# Lab 5 - Serial Parser, String Manipulation & Libraries

At the end of the previous lab you got a taste of adding a parser to your project. A parser is simply a program that looks for (parses) information from a stream of data (in our case strings of data coming from the serial port). In this lab we are going to expand our skills with the parser to allow complex bi-directional communications to occur that will enable us to make a program that executes code on our command as well as tell us about what has just happened using the written word and a serial terminal. The general term in computer science for extracting information from a more complex set of data is called a parser and indeed to parse means to extract information from. Are you ready to learn about parsers? Then don't miss this lab!

## Part 1 – Software Reset

NOTE: This functionality only works on chipKIT boards. Arduino boards do not support software reset in this manner. There are workarounds that are available for Arduino but at this writing there is no standardized support in the Arduino wiring libraries. If you are attempting this class with an Arduino borrow a board to evaluate and get this section of the lab checked off.

In a previous lab we looked at the functionality of the reset button. As a reminder, when the reset button is pressed on the Fubarino the PIC32 chip is “reset” to its power on state and starts running the bootloader program. On a Fubarino, and other chipKIT board that have a program input, the bootloader looks at the state of this input. If the input is asserted the bootloader enters a mode where it waits for a new program to be sent over the serial port to be programmed in to the memory of the chip. If the program input is not asserted, then the bootloader tries to run a previously loaded program.

The chipKIT-core library includes a function called executeSoftReset that allows us, in software, to reset the board. The function takes a single parameter that allows us to choose the functionality after reset. Either we can enter the bootloader to accept a new program or we can run a sketch that was previously programmed. The two ways to call the function are shown below:

```
// Run previously loaded sketch after executing the soft reset
executeSoftReset(RUN_SKETCH_ON_BOOT);
// Enter the bootloader after executing the soft reset
executeSoftReset(ENTER_BOOTLOADER_ON_BOOT);
```

Some chipKIT boards that have an external USB chip and typically will not support the ENTER_BOOTLOADER_ON_BOOT option. If this is the case the function will return a false value and your program will continue execution.

Hook a pullup resistor up to pin 22 of your board and try this program:

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(22, INPUT);
  digitalWrite(LED_BUILTIN, HIGH);
  delay(2000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(2000);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(300);
  digitalWrite(LED_BUILTIN, LOW);
  delay(50);
  if(digitalRead(22) == LOW) {
    executeSoftReset(RUN_SKETCH_ON_BOOT);
  }
}
```

If you pull the input low with a switch this program reset the board! Modify the program to read a second input on pin 12 that allows the board to enter the bootloader. Hint: don't forget to set the pin mode for pin 12 and just add a second if statement below the first to check pin 12.

## Check off

When both pin 12 and pin 22 will reset the board, call over the instructor to demonstrate your work.

## Part 2 - Parse, Execute, Respond and Reset

From now on, most labs will be building on functionality of previous labs. From time to time a simple example will be shown to illustrate a point in isolation for clarity.

***<u>Start off by opening the code from the last part of the previous lab</u>*** and save it with a new name for this lab (maybe lab5_part2 might be a good name).

At the end of the setup() add a 3 second delay. If you don't remember how to use the delay function, refer to lab 2. After the delay transmit the string "Command Parser" out the serial port so that we know the program is starting.

Note: This function is specific to the bootloader on the chipKIT boards and **<u>will not work on Arudino</u>**.

Create a function called **reset()** that takes no parameters and returns a void type to the end of your program:

```
void reset(void) {
  Serial.println("Close serial terminal, resetting board in...");
  uint8_t sec = 5;
  while (sec >= 1)
  {
    Serial.print(sec, DEC);
    Serial.println(" seconds...");
    delay(1000);
    sec--;
  }
  executeSoftReset(ENTER_BOOTLOADER_ON_BOOT);
}
```

Add this code to the parser in the loop() before the last else:

```
    else if (strcmp(command, "reset") == 0) {
      reset();
    }
```

Create a new function called blink() that is called when the command "**blink**" is received by the parser. The blink() should turn on the LED_BUILTIN on and off four times using a while loop.

Parse the "**blink"** command in loop() utilizing the structure outlined above for the reset command.

Test the code by running the program and typing the **blink** and **reset** commands into the terminal.

### Analysis

The reset function accomplishes the same task as holding the PRG button while pushing the RESET button on the Fubarino. It puts the board into the bootloader so you can load a new program, without physically pushing the buttons.

### Check off

Call over the instructor to check your work.

Be prepared to explain why there is a countdown before calling the executeSoftReset function

## Part 3 - More commands

In this section add and test each command before progressing to the next.

Add a command to the board called **bye** that prints out "leaving so soon?" when entered.

Add a command called **toggle** that toggles the state of LED_BUILTIN each time it is run.

Command **lightshow** will make the LEDs flash a pattern, like that we did in Lab 2, for a few seconds.

Command **help** or **?** prints outs a list of serial commands the board recognizes.

### Check off

Call over the instructor to check your work. Be prepared to demonstrate all commands.

## Part 4 - Buffer Size Modification

In this section, you test each section independently and try to understand how the changes are affecting the program operation.

**Buffer Size**

> Decrease the size of the command[] to 5. Try sending some commands to the board. Does this affect program functionality, and if so how?

### Check off

Call the instructor over to discuss this and to get checked off.

Reset command[] size back to 0x20 and continue to the next section.

## Part 5 - Count Up / Count Down

Add the command **count up** to the parser. The **count up** command should use a loop to print out the numbers 1 to 15 on each <u>on their own line</u> with a quarter second delay between each number. Also display the number on the connected LED's.

Add the command **count down** to the parser. The **count down** command should use a loop to print out the numbers 15 down to 0 <u>on a single line with a space between each number</u> with a quarter second delay between each number. Also display the output on LED's.

### Check off

Call the instructor over when both commands are working.

## Part 6 – parse s

In this section add and test each command before progressing to the next.

In this section we will add a command called **parse s** that will iterate through the command buffer and display the character in each position in the array.

```
void parse(char* s) {
  uint8_t i = 0;

  while(i < strlen(s) ) {
    Serial.print("s[");
    Serial.print(i, DEC);
    Serial.print("]='");
    Serial.print(s[i]);
    Serial.print("' (0x");
    Serial.print((uint8_t)s[i], HEX);
    Serial.print(")");
    Serial.println("");
    i++;
  }
}
```

To use call this function from our parser we will need to utilize the strncmp() to **search for a match in the first n characters of the string**. This is accomplished by adding the following to your command parser:

```
    else if (strncmp(command, "parse", 5) == 0) {
      parse(command);
    }
```

### Analysis

strncmp(char* string1, char* string2, int n) which take three parameters is different than

strcmp(char* string1, char* string2) which takes two parameters.

string1 is the string to search, string2 is the string we are looking for and n is the number of character to try and match.

int strlen(char string) returns the number of characters in string (not counting the null character)

The parse() accepts a pointer to a string named s as its only parameter. We utilize the strlen() to look for length of the string and iterate through all indices in the array. We then print out the index, character and ASCII value for each index in the array.

### Check off

Call over the instructor to check your work. Be prepared to demonstrate all commands.

## Part 7 – flip s

Command **flip s** where s is a string, returns a flip cased string of all characters received after flip. So if the string “flip hEllO” is sent to the board the string “HeLLo” will be returned. **NOTE: This should only print out “HeLLo” and not “FLIP HeLLo”**

### Check off

Call over the instructor to check your work. Be prepared to demonstrate all commands.

## Part 8 - LED State Machine

Use parser to control an array of LED patterns. Command **forward** shows the next pattern and **previous** shows the last pattern, if you are at the first position do nothing when the **previous** command is entered. When you are at the last position do nothing when the **forward** command is entered. Hint: You will need to create a variable to keep track of the position in the LED pattern sequence. Also you will need to test for array bounds so that you don’t exceed the positions in the array.

```
uint8_t pattern[] = {
          0b000000001,
          0b000000011,
          0b000000111,
          0b000001111,
          0b000001110,
          0b000001100,
          0b000001000,
          0b000000000
    }
uint8_t pattern_position = 0;
```

### Check off

Call the instructor over when working.

## Part 9 - Pointers to uint8_t

Create a new sketch (do not add this to the previous sketch) called lab5-bonus and try this mini program.

```
uint8_t pattern1 = 0b0110;
uint8_t pattern2 = 0b1001;

void setup()
{
  nibbleInit(0, 1, 2, 3, OUTPUT);
}

void loop()
{
  nibbleToPins(pattern1, 0, 1, 2, 3);
  swapchars(&pattern1, &pattern2);
  delay(200);
}

void swapchars(uint8_t* a_ptr, uint8_t* b_ptr)
{
  uint8_t temp;
  temp = *a_ptr;
  *a_ptr = *b_ptr;
  *b_ptr = temp;
}
```

Spend some time to understanding each line of code.

### Check off

Call the instructor over to demonstrate a working program and explain how the swapchars function works.

## Part 10 - Pointers to us8 Modified

Write a program that uses pointers to switch between 3 patterns.

### Check off

Call the instructor over to demonstrate when working.

## Homework – Extra Credit

Receive one lab credit (1/10 of a lab) extra credit for each of the two items completed.

### LED State Machine - Advanced

Change the program so that the when at the last position the **forward** command cycles back to the first value and when you are at the first position the **previous** command cycles back to the last value.

### Adventure

Use a parser to write a little adventure game. You can print out the response through the terminal, or using buttons and LEDs on your board. Keep track of a 10x10 space using an array. You can navigate through the space using commands **north**, **south**, **east** and **west**. Use the LED's to indicate items present in the space, LED0 indicates treasure, LED1 indicates a weapon, LED2 indicates food, LED3 indicates monster. As you move thought the space use the LED's and serial terminal to indicate what is present in the room.