Advanced Topics: Pointers

Pointers offer a convenient way to handle variables. Some say pointers give the programmer a lot of power. With great power...

This is an advanced topic, so don’t worry about understanding it right away.

Suggested Prerequisites

• Programming Intro
• Conditional Statements
• Loops
• Functions

Setup and Preparation

List of Supplies

• TI MSP430G2 LaunchPad
• USB cable
• Your mind

Background: What is a pointer?

Every variable you create has to be stored somewhere, somewhere in the MSP430G2553 chip’s memory. And more importantly, the computer needs a way to keep track of where these variables are stored. It does so by keeping track of the variable’s memory address. So what do these memory addresses look like to us? Let’s have a look.
Program 1: Observing the memory address

Create and upload the following program (Figure 1) to the LaunchPad.

```cpp
void setup() {
    Serial.begin(9600);

    byte value = 25;
    Serial.print(value);
    Serial.print(" is stored at address: ");
    Serial.println((long)&value); // & = "the address of"
    Serial.println();
}

void loop() {
}
```

**Figure 1: Displaying Variable Addresses**

**Discussion**

- Using the Serial Monitor, you can see that the number “25” is stored at some number. This is the memory address of the variable value. If we tore open the chip and examined the bits at this address, we would read the number 25.
- As we can see, the memory address is just another number. Which begs the question:
  - Can we store the memory address of value in another variable?
    - Yes, and we call that variable a pointer to value because it contains a memory address that refers (or “points”) to the number stored at value.
Create and upload the following program (Figure 2) to the LaunchPad.

```cpp
void setup() {
    Serial.begin(9600);

    byte num = 25;
    byte *numPtr = &num; // & = "the address of"
    Serial.print(*numPtr); // * = "the value stored at"
    Serial.print(" is stored at address: ");
    Serial.print((long) numPtr);
    Serial.println();

    *numPtr = 7; // "Change 'the value stored at' 'the address of' num to 7"
    Serial.print("New value of 'num' is: ");
    Serial.print(num);
    Serial.println();
}

void loop() {
}
```

**Figure 2: Creating and Using Pointers**

**Things to consider**

- Use the *Serial Monitor* to observe the output of this program.
- Try to determine the difference between `num`, `&num`, `numPtr`, and `*numPtr`.
  - Consider drawing a picture of how you think things look in memory.
- In the last section of code, notice how we can change the value of the original variable using its pointer.
- In the next example, we will look at how we can apply pointers.
Program 3: A situation that could use a pointer...

Create and upload the following program (Figure 3) to the LaunchPad.

```java
void setup() {
  pinMode(RED_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);
}

boolean pin = RED_LED;

void loop() {
  pin = getOtherPin(pin); // What happens here?
  blinkLED(pin);
}

int getOtherPin(int pinNumber) {
  if (pinNumber == RED_LED) {
    return GREEN_LED;
  } else {
    return RED_LED;
  }
}

void blinkLED(int pinNumber) {
  digitalWrite(pinNumber, HIGH);
  delay(500);
  digitalWrite(pinNumber, LOW);
  delay(500);
}
```

Figure 3: Setting the Scene

Things to consider

- Observe what this program does, and then try to explain how this program works.
- In particular, what is the purpose of the following line in the program?
  ```java
  pin = getOtherPin(pin);
  ```
  - If you are not sure, use the Serial Monitor to observe how pin changes if.
- This program can be more elegantly implemented using pointers in the next example.
Program 4: Applying a Pointer

Create and upload the following program (Figure 4) to the LaunchPad.

```c
void setup() {
  pinMode(RED_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);
}

boolean pin = RED_LED;

void loop() {
  switchPin(&pin); // &pin = "pointer to pin"
  blinkLED(pin);
}

void switchPin(boolean *pinPointer) { // *pinPointer = "the value at which
  if (*pinPointer == RED_LED) { // pinPointer is pointing"
    *pinPointer = GREEN_LED;
  } else {
    *pinPointer = RED_LED;
  }
}

void blinkLED(int pinNumber) {
  digitalWrite(pinNumber, HIGH);
  delay(500);
  digitalWrite(pinNumber, LOW);
  delay(500);
}
```

Figure 4: Applying a Pointer

Things to consider

- Try to explain how this program is working. Use the Serial Monitor to examine the values of pin in the loop() function.
- Somehow, the switchPin() function is changing the value of pin. You could probably guess that pointers are somehow making this work.
  - Try removing all the * and & characters and see if it still works properly.
- Try drawing a picture of what you think is happening.
Program 5: References of Elements in an Array

Create and upload the following program (Figure 5) to the LaunchPad.

```java
void setup() {
  Serial.begin(9600);

  int arrayLength = 5;
  byte values[] = {5, 15, 25, 35, 45};
  for (int i = 0; i < arrayLength; i++) {
    Serial.print(values[i]);
    Serial.print(" is stored at address: ");
    Serial.println((long *)&values[i]);
  }
}

void loop() {
}
```

Discussion

- This program is identical to program 1, except now we are looking at the addresses of several elements, all from the same array.
- What?! Is there a pattern in the memory addresses of these elements?! Take a look.
- This discovery begs another critical question:
  - If the memory address is just a number, and we can see an obvious mathematical pattern in the elements’ addresses, can we simply use the first element’s address to calculate the addresses of the other element?
  - Let’s look at Program 6 to find out.
Create and upload the following program (Figure 6) to the LaunchPad.

```cpp
void setup()
{
  Serial.begin(9600);

  int arrayLength = 5;
  byte values[] = {5, 15, 25, 35, 45};
  byte *ptrToFirstElement = &values[0];
  for (int i = 0; i < arrayLength; i++) {
    Serial.print(*ptrToFirstElement + i);
    Serial.print(" is stored at address: ");
    Serial.println((long) &values[i]);
  }
}

void loop() {}
```

Discussion

- Note that `ptrToFirstElement` contains a reference to the first element of the array.
- We can simply add a number to it, and calculate the address of another element in the array.
- Note that we use the `*` after we have calculated the new address.
  - Recall that you can read `*` as “the value stored at”
- How high does the loop variable `i` go?
Review

After this lab, you should have a good understanding of the following topics. If you’re not sure about some of them, go back through the lab and try to find a good place to explore the topic.

You should be able to do the following

- Draw a diagram of the LaunchPad’s internal memory and explain how pointers work.
- Get the memory address of a variable using the & operator.
- Store memory addresses into pointer variables, declared with the * operator.
- Use the variable stored at a pointer, using the * operator.
- Use a pointer to change the value of the variable it references.