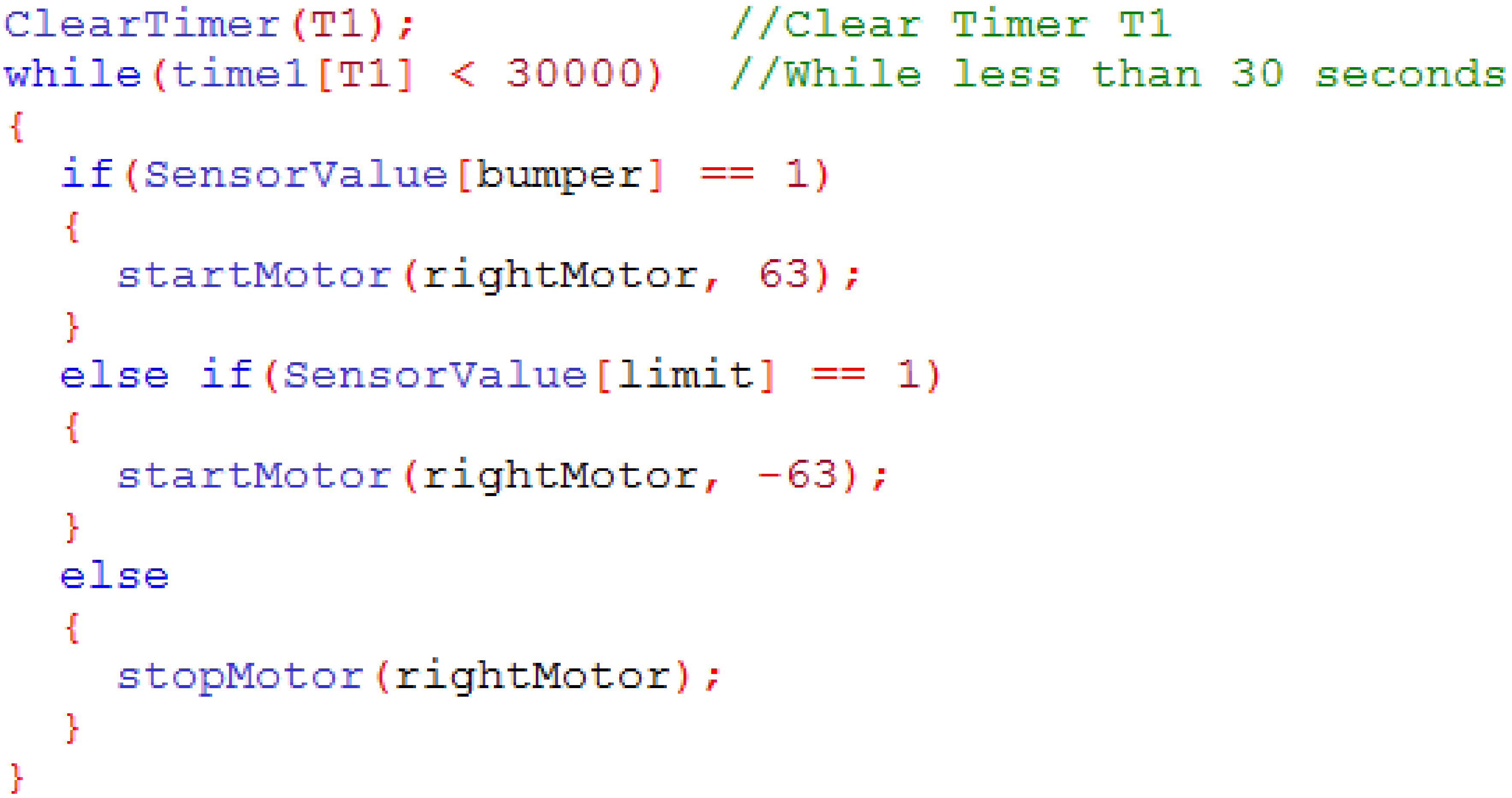
range from 0 to 100

Timers

- Loop control
  - – Where would the *wait* statement go if we wanted the loop to repeat for a controlled amount of time?
  - – Nowhere! We need something else.
- Solution: Timers
  - – Internal stopwatches (4 available)
  - – Like encoders, timers should be cleared before they are used
  - – Be careful: don’t clear a timer in a timed loop

Timers

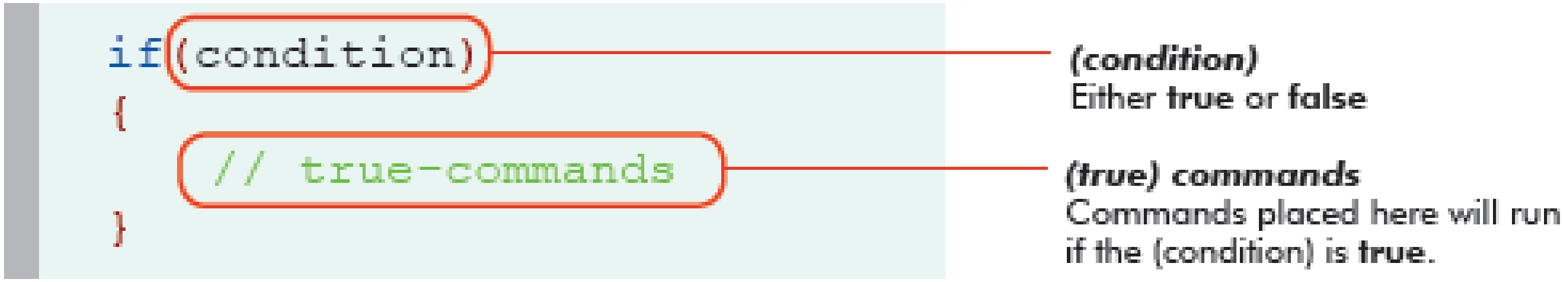
Timer T1 is used as the condition for the *while* loop, which will run for 30 seconds



```
ClearTimer(T1);              //Clear Timer T1
while(time1[T1] < 30000)     //While less than 30 seconds
{
  if(SensorValue[bumper] == 1)
  {
    startMotor(rightMotor, 63);
  }
  else if(SensorValue[limit] == 1)
  {
    startMotor(rightMotor, -63);
  }
  else
  {
    stopMotor(rightMotor);
  }
}
```

If Statements

- *If* statement in the program is evaluated by condition contained in parentheses
  - If condition is true, commands between braces are run
  - If condition is false, those commands are ignored
- Very similar to how a *while* loop works, but does not repeat the code



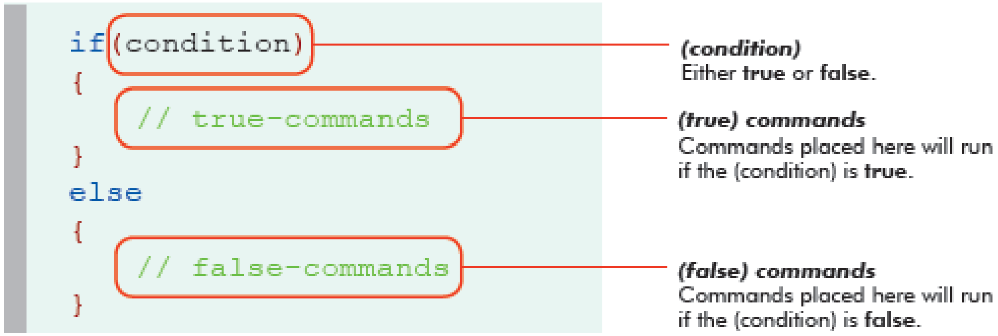
```
if(condition)
{
    // true-commands
}
```

(condition)
Either **true** or **false**

(true) commands
Commands placed here will run if the (condition) is **true**.

If-Else Statements

- *If-else* statement is an expansion of *if* statement
  - *If* checks condition and runs appropriate commands when it evaluates to true
  - *Else* allows code to run when condition is false
  - Either *if* or *else* branch is always run once



```
if(condition)
{
    // true-commands
}
else
{
    // false-commands
}
```

(condition)
Either **true** or **false**.

(true) commands
Commands placed here will run if the (condition) is **true**.

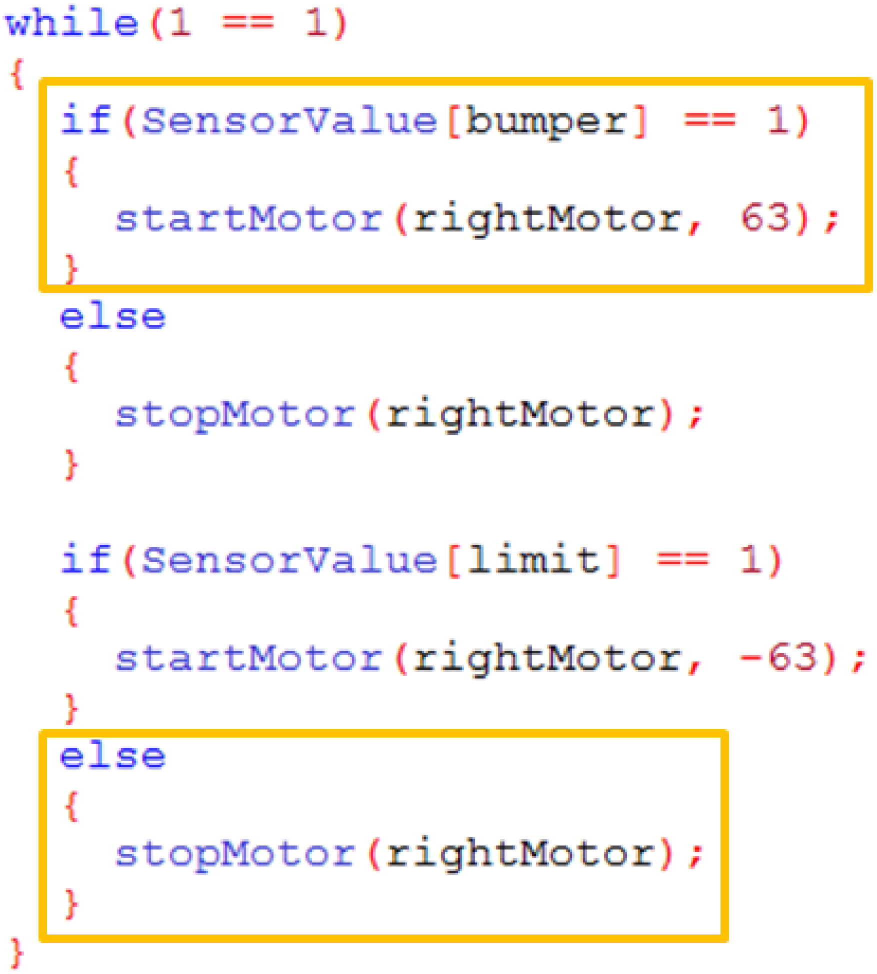
(false) commands
Commands placed here will run if the (condition) is **false**.

Multiple If-Else Statements

- Be careful when using two separate *if-else* statements, particularly if both are used to control the same mechanism
- One branch of each *if-else* statement is always run so that you may create a scenario where the two statements ‘fight’ one another

Multiple If-Else Statements

In this example, if one of the touch sensors is pressed, the rightMotor will be turned on in one *if-else* statement and immediately turned off in the other

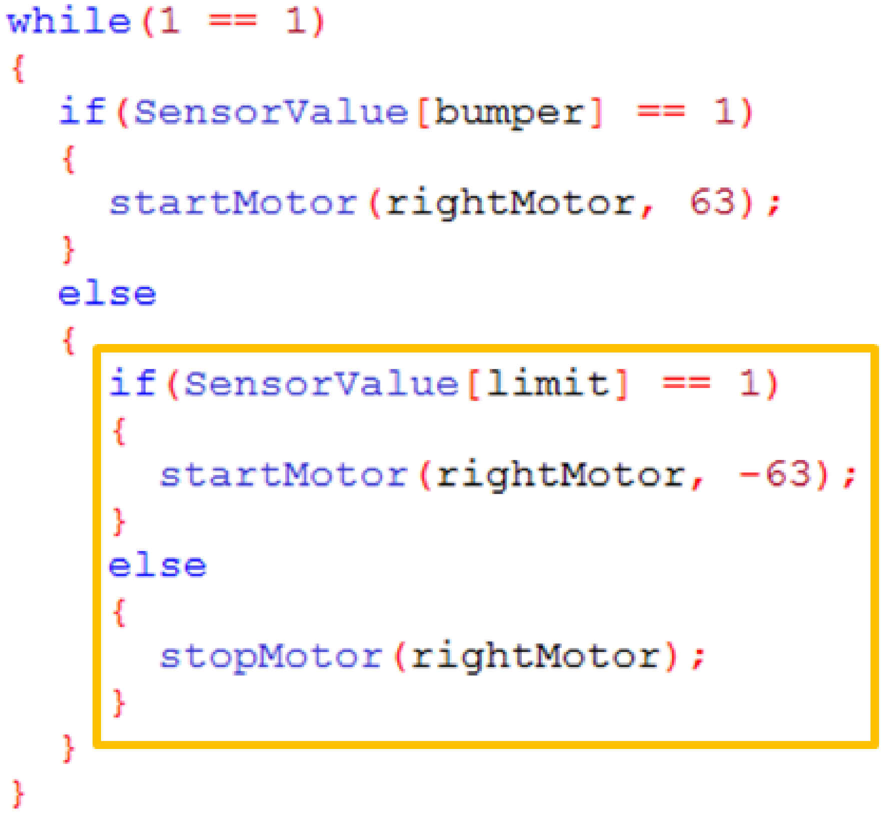


```
while(1 == 1)
{
  if(SensorValue[bumper] == 1)
  {
    startMotor(rightMotor, 63);
  }
  else
  {
    stopMotor(rightMotor);
  }

  if(SensorValue[limit] == 1)
  {
    startMotor(rightMotor, -63);
  }
  else
  {
    stopMotor(rightMotor);
  }
}
```

Multiple If-Else Statements

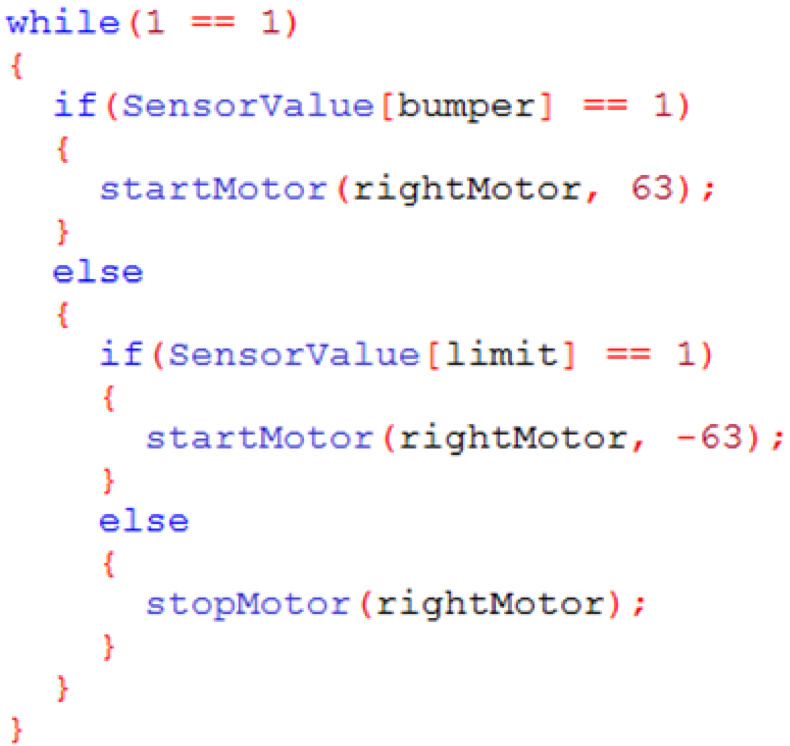
This can be corrected by embedding the second *if-else* within the *else* branch of the first *if-else.* ,The second condition is only checked if the first condition is false.



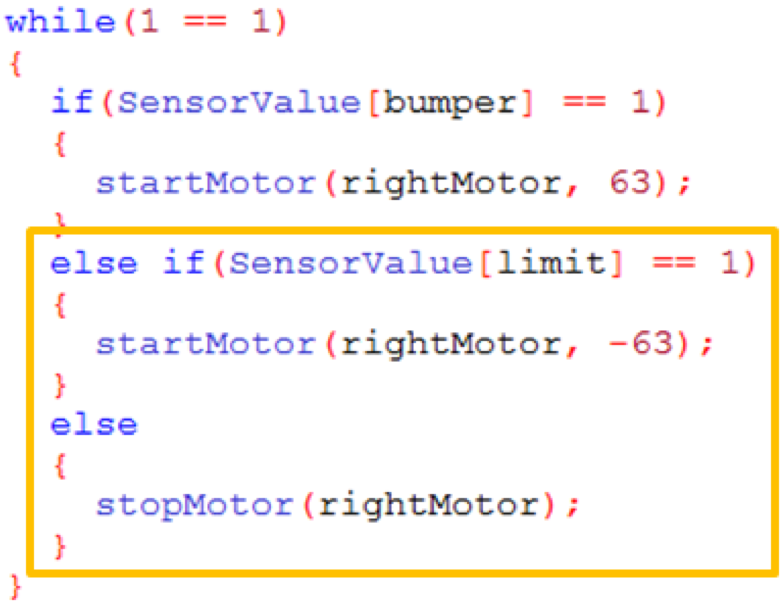
```
while(1 == 1)
{
  if(SensorValue[bumper] == 1)
  {
    startMotor(rightMotor, 63);
  }
  else
  {
    if(SensorValue[limit] == 1)
    {
      startMotor(rightMotor, -63);
    }
    else
    {
      stopMotor(rightMotor);
    }
  }
}
```

Nested if-else statements: else if

An *else {if else}* statement can also be represented as an *else if - else*



```
while(1 == 1)
{
  if(SensorValue[bumper] == 1)
  {
    startMotor(rightMotor, 63);
  }
  else
  {
    if(SensorValue[limit] == 1)
    {
      startMotor(rightMotor, -63);
    }
    else
    {
      stopMotor(rightMotor);
    }
  }
}
```



```
while(1 == 1)
{
  if(SensorValue[bumper] == 1)
  {
    startMotor(rightMotor, 63);
  }
  else if(SensorValue[limit] == 1)
  {
    startMotor(rightMotor, -63);
  }
  else
  {
    stopMotor(rightMotor);
  }
}
```

Using a range of values in a condition

Two strategies will work:

- Boolean logic
- Nested if-else statements

Example:

Task: Control motor with potentiometer “knob”:

Potentiometer Value	Motor Speed
0-500	0
501-1000	63
1001-4095	127

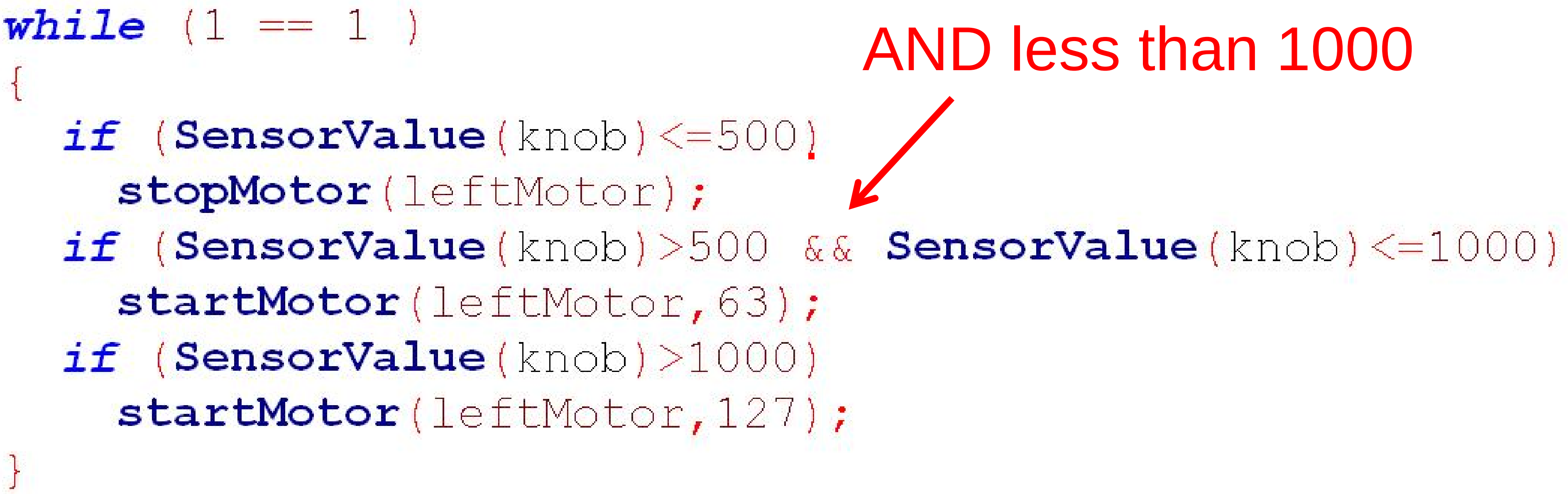
Using a range of values in a condition

Strategy #1: Boolean logic

Boolean operator	RobotC symbol
AND	&&
OR	||

Potentiometer Value	Motor Speed
0-500	0
501-1000	63
1001-4095	127

True only if the sensor value is more than 500 AND less than 1000



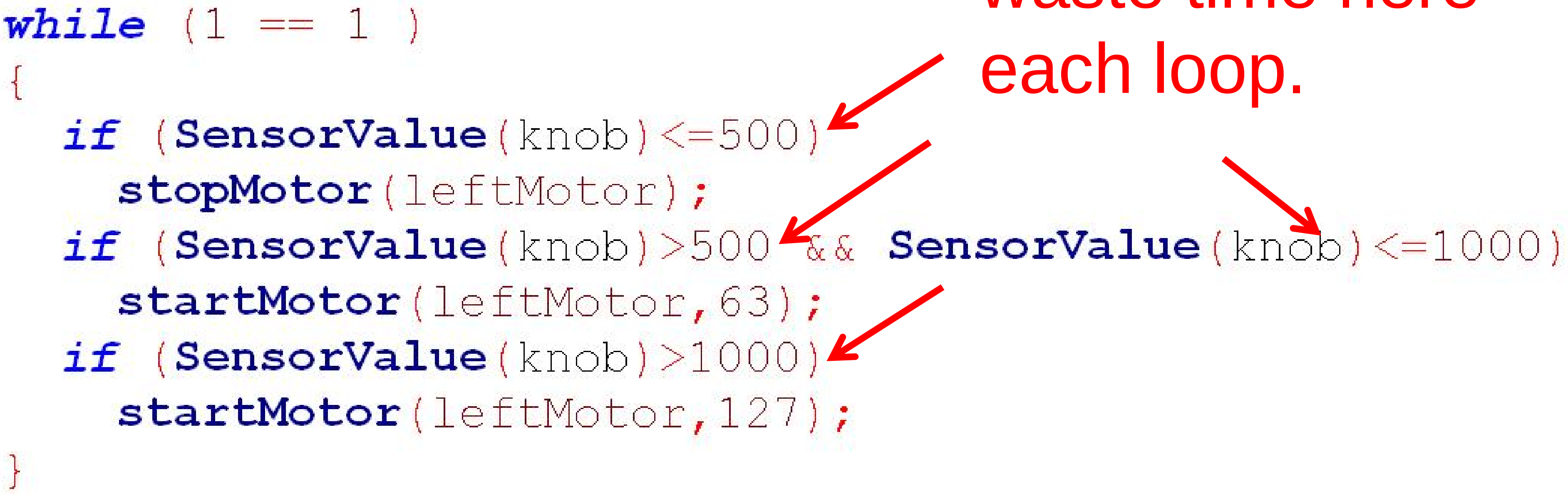
```
while (1 == 1 )
{
  if (SensorValue(knob)<=500)
    stopMotor(leftMotor);
  if (SensorValue(knob)>500 && SensorValue(knob)<=1000)
    startMotor(leftMotor,63);
  if (SensorValue(knob)>1000)
    startMotor(leftMotor,127);
}
```

Using a range of values in a condition

Strategy #1: Boolean logic.

In this example, this strategy wastes time and processor power. The next strategy is better…

Four comparisons waste time here each loop.

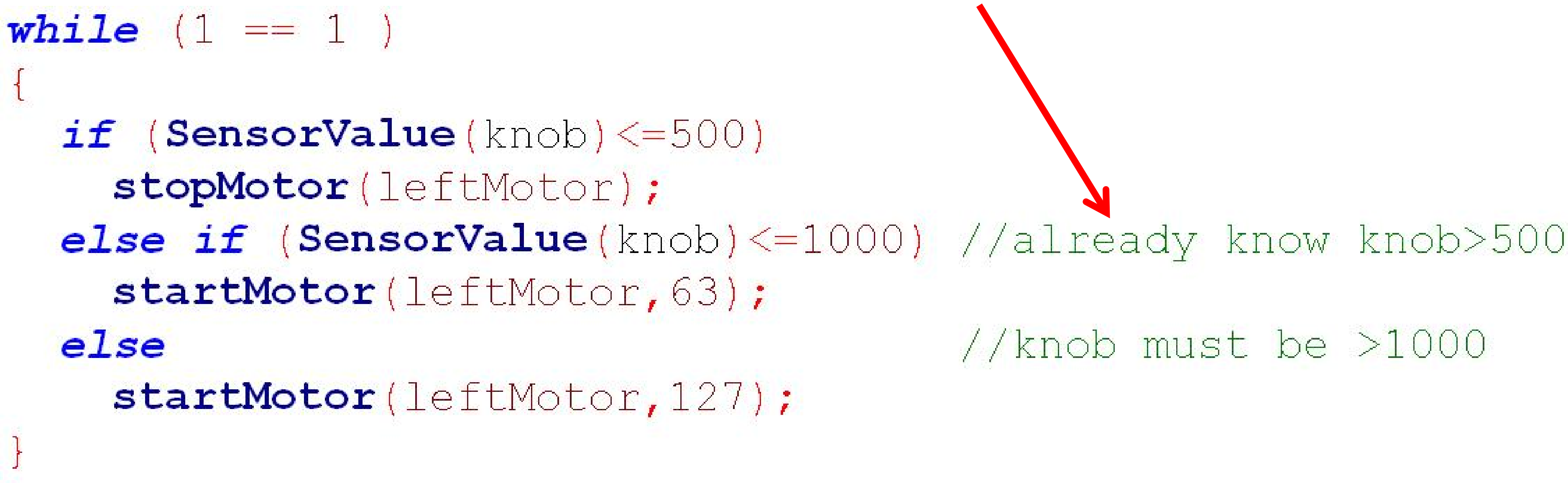


```
while (1 == 1 )
{
  if (SensorValue(knob)<=500)
    stopMotor(leftMotor);
  if (SensorValue(knob)>500 && SensorValue(knob)<=1000)
    startMotor(leftMotor,63);
  if (SensorValue(knob)>1000)
    startMotor(leftMotor,127);
}
```

Using a range of values in a condition

Potentiometer Value	Motor Speed
0-500	0
501-1000	63
1001-4095	127

Strategy #2: Nested if-else preferable in this example. In this case, the false value of the first condition can be used again by nesting a 2nd if statement inside the first else.



```
while (1 == 1 )
{
  if (SensorValue(knob)<=500)
    stopMotor(leftMotor);
  else if (SensorValue(knob)<=1000)  //already know knob>500
    startMotor(leftMotor,63);
  else                               //knob must be >1000
    startMotor(leftMotor,127);
}
```

References

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