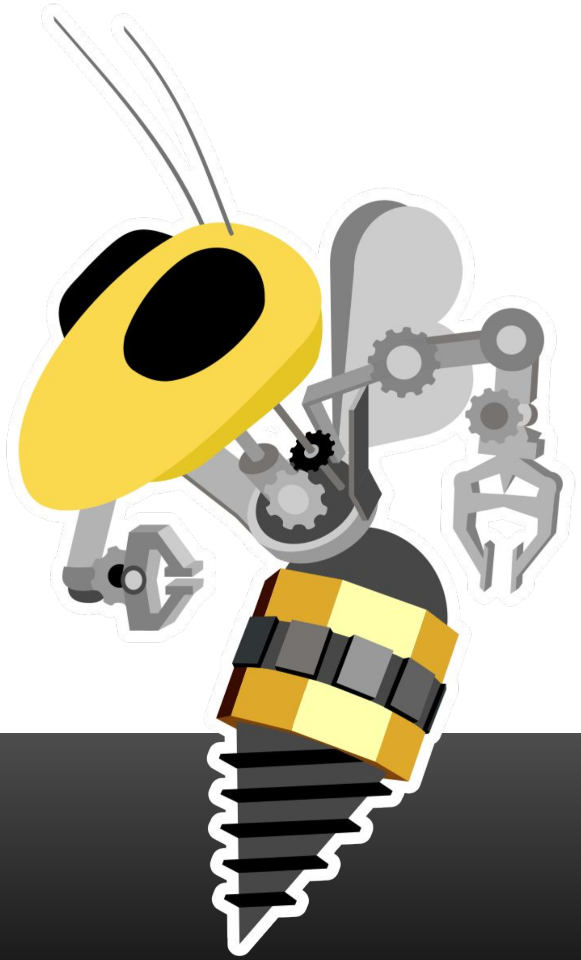
Welcome!

Firmware Training
Week 2



RoboJackets

Competitive Robotics at Georgia Tech

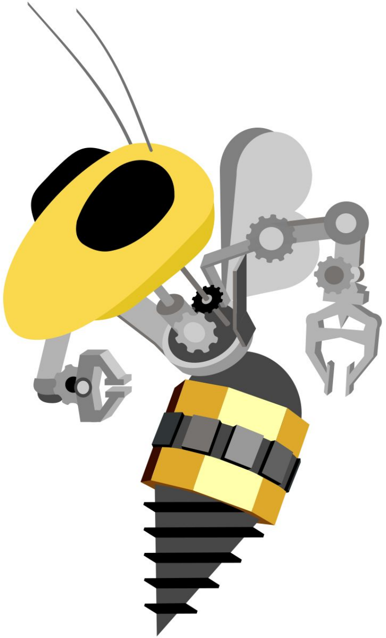
www.robojackets.org

Last Week!

- Microcontrollers
- C++, Part 1
    - Variables in C++
    - Arithmetic & Making Comparisons
    - If / Else
- Prototyping

This Week!

- Memory
- C++, Part 2
  - Functions
  - For and While Loops
  - Arrays and Structs
- Interrupts



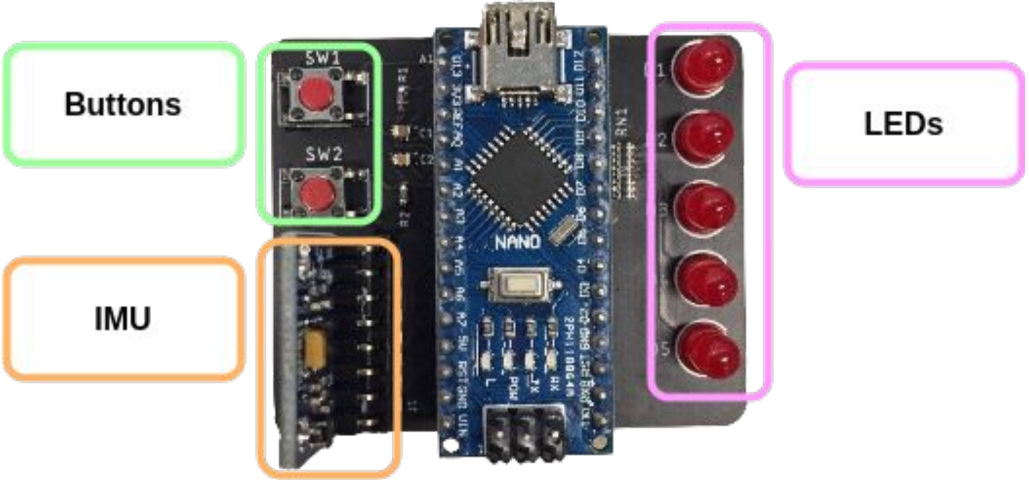
Firmware Training Board

What is this thing?

ROBOJACKETS
COMPETITIVE ROBOTICS AT GEORGIA TECH

Firmware Training Board

No more breadboards

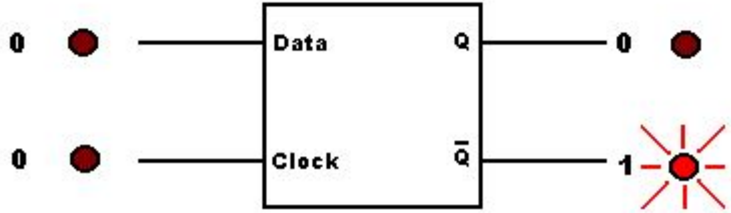


Memory

How do computers remember?

Meet the Register!

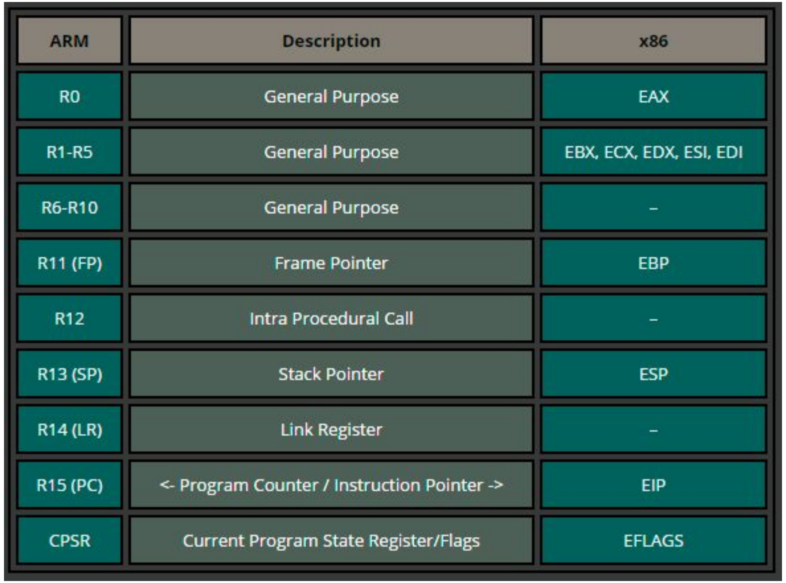
- Register: small, fast memory!
  - Pictured: D flip-flop
  - Dependent on clock
  - Volatile memory: forgets when power is removed



The Register File (Regfile)

- The CPU has its own set of registers called the **Regfile**

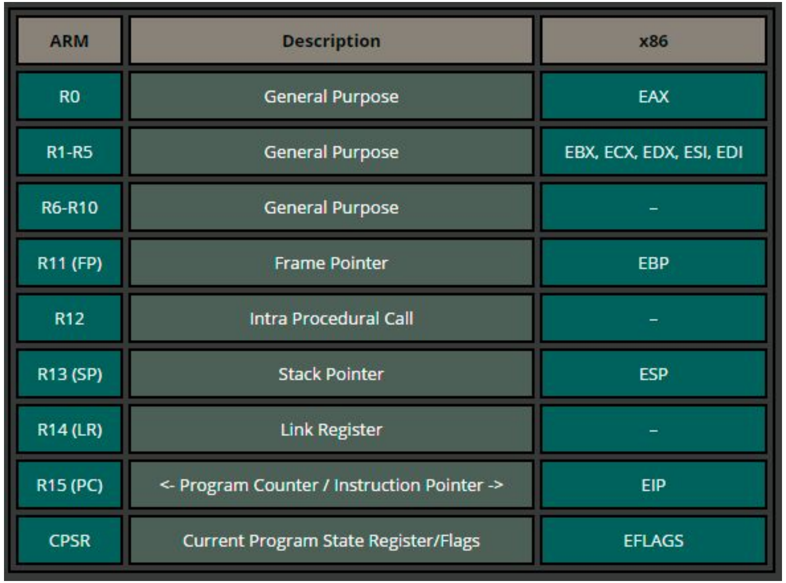
- General Purpose handles normal data, such as:
  - Data from RAM
  - Data from instructions
  - Memory addresses



ARM	Description	x86
R0	General Purpose	EAX
R1-R5	General Purpose	EBX, ECX, EDX, ESI, EDI
R6-R10	General Purpose	–
R11 (FP)	Frame Pointer	EBP
R12	Intra Procedural Call	–
R13 (SP)	Stack Pointer	ESP
R14 (LR)	Link Register	–
R15 (PC)	<- Program Counter / Instruction Pointer ->	EIP
CPSR	Current Program State Register/Flags	EFLAGS

The Register File (Regfile)

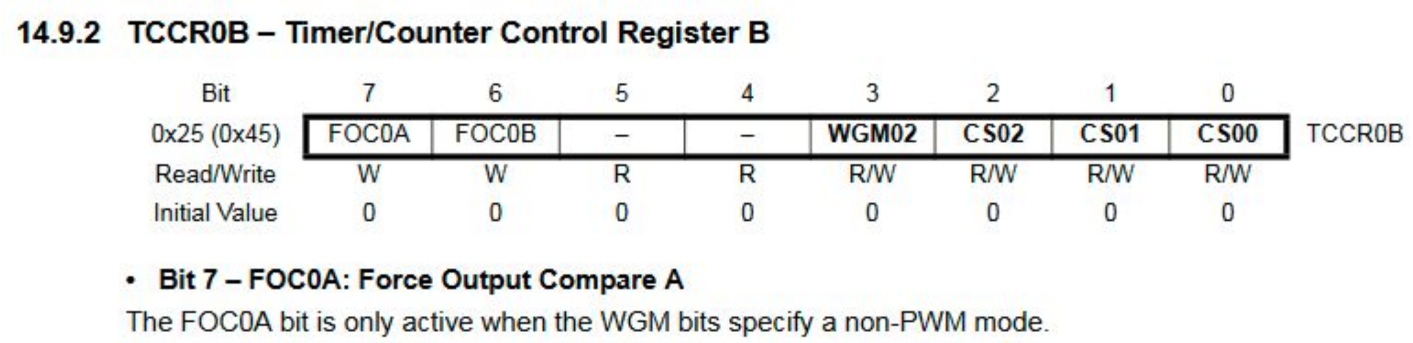
- Stack Pointer: points to the top of the **stack**
- Frame pointer: helps manage the **stack**
- Program Counter (PC) or Instruction Pointer (IP): points to the next instruction



ARM	Description	x86
R0	General Purpose	EAX
R1-R5	General Purpose	EBX, ECX, EDX, ESI, EDI
R6-R10	General Purpose	–
R11 (FP)	Frame Pointer	EBP
R12	Intra Procedural Call	–
R13 (SP)	Stack Pointer	ESP
R14 (LR)	Link Register	–
R15 (PC)	<- Program Counter / Instruction Pointer ->	EIP
CPSR	Current Program State Register/Flags	EFLAGS

Hardware Registers

- Registers outside the CPU, used especially in MCUs
- Controls peripheral devices
- Week 3: Playing with hardware registers!



14.9.2 TCCR0B – Timer/Counter Control Register B

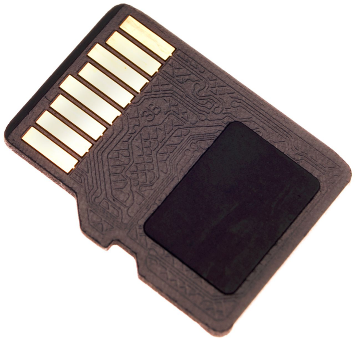
Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	**WGM02**	**CS02**	**CS01**	**CS00**	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOC0A: Force Output Compare A**

The FOC0A bit is only active when the WGM bits specify a non-PWM mode.

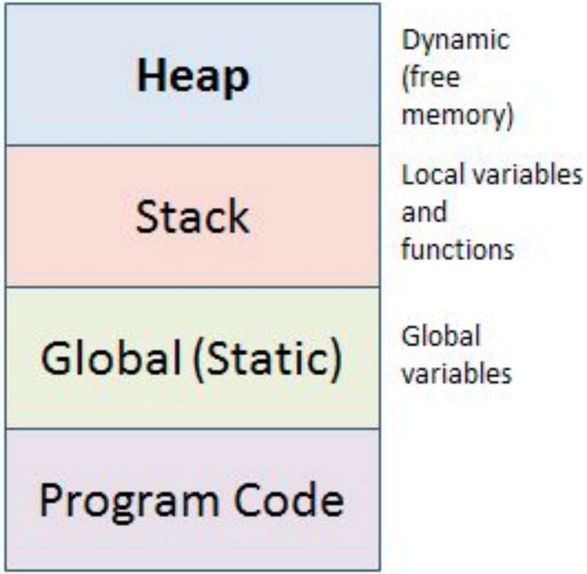
Now, meet EEPROM!

- EEPROM: Electrically Erasable Programmable Read-Only Memory
  - Flash memory is a type of EEPROM (bigger blocks)
  - Non-volatile: it remembers even when power is lost!

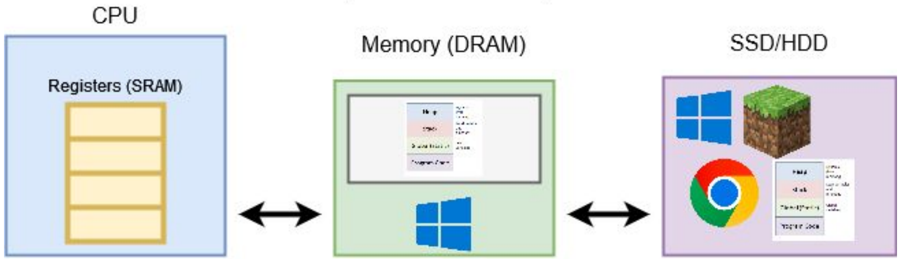


C++ Memory Layout

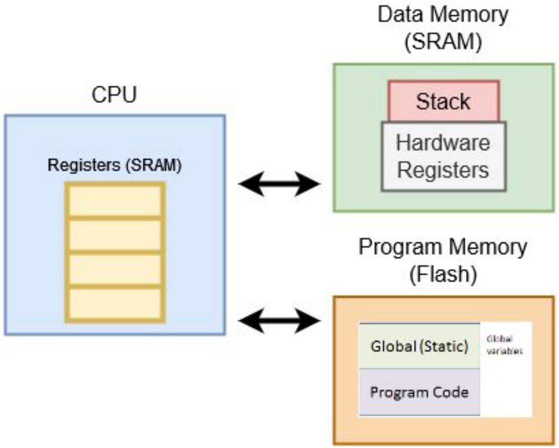
- Program Code
  - Where your program goes
  - Library functions get copied in, too
- Global variables
  - Data declared outside of functions goes here
- (Call) Stack
  - Initially empty
  - Stores information about active subroutines of a computer program
- The Heap / Free Memory / Dynamic Allocation
  - Memory that is allocated at runtime (i.e. dynamically)



Typical PC



Typical MCU

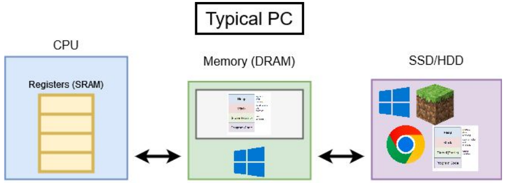


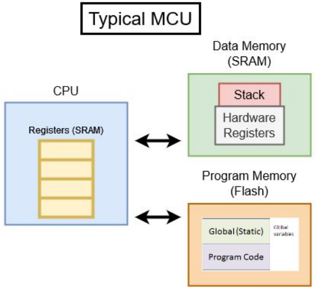
Similarities

- Address Space looks the same!

Differences

- Architecture
  - PC: Von Neumann
  - MCU: Harvard
- Memory size
  - PC deals with much bigger programs, needs bigger memory.
  - MCU can get away with small but fast SRAM for data.
- Dynamic Memory / Operating System
  - On PC, memory is managed by the operating system. Processes have their own virtual address space.
  - MCU often has no operating system. Dynamic memory allocation frowned upon, some platforms do not natively support it.





C++, Part 2

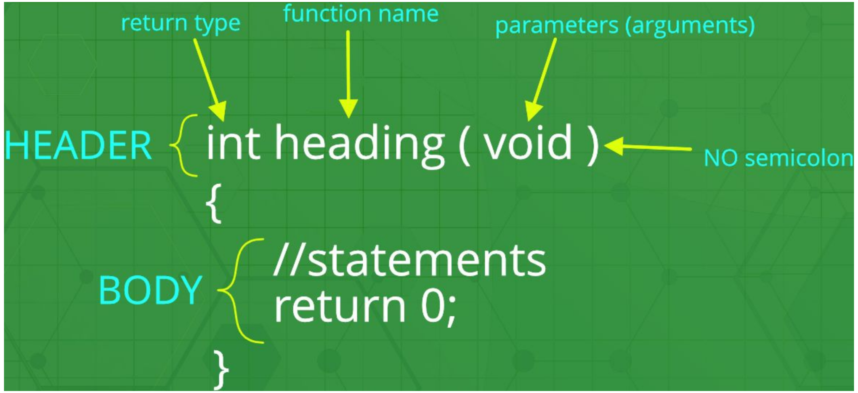
See Plus Plus

https://github.com/RoboJackets/firmware-training

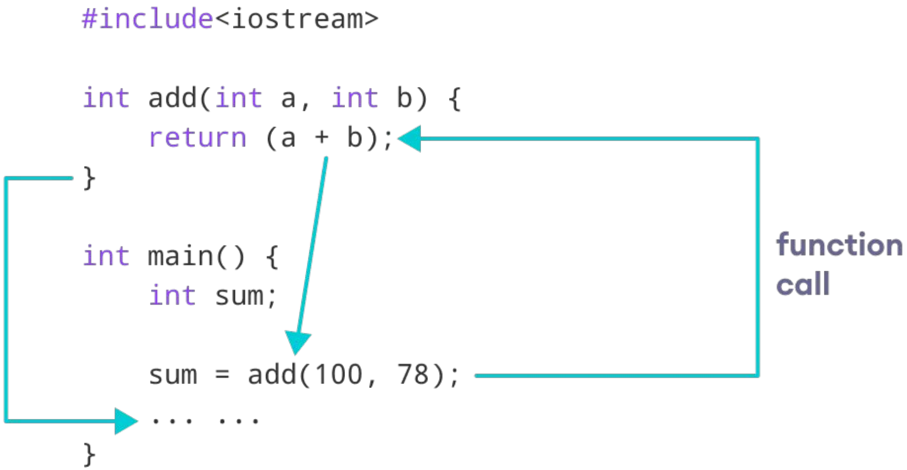
Go to “Binder Link” at bottom of page!

Functions

- Helps to organize code into chunks
- Makes it easier to read and prevents duplicated code
- Define before using it in your code



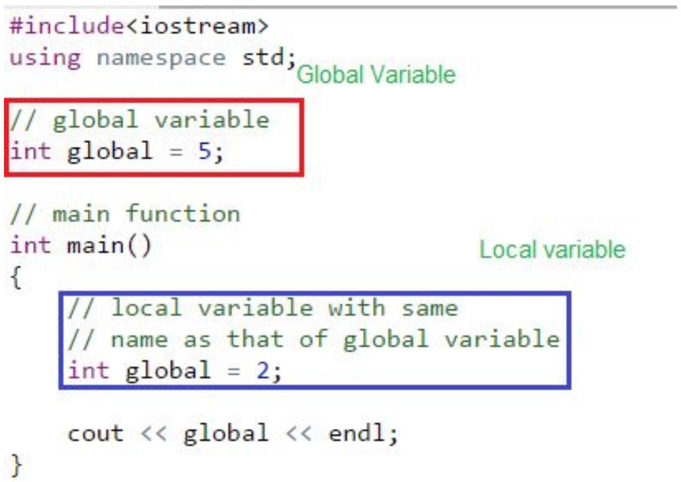
```
int heading ( void )
{
    //statements
    return 0;
}
```



```
#include<iostream>

int add(int a, int b) {
    return (a + b);
}

int main() {
    int sum;

    sum = add(100, 78);
    ... ...
}
```

Variable Scope

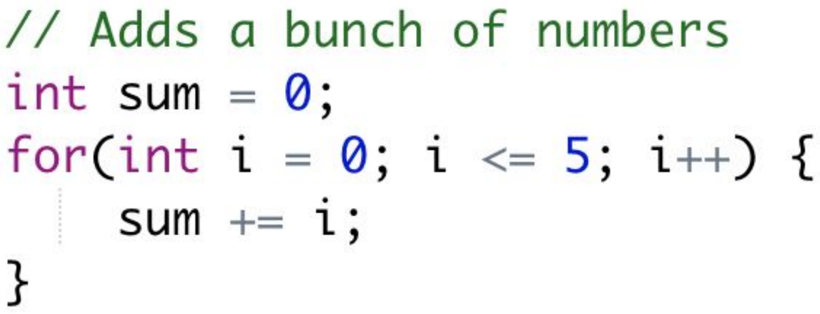
- Local variables
  - Within functions
  - Can’t be accessed elsewhere
- Global variables
  - What you used before (before setup)
  - Accessible everywhere in the file
- Volatile
  - Variables used in interrupts
  - Compilers check for variables that are not either assigned to or read from (dead code)
  - Since ISRs are things called by hardware and not the code the compiler doesn't understand it's still affecting the value of a variable
  - Marking the variable as volatile tells the compiler not to remove operations to a variable as it's changing in ways you can't see



```
#include<iostream>
using namespace std;

// global variable
int global = 5;

// main function
int main()
{
    // local variable with same
    // name as that of global variable
    int global = 2;

    cout << global << endl;
}
```

For Loops

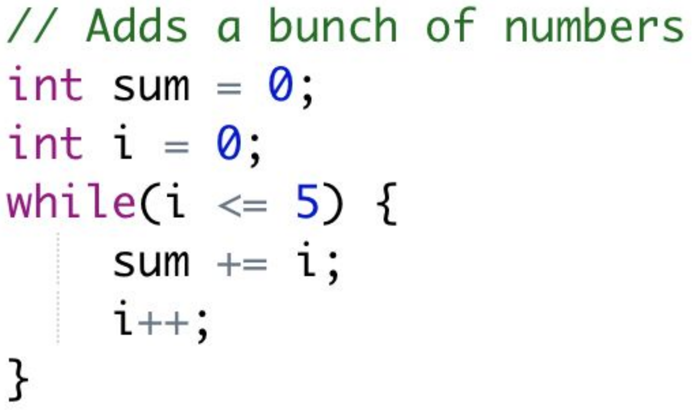
- Designed to repeat code a fixed number of times
- Three part syntax
    - Initialize counter
    - Bounds check
    - Increment counter



```
// Adds a bunch of numbers
int sum = 0;
for(int i = 0; i <= 5; i++) {
    sum += i;
}
```

While Loops

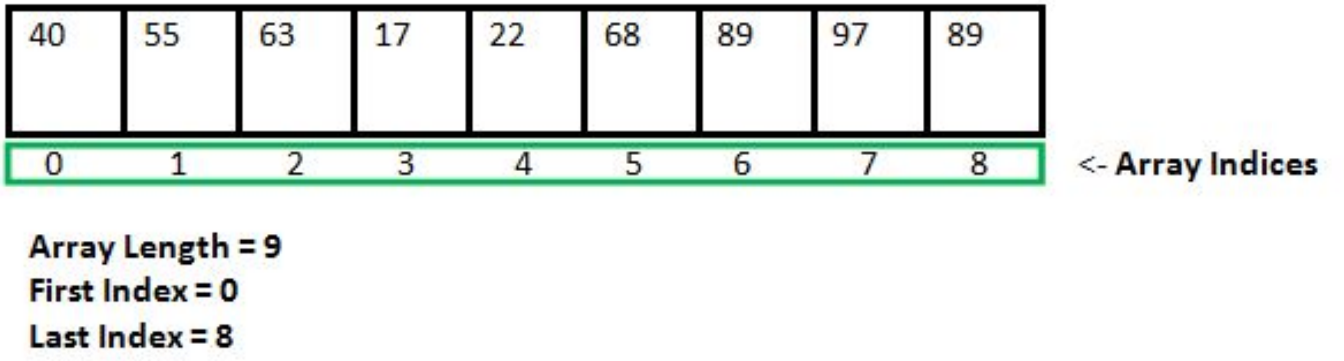
- Designed to repeat code until condition is met
- Three part syntax
  - Initialize counter
  - Condition check
  - Increment counter



```
// Adds a bunch of numbers
int sum = 0;
int i = 0;
while(i <= 5) {
    sum += i;
    i++;
}
```

Arrays

- Way to organize data (collection of same data type)
- Has a fixed size at creation
- Access using an index

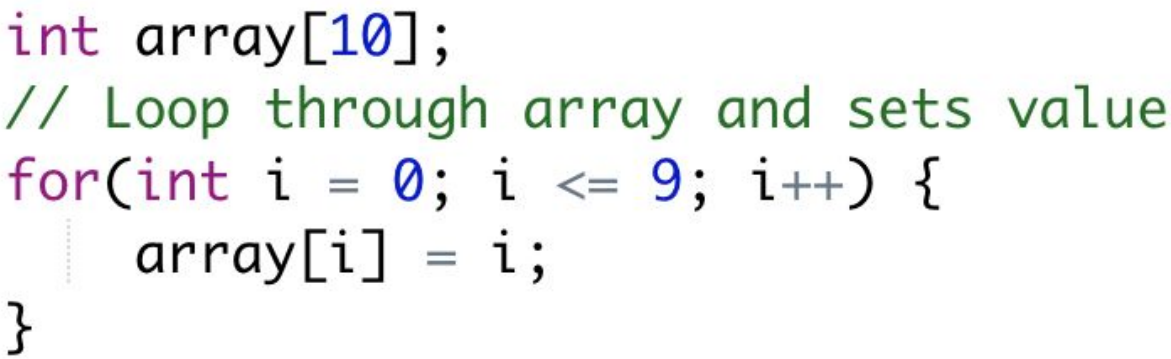


40	55	63	17	22	68	89	97	89	
0	1	2	3	4	5	6	7	8	<- **Array Indices**

Array Length = 9
First Index = 0
Last Index = 8

Using Arrays

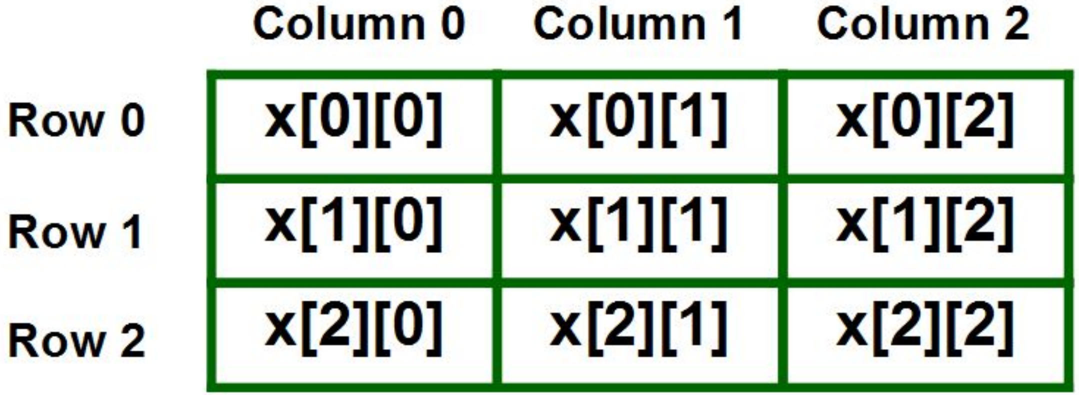
- Create with *type name[size]* syntax
- Read and set data with *name[index]* syntax
    - You should always set before you read



```
int array[10];
// Loop through array and sets value
for(int i = 0; i <= 9; i++) {
    array[i] = i;
}
```

2D Arrays

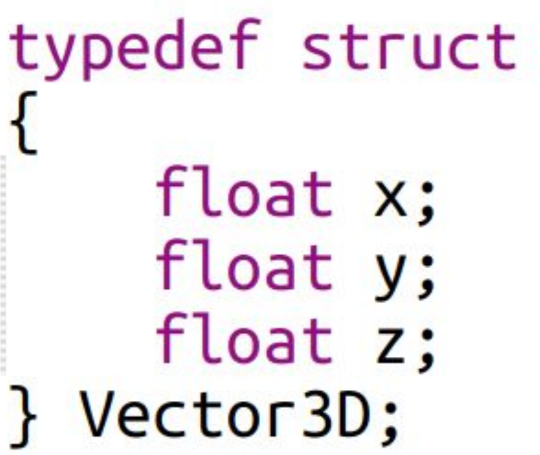
- Make an array of arrays (multiple dimensions)
- Created by adding another dimension *[size1][size2]*



	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

Structs

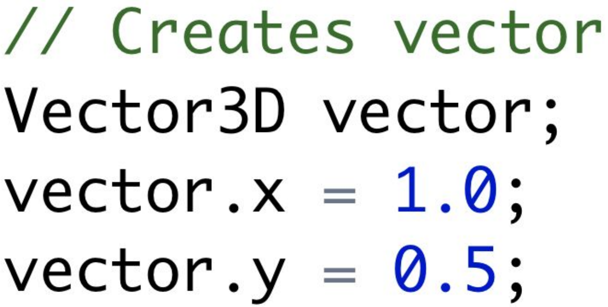
- Ways to organize data of same or different types
  - A vector has multiple different quantities
- Commonly used to package related data



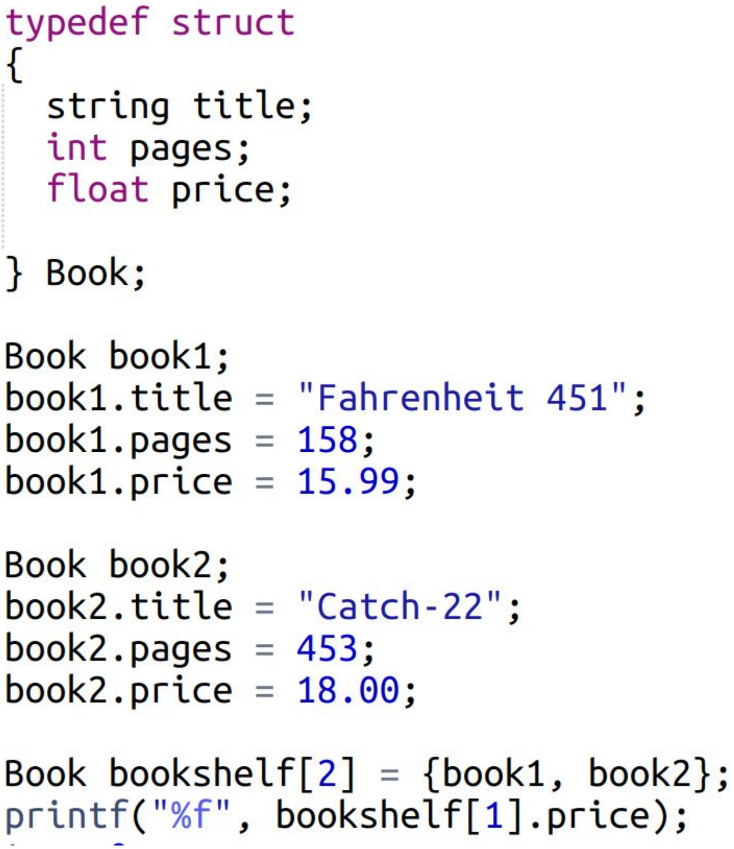
```
typedef struct
{
    float x;
    float y;
    float z;
} Vector3D;
```

Using Structs

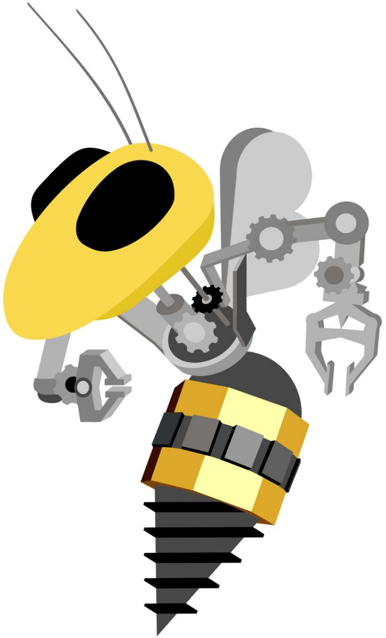
- Create as you would a variable, *type name* syntax
- Read and set using *name.field* syntax
  - You should always set before you read



```
// Creates vector
Vector3D vector;
vector.x = 1.0;
vector.y = 0.5;
```



```
typedef struct
{
  string title;
  int pages;
  float price;

} Book;

Book book1;
book1.title = "Fahrenheit 451";
book1.pages = 158;
book1.price = 15.99;

Book book2;
book2.title = "Catch-22";
book2.pages = 453;
book2.price = 18.00;

Book bookshelf[2] = {book1, book2};
printf("%f", bookshelf[1].price);
```



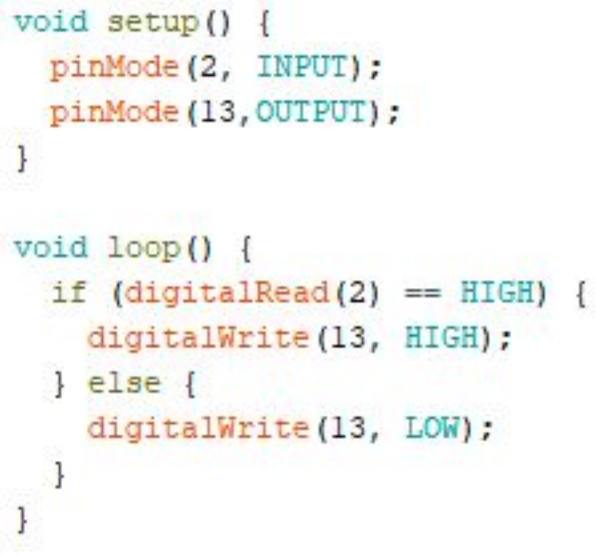
Interrupts

Hey Mom. Mom. Mom. Mom. Mo- WHAT!

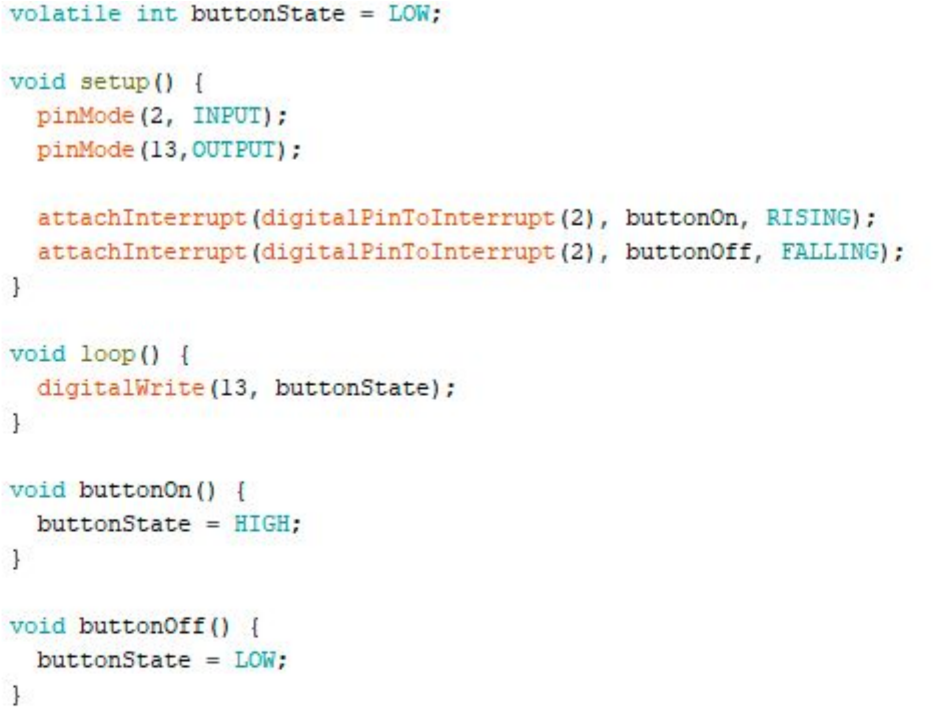
What are Interrupts

- Mechanism built into processors to run an **interrupt handler / interrupt service routine** when an event occurs
    - Can be hardware (a pin) or software (a timer)
- It stops (interrupts) the main code before returning back to the main code

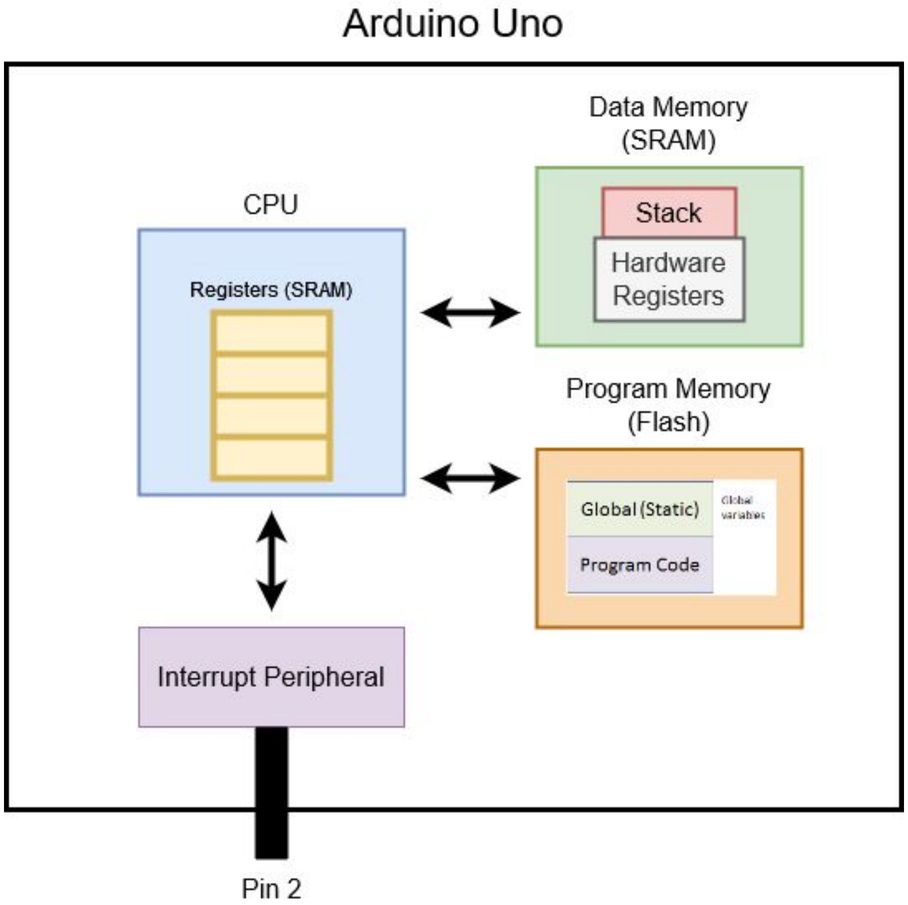
Polling vs Interrupts



```
void setup() {
  pinMode(2, INPUT);
  pinMode(13,OUTPUT);
}

void loop() {
  if (digitalRead(2) == HIGH) {
    digitalWrite(13, HIGH);
  } else {
    digitalWrite(13, LOW);
  }
}
```

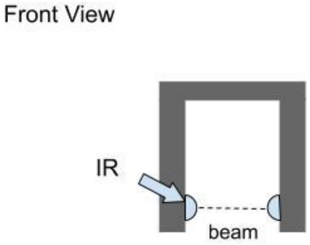


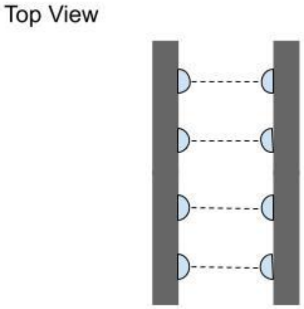
```
volatile int buttonState = LOW;

void setup() {
  pinMode(2, INPUT);
  pinMode(13,OUTPUT);

  attachInterrupt(digitalPinToInterrupt(2), buttonOn, RISING);
  attachInterrupt(digitalPinToInterrupt(2), buttonOff, FALLING);
}

void loop() {
  digitalWrite(13, buttonState);
}

void buttonOn() {
  buttonState = HIGH;
}

void buttonOff() {
  buttonState = LOW;
}
```


Arduino Uno
CPU
Registers (SRAM)
Data Memory
(SRAM)
Stack
Hardware
Registers
Program Memory
(Flash)
Global (Static)
Global
variables
Program Code
Interrupt Peripheral
Pin 2

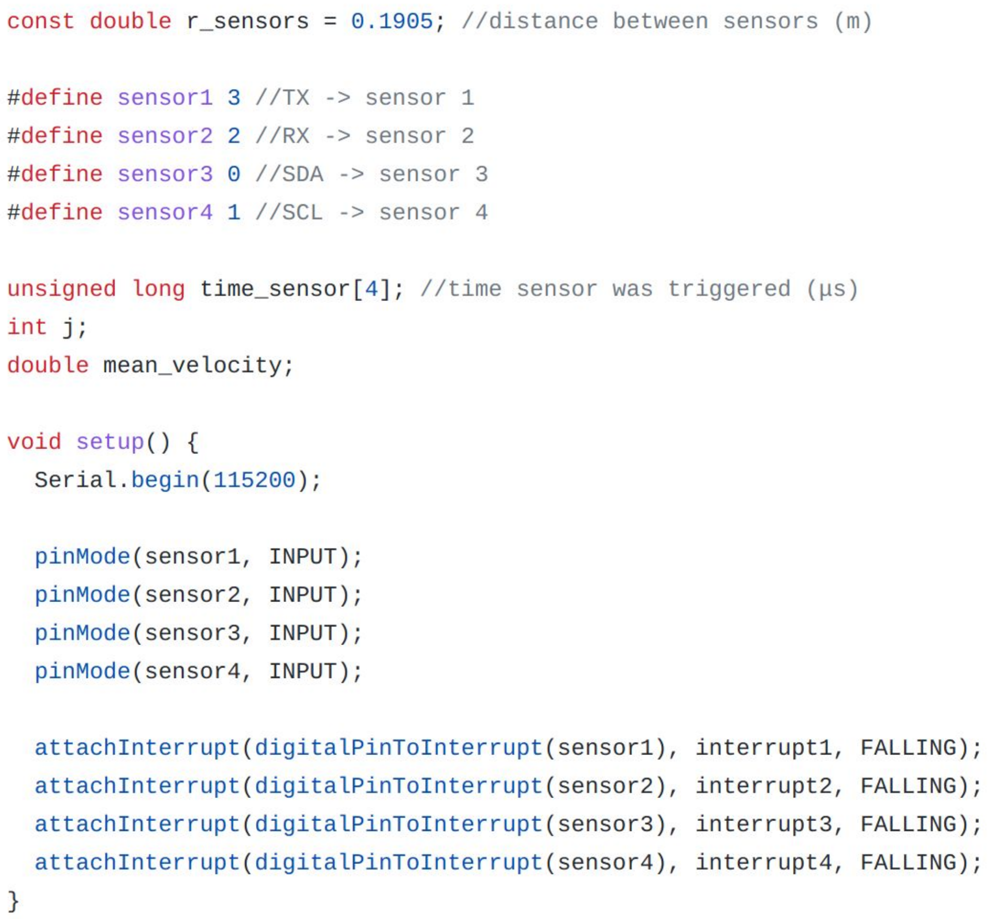
RoboCup Ball Speed Example

- Want to calculate the speed of the ball after the robot kicks ball
- IR sensors determine when the ball has passed
  - Using the time this occurs and distance between sensors can calculate average speed



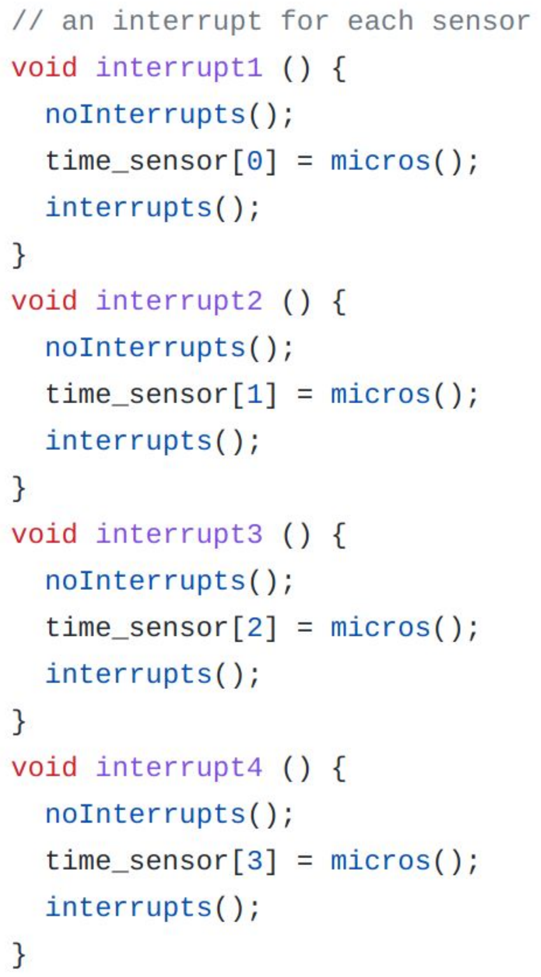


Code: Interrupt Setup



```
const double r_sensors = 0.1905; //distance between sensors (m)

#define sensor1 3 //TX -> sensor 1
#define sensor2 2 //RX -> sensor 2
#define sensor3 0 //SDA -> sensor 3
#define sensor4 1 //SCL -> sensor 4

unsigned long time_sensor[4]; //time sensor was triggered (µs)
int j;
double mean_velocity;

void setup() {
  Serial.begin(115200);

  pinMode(sensor1, INPUT);
  pinMode(sensor2, INPUT);
  pinMode(sensor3, INPUT);
  pinMode(sensor4, INPUT);

  attachInterrupt(digitalPinToInterrupt(sensor1), interrupt1, FALLING);
  attachInterrupt(digitalPinToInterrupt(sensor2), interrupt2, FALLING);
  attachInterrupt(digitalPinToInterrupt(sensor3), interrupt3, FALLING);
  attachInterrupt(digitalPinToInterrupt(sensor4), interrupt4, FALLING);
}
```

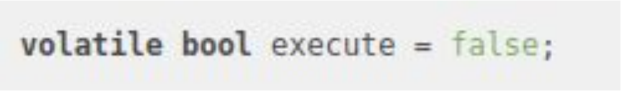
Code: Creating the ISR



```
// an interrupt for each sensor
void interrupt1 () {
  noInterrupts();
  time_sensor[0] = micros();
  interrupts();
}
void interrupt2 () {
  noInterrupts();
  time_sensor[1] = micros();
  interrupts();
}
void interrupt3 () {
  noInterrupts();
  time_sensor[2] = micros();
  interrupts();
}
void interrupt4 () {
  noInterrupts();
  time_sensor[3] = micros();
  interrupts();
}
```

Tips for Using Interrupts

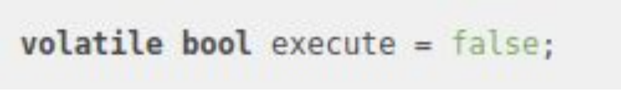
- ISRs only work on pins with interrupts!
- Keep ISRs short
  - NO PRINTING! NO DELAY!
  - Use Flags
  - Use simple if / else or case / switch
- Use `volatile` keyword!
  - Tells compiler not to optimize variable out!



```
volatile bool execute = false;
```

More Tips for Using Interrupts

- Disable Interrupts when reading / writing outside the ISR
  - Enable Interrupts: `interrupts()`
  - Disable Interrupts: `noInterrupts()`
- ISRs have a cost, too!
  - ISR occurs → context switch
  - Polling isn't always bad!



```
volatile bool execute = false;
```

Lab Time

Lab Info

- Create a counter state machine
- Write interrupts for each button
  - One to count up
  - One to count down
- Implement state machine
- Display state using board LEDs