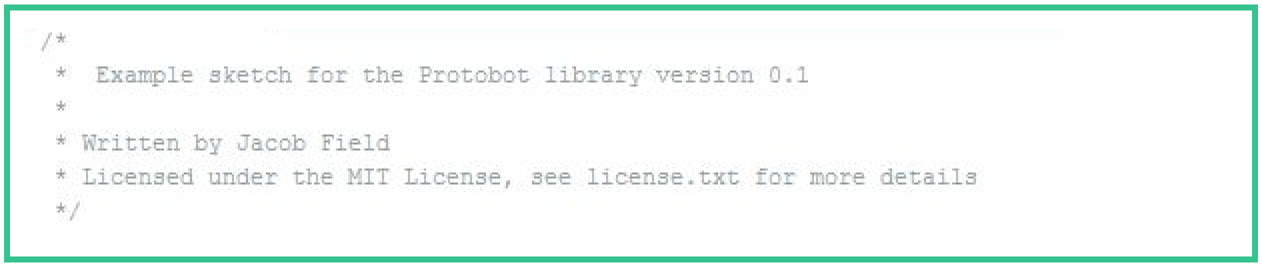
Example program Explanation

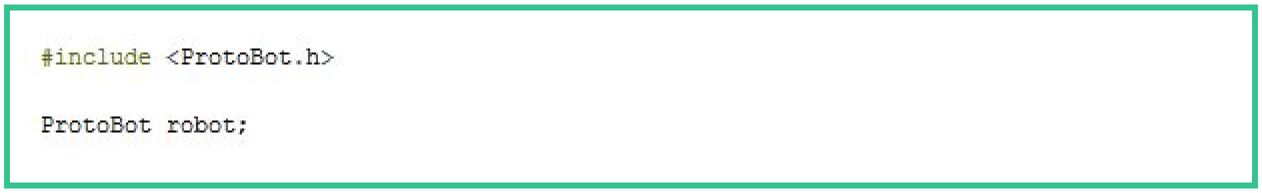
Now that you've uploaded the example program, let's take a look at the code that makes the robot work.

Start at the top of the program.



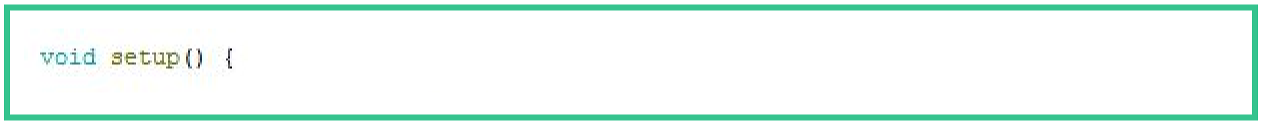
```
/*
 *  Example sketch for the Protobot library version 0.1
 *
 * Written by Jacob Field
 * Licensed under the MIT License, see license.txt for more details
 */
```

The stuff in light grey is a comment. They're just for us humans looking at the code, the robot ignores them. Comments either start with "/*", or with "//".



```
#include <ProtoBot.h>

ProtoBot robot;
```

The next two lines just tell the programming program to include and set up the ProtoBot library. You probably won't ever need to change these.



```
void setup() {
```

This is called a program control structure. Control structures help separate the code into different parts.

The "void startup" structure tells the robot what to do when it first starts running. Any code in it is run only once when the robot is given power, or reset.

```
//Setup the library, init pins
robot.setupRobot();

//Now, tell the robot to wait till the left bump sensor is pressed
robot.waitForBump();
```

These two lines tell the program to set up some stuff for the library, and then tells the program to wait until the left bumper is pressed. You probably won't need to change anything here, either.

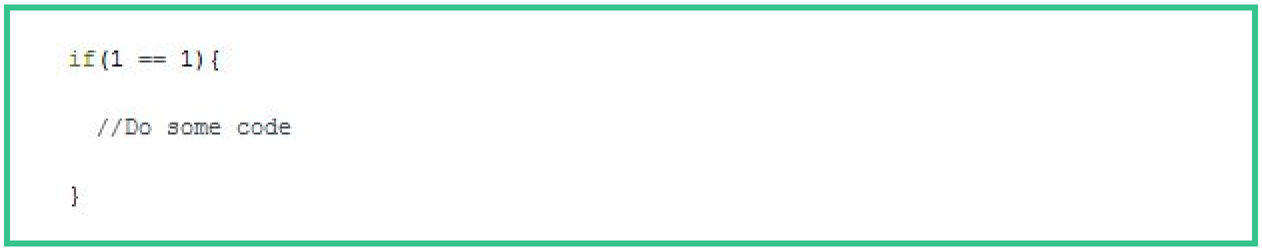
```
//This loops infinitely until the robot is reset or loses power
void loop() {
```

Now we're going to look at the "void loop" control structure. This "loop" runs over and over until the robot loses power, or is reset.

This is also where we're going to write most of our code for the robot.

```
  }

}
```

At the end of any control structure, you'll see a curly bracket "}". This tells the robot to stop looking for code inside of a control structure, so it's important that they're in the right place!

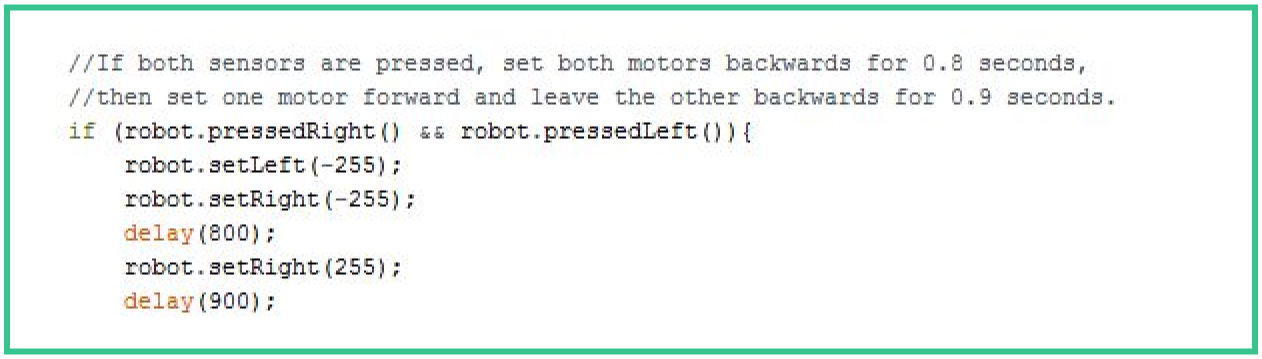


```
if(1 == 1){

  //Do some code

}
```

This is another control structure, called an “if” statement. It checks to see if the code inside the round brackets “()” is true, and if it is, runs the code inside the curly “{}” brackets.

Remember the curly brackets? Here, the open bracket “{“ tells the robot when to start looking for code, and the close “}” bracket tells it when to stop.

Any code you want to run if the “if” statement is true must be inside these brackets.



```
//If both sensors are pressed, set both motors backwards for 0.8 seconds,
//then set one motor forward and leave the other backwards for 0.9 seconds.
if (robot.pressedRight() && robot.pressedLeft()){
    robot.setLeft(-255);
    robot.setRight(-255);
    delay(800);
    robot.setRight(255);
    delay(900);
```

The example program uses a lot of “if” statements, to see if the robot has bumped into something, or run over an edge. It might look confusing, but it’s actually pretty easy to understand.

```
  if (robot.pressedRight() && robot.pressedLeft()){
```

To begin, the “if” statement checks to see if either the Right or Left antenna are pressed. The “robot.pressedRight()” and “robot.pressedLeft()” statements check to see if either antenna is pressed, and the “&&” tells the “if” statement to run the code if either of those are true.

```
      robot.setLeft(-255);
      robot.setRight(-255);
```

These two lines of code turn each motor on, at full speed backwards. The value inside the round brackets is what speed to use. 255 is full speed ahead, 0 is stopped, and -255 is full speed backwards. So, these lines of code make the robot go backwards at full speed.

```
      delay(800);
```

This bit of code just tells the robot to wait for a certain amount of time, before it does the next thing. It will keep doing whatever you told it to do before until the time is up.

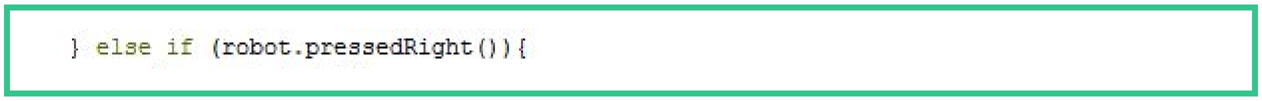
The number inside the round brackets is how long to wait, in Milliseconds. A millisecond is one thousandth of a second, so 1000 milliseconds is 1 second.

```
      robot.setRight(255);
      delay(900);

  }
```

The rest of the code inside the “if” statement is pretty easy to understand. It sets the right motor at full speed ahead, so that the robot will spin, and waits for 900 milliseconds.

The curly bracket tells the robot that that’s the end of the code inside the “if” statement.

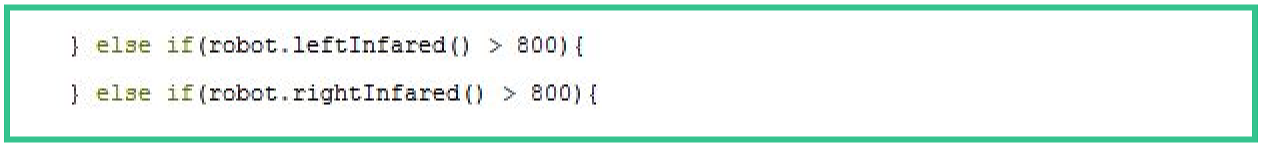
Take a look at the rest of the program. It should be pretty easy to understand, now that you know about “if” statements. You might notice that there’s a special kind of “if” statement being used, though. This is an “else if” statement.



```
} else if (robot.pressedRight()){
```

An “else if” statement can only run if there is an “if” (or an “else if”) statement before it, and if that “if” or “if else” statement didn’t run, because the code inside the round brackets wasn’t true.

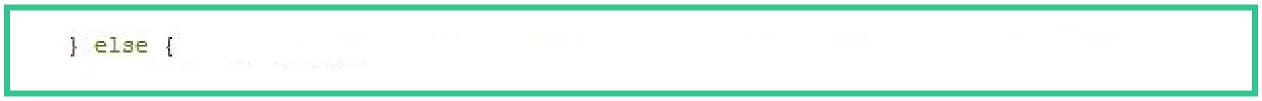
These are useful if you want the robot to only be able to do something if it wasn’t able to do something else.



```
} else if(robot.leftInfared() > 800){
} else if(robot.rightInfared() > 800){
```

Inside the “else if” statements, you’ll also notice another thing the robot can check for. The “robot.leftInfared()” and “robot.rightInfared()” functions check to see if either the left or right infrared (line) sensors are sensing a certain value inside the round brackets.

In this case, 800 is about right for sensing an edge.



```
} else {
```

At last, at the very end of the code, you’ll see another special statement. This is the “else” statement. An “else” statement is like an “else if” statement, but it will always run, if the “if” or “else if” statements before it didn’t run.

This is good if you want the robot to do something by default if it can’t do anything else.