# Project 10 - Serial Controlled Mood Lamp

We will now use the same circuit as in Project 9, but will now delve into the world of serial communications and control our lamp by sending commands from the PC to the Arduino using the Serial Monitor in the Arduino IDE. This project also introduces how we manipulate text strings. So leave the hardware set up the same as before and enter the new code

## Enter the code

```
// Project 10 - Serial controlled RGB Lamp
char buffer[18];
int red, green, blue;
int RedPin = 11;
int GreenPin = 10;
int BluePin = 9;
void setup()
{
Serial.begin(9600);
Serial.flush();
pinMode(RedPin, OUTPUT);
pinMode(GreenPin, OUTPUT);
pinMode(BluePin, OUTPUT);
}
void loop()
{
if (Serial.available() > 0) {
int index=0;
delay(100); // let the buffer fill up
int numChar = Serial.available();
if (numChar>15) {
numChar=15;
}
while (numChar--) {
buffer[index++] = Serial.read();
}
splitString(buffer);
}
}
void splitString(char* data) {
Serial.print("Data entered: ");
Serial.println(data);
char* parameter;
parameter = strtok (data, " ,");
while (parameter != NULL) {
setLED(parameter);
parameter = strtok (NULL, " ,");
}
// Clear the text and serial buffers
for (int x=0; x<16; x++) {
buffer[x]='\0';
}
Serial.flush();
}



void setLED(char* data) {
if ((data[0] == 'r') || (data[0] == 'R')) {
int Ans = strtol(data+1, NULL, 10);
Ans = constrain(Ans,0,255);
analogWrite(RedPin, Ans);
Serial.print("Red is set to: ");
Serial.println(Ans);
}
if ((data[0] == 'g') || (data[0] == 'G')) {
int Ans = strtol(data+1, NULL, 10);
Ans = constrain(Ans,0,255);
analogWrite(GreenPin, Ans);
Serial.print("Green is set to: ");
Serial.println(Ans);
}
if ((data[0] == 'b') || (data[0] == 'B')) {
int Ans = strtol(data+1, NULL, 10);
Ans = constrain(Ans,0,255);
analogWrite(BluePin, Ans);
Serial.print("Blue is set to: ");
Serial.println(Ans);
}
```

Once you've verified the code, upload it to your Arduino.

Now when you upload the program nothing seems to happen. This is because the program is waiting for your input. Start the Serial Monitor by clicking it's icon in the Arduino IDE taskbar.

In the Serial Monitor text window you can now enter the R, G and B values for each of the 3 LED's manually and the LED's will change to the colour you have input.

E.g. If you enter R255 the Red LED will display at full brightness.

If you enter R255, G255, then both the red and green LED's will display at full brightness.

Now enter R127, G100, B255 and you will get a nice purplish colour.

If you type, r0, g0, b0 all the LED's will turn off.

The input text is designed to accept both a lowercase or upper-case R, G and B and then a value from 0 to 255. Any values over 255 will be dropped down to 255 maximum. You can enter a comma or a space in between parameters and you can enter 1, 2 or 3 LED values at any one time. E.g.

```
r255 b100 r127

b127 g127

G255, B0 B127,

R0, G255
```

Etc.

# Project 10 - Code Overview

This project introduces a whole bunch of new concepts, including serial communication, pointers and string manipulation. So, hold on to your hats this will take a lot of explaining.

First we set up an array of char (characters) to hold our text string. We have made it 18 characters long, whichis longer than the maximum of 16 we will allow to ensure we don't get "buffer overflow" errors.

```
char buffer[18];
```

We then set up the integers to hold the red, green and blue values as well as the values for the digital pins.

```
int red, green, blue;

int RedPin = 11;
int GreenPin = 10;
int BluePin = 9;
```

In our setup function we set the 3 digital pins to be outputs. But, before that we have the Serial.begin command.

```
void setup()
{
  Serial.begin(9600);
  Serial.flush();
pinMode(RedPin, OUTPUT);
pinMode(GreenPin, OUTPUT);
pinMode(BluePin, OUTPUT); }
```

Serial.begin tells the Arduino to start serial communications and the number within the parenthesis, in this case 9600, sets the baud rate (characters per second) that the serial line will communicate at.

The Serial.flush command will flush out any characters that happen to be in the serial line so that it is empty and ready for input/output.

The serial communications line is simply a way for the Arduino to communicate with the outside world, in this case to and from the PC and the Arduino IDE's Serial Monitor.

In the main loop we have an if statement. The condition it is checking for is

```
  if (Serial.available() > 0) {
```

The Serial.available command checks to see if any characters have been sent down the serial line. If any characters have been received then the condition is met and the code within the if statements code block is now executed.

```
if (Serial.available() > 0) {
int index=0;
delay(100); // let the buffer fill up
int numChar = Serial.available();
if (numChar>15) {
numChar=15;
}
while (numChar--) {
buffer[index++] = Serial.read();
}
splitString(buffer);
}
}
```

An integer called  index is declared and initialised as zero. This integer will hold the position of a pointer to the characters within the char array.

We then set a delay of 100. The purpose of this is to ensure that the serial buffer (the place in memory where the serial data that is received is stored prior to processing) is full before we carry on and process the data. If we don't do that, it is possible that the function will execute and start to process the text string, before we have received all of the data. The serial communications line is very slow compared to the speed the rest of the code is executing at. When you send a string of characters the Serial.available function will immediately have a value higher than zero and the if function will start to execute. If we didn't have the delay(100) statement in there it could start to execute the code within the if statement before all of the text string had been received and the serial data may only be the first few characters of the line of text entered.

After we have waited for 100ms for the serial buffer to fill up with the data sent, we then declare and initialise the numChar integer to be the number of characters within the text string.

E.g. If we sent this text in the Serial Monitor:

R255, G255, B255

Then the value of numChar would be 17. It is 17 and not 16 as at the end of each line of text there is an invisible character called a NULL character. This is a 'nothing' symbol and simply tells the Arduino that the end of the line of text has been reached.

The next if statement checks if the value of numChar is greater than 15 or not and if so it sets it to be 15. This ensures that we don't overflow the array `char buffer[18];`

After this comes a while command. This is something we haven't come across before so let me explain.

We have already used the for loop, which will loop a set number of times. The while statement is also a loop, but one that executes only while a condition is true.

The syntax is

```
while(expression) {
!       // statement(s)
}
```

In our code the while loop is

```
while (numChar--)
{       buffer[index++] =
Serial.read();     }
```

The condition it is checking is simply numChar, so in other words it is checking that the value stored in the integer numChar is not zero. numChar has -- after it. This is what is known as a post-decrement. In other words, the value is decremented AFTER it is used. If we had used --numChar the value in numChar would be decremented (have one subtracted from it) before it was evaluated. In our case, the while loop checks the value of numChar and then subtracts one from it. If the value of numChar was not zero before the decrement, it then carries out the code within its code block.

numChar is set to the length of the text string that we have entered into the Serial Monitor window. So, the code within the while loop will execute that many times.

The code within the while loop is `buffer[index++]`

```
= Serial.read();
```

Which sets each element of the buffer array to each character read in from the Serial line. In other words, it fills up the buffer array with the letters we have entered into the Serial Monitor's text window.

The Serial.read() command reads incoming serial data, one byte at a time.

So now that our character array has been filled with the characters we entered in the Serial Monitor the while loop will end once numChar reaches zero (i.e. The length of the string).

After the while loop we have

```
splitString(buffer);
```

Which is a call to one of the two functions we have created and called splitString(). The function looks like this:

```
void splitString(char* data) {
Serial.print("Data entered: ");
Serial.println(data);
char* parameter;
parameter = strtok (data, " ,");
while (parameter != NULL) {
setLED(parameter);
parameter = strtok (NULL, " ,");
}
// Clear the text and serial buffers
for (int x=0; x<16; x++) {
buffer[x]='\0';
}
Serial.flush();

}
```

We can see that the function returns no data, hence it's data type has been set to void. We pass the function one parameter and that is a char data type that we have called data. However, in the C and C++ programming languages you are not allowed to send a character array to a function. We have got around that by using a pointer. We know we have used a pointer as an asterisk '*' has been added to the variable name *data. Pointers are quite an advanced subject in C so we won't go into too much detail about them. All you need to know for now is that by declaring 'data' as a pointer it is simply a variable that points to another variable.

You can either point it to the address that the variable is stored within memory by using the & symbol, or in our case, to the value stored at that memory address using the * symbol. We have used it to 'cheat' the system, as we are not allowed to send a character array to a function. However we are allowed to send a pointer to a character array to our function. So, we have declared a variable of data type Char and called it data, but the * symbol before it means that it is 'pointing to' the value stored within the 'buffer' variable.

When we called splitString we sent it the contents of 'buffer' (actually a pointer to it as we saw above).

```
splitString(buffer);
```

So we have called the function and passed it the entire contents of the buffer character array.

The first command is

```
Serial.print("Data entered: ");
```

and this is our way of sending data back from the Arduino to the PC. In this case the print command sends whatever is within the parenthesis to the PC, via the USB cable, where we can read it in the Serial Monitor window. In this case we have sent the words

"Data entered: ". Text must be enclosed within quotes "". The next line is similar Serial.println(data);

and again we have sent data back to the PC, this time we send the char variable called data. The Char type variable we have called 'data' is a copy of the contents of the 'buffer' character array that we passed to the function. So, if our text string entered
was

R255 G127 B56

Then the

Serial.println(data);

Command will send that text string back to the PC and print it out in the Serial Monitor window (make sure you have enabled the Serial Monitor window first).

This time the print command has ln on the end to make it println. This simply means 'print' with a 'linefeed'.

When we print using the print command, the cursor (the point at where the next symbol will appear) remains at the end of whatever we have printed. When we use the println command a linefeed command is issued or in other words the text prints and then the cursor drops down to the next line.

```
Serial.print("Data entered: ");
Serial.println(data);
```

If we look at our two print commands, the first one prints out "Data entered: " and then the cursor remains at the end of that text. The next print command will print 'data', or in other words the contents of the array called 'buffer' and then issue a linefeed, or drop the cursor down to the next line. This means that if we issue another print or println statement after this whatever is printed in the Serial Monitor window will appear on the next line underneath the last.

We then create a new char data type called parameter

```
Char* parameter;
```

and as we are going to use this variable to access elements of the 'data' array it must be the same type, hence the * symbol. You cannot pass data from one data type type variable to another as the data must be converted first. This variable is another example of one that has 'local scope'. It can be 'seen' only by the code within this function. If you try to access the parameter variable outside of the splitString function you will get an error.

We then use a strtok command, which is a very useful command to enable us to manipulate text strings. Strtok gets it's name from String and Token as it's purpose is to split a string using tokens. In our case the token it is looking for is a space or a comma. It is used to split text strings into smaller strings.

We pass the 'data' array to the strtok command as the first argument and the tokens (enclosed within quotes) as the second argument. Hence `parameter = strtok (data, " ,");`

And it splits the string at that point. So we are using it to set 'parameter' to be the part of the string up to a space or a comma.

So, if our text string was

R127 G56 B98

Then after this statement the value of 'parameter' will be R127

as the strtok command would have split the string up to the first occurrence of a space of a comma.

After we have set the variable 'parameter' to the part of the text string we want to strip out (i.e. The bit up to the first space or comma) we then enter a while loop whose condition is that parameter is not empty
(i.e. We haven't reached the end of the string) using

```
  while (parameter != NULL) {
```

Within the loop we call our second function

```
setLED(parameter);
```

Which we will look at later on. Then it sets the variable 'parameter' to the next part of the string up to the next space or comma. We do this by passing to strtok a NULL parameter `parameter = strtok (NULL, " ,");`

This tells the strtok command to carry on where it last left off.

So this whole part of the function

```
char* parameter;
parameter = strtok (data, " ,");
while (parameter != NULL) {
setLED(parameter);
parameter = strtok (NULL, " ,");
}
```

is simply stripping out each part of the text string that is separated by spaces or commas and sending that part of the string to the next function called setLED().

The final part of this function simply fills the buffer array with NULL character, which is done with the /0 symbol and then flushes the Serial data out of the

Serial buffer ready for the next set of data to be entered.

```
  // Clear the text and serial buffers
for (int x=0; x<16; x++) {
buffer[x]='\0';
}
Serial.flush();
```

The setLED function is going to take each part of the text string and set the corresponding LED to the colour we have chosen. So, if the text string we enter is

G125 B55

Then the splitString() function splits that into the two separate components

G125
B55

and send that shortened text string onto the setLED() function, which will read it, decide what LED we have chosen and set it to the corresponding brightness value.

So let's take a look at the second function called setLED().

```
void setLED(char* data) {
if ((data[0] == 'r') || (data[0] == 'R'))
{
int Ans = strtol(data+1, NULL, 10);
Ans = constrain(Ans,0,255);
analogWrite(RedPin, Ans);
Serial.print("Red is set to: ");
Serial.println(Ans);
}
if ((data[0] == 'g') || (data[0] == 'G'))
{
int Ans = strtol(data+1, NULL, 10);
Ans = constrain(Ans,0,255);
analogWrite(GreenPin, Ans);
Serial.print("Green is set to: ");
Serial.println(Ans);
}
if ((data[0] == 'b') || (data[0] == 'B'))
{
int Ans = strtol(data+1, NULL, 10);
Ans = constrain(Ans,0,255);
analogWrite(BluePin, Ans);
Serial.print("Blue is set to: ");
Serial.println(Ans);
}
}
```

We can see that this function contains 3 very similar if statements. We will therefore take a look at just one of them as the other 2 are almost identical.

```
if ((data[0] == 'r') || (data[0] == 'R')) {
int Ans = strtol(data+1, NULL, 10);
Ans = constrain(Ans,0,255);
analogWrite(RedPin, Ans);
Serial.print("Red is set to: ");
Serial.println(Ans);
}
```

The if statement checks that the first character in the string data[0] is either the letter r or R (upper case and lower case characters are totally different as far as C is concerned. We use the logical OR command whose symbol is || to check if the letter is an r OR an R as either will do.

If it is an r or an R then the if statement knows we wish to change the brightness of the Red LED and so the code within executes. First we declare an integer called Ans (which has scope local to the setLED function only) and use the strtol (String to long integer) command to convert the characters after the letter R to an integer. The strtol command takes 3 parameters and these are the string we are passing it, a pointer to the character after the integer (which we don't use as we have already stripped the string using the strtok command and hence pass a NULL character) and then the 'base', which in our case is base 10 as we are using normal decimal numbers (as opposed to binary, octal or hexadecimal which would be base 2, 8 and 16 respectively). So in other words we declare an integer and set it to the value of the text string after the letter R (or the number bit).

Next we use the constrain command to make sure that Ans goes from 0 to 255 and no more. We then carry out an analogWrite command to the red pin and send it the value of Ans. The code then sends out “Red is set to: “ followed by the value of Ans back to the Serial Monitor. The other two if statements do exactly the same but for the Green and the Blue LED’s.

We have covered a lot of ground and a lot of new concepts in this project. To make sure you understand exactly what is going on in this code I am going to set the project code side by side with pseudo-code (an fake computer language that is essentially the computer language translated into a language humans can understand).

## The C Programming Language

```
// Project 10 - Serial controlled RGB Lamp

char buffer[18]; int
red, green, blue; int
RedPin = 11; int
GreenPin = 10; int
BluePin = 9;

void setup()
{
  Serial.begin(9600);
  Serial.flush();
pinMode(RedPin, OUTPUT);
pinMode(GreenPin, OUTPUT);
pinMode(BluePin, OUTPUT);
}

void loop()
{

  if (Serial.available() > 0)
{     int index=0;
    delay(100); // let the buffer fill up
int numChar = Serial.available();     if
(numChar>15) {       numChar=15;
    }
    while (numChar--) {
      buffer[index++] = Serial.read();
    }
    splitString(buffer);
  } }

void splitString(char* data) {
  Serial.print("Data entered: ");
  Serial.println(data);
char* parameter;
  parameter = strtok (data, " ,");
while (parameter != NULL)
{     setLED(parameter);
    parameter = strtok (NULL, " ,");
  }

 // Clear the text and serial buffers
for (int x=0; x<16; x++)
{     buffer[x]='\0';
  }
  Serial.flush(); }
```

*Continued on next page......*

## Pseudo-Code

```
A comment with the project number and name

Declare a character array of 18 letters
Declare 3 integers called red, green and blue
An integer for which pin to use for Red LED
“    “  Green
“    “   Blue

The setup function

Set serial comms to run at 9600 chars per
second
Flush the serial line
Set the red led pin to be an output pin
Same for green
And blue


The main program loop


If data is sent down the serial line...
Declare integer called index and set to 0
Wait 100 millseconds
Set numChar to the incoming data from serial
If numchar is greater than 15 characters...
   Make it 15 and no more

While numChar is not zero (subtract 1 from it)
Set element[index] to value read in (add 1)

Call splitString function and send it data in
buffer


The splitstring function references buffer data
Print “Data entered: “
Print value of data and then drop down a line
Declare char data type parameter
Set it to text up to the first space or comma
While contents of parameter are not empty..
!      Call the setLED function
Set parameter to next part of text string


Another comment
We will do the next line 16 times
Set each element of buffer to NULL (empty)

Flush the serial comms
```

## The C Programming Language

(continued from previous page.......)

```
void setLED(char* data) {
  if ((data[0] == 'r') || (data[0] == 'R'))
{     int Ans = strtol(data+1, NULL, 10);
    Ans = constrain(Ans,0,255);
analogWrite(RedPin, Ans);
    Serial.print("Red is set to: ");
Serial.println(Ans);
  }
  if ((data[0] == 'g') || (data[0] == 'G'))
{     int Ans = strtol(data+1, NULL, 10);
    Ans = constrain(Ans,0,255);
analogWrite(GreenPin, Ans);
    Serial.print("Green is set to: ");
Serial.println(Ans);
  }
  if ((data[0] == 'b') || (data[0] == 'B'))
{     int Ans = strtol(data+1, NULL, 10);
    Ans = constrain(Ans,0,255);
analogWrite(BluePin, Ans);
    Serial.print("Blue is set to: ");
Serial.println(Ans);
  }
}
```

## Pseudo-Code

```
A function called setLED is passed buffer If
first letter is r or R... Set integer Ans to
number in next part of text
Make sure it is between o and 255
Write that value out to the red pin
Print out "Red is set to: "
And then the value of Ans

If first letter is g or G... Set integer Ans
to number in next part of text
Make sure it is between o and 255
Write that value out to the green pin
Print out "Green is set to: "
And then the value of Ans

If first letter is b or B... Set integer Ans
to number in next part of text
Make sure it is between o and 255
Write that value out to the blue pin
Print out "Blue is set to: "
And then the value of Ans
```

Hopefully you can use this 'pseudo-code' to make sure you understand exactly what is going on in this projects code.

We are now going to leave LED's behind for a little while and look at how to control a DC Motor.