Variables and Arithmetic

Lesson 2

Everything you wanted to know, and more, about variables

Lesson Objective

Obtain an understanding of the following concepts

- What is a variable
- Declaring and initializing
- Variable types
- Variable rollover
- Variable scope
- Performing arithmetic operations

What is a variable?

A container (of space in memory) for information

Three main properties

1. Name
2. Type
3. Value

Variables in computing are the same as in math

Variable Structure

Variable (Constants too) Structure

- Store information referenced by a name
- Should be clearly named and **specific**
- Multiple data types are available
- Constants are variables whose value will not change (during runtime)

Modifier	**Data Type**	**Informative Name**	**Assignment**
const	int	DELAY_TIME_MS	= 1000;
optional	char byte int double float string	Constants use **UPPER_CASE** Variables use **camelCase**	*optional*

Variables are like maps to storage lockers and if you have a lot of maps, you want helpful names telling you what you'll find in each locker.

Declaring variables

As simple as

```
int x;
```

Should always **initialize** during declaration (give the variable a value)

```
int x = 0;
```

Always give informative names

```
int leftMotorPin = 9;
```

⚠ Variables should always be initialized when declared.

Naming Rules

- Must start with letter or underscore ‘_’
- Case sensitive
    - `int x;` is different than `int X;`

⚠ Do **NOT** use case to differentiate two variables

Naming Conventions

- camelCase
  - The first word is all lower-case, each following word is title-case
- UNDERSCORE_CASE
  - Typically used for CONSTANTS, variables whose value doesn’t change
- _underscoreFirst
  - Generally used for member variables (variables within a function)
- underscore_variable
  - Another common alternative to camelCase

Initializing variables

- The act of giving a variable its initial value
- This should always be done at variable creation or declaration
- Uninitialized variables contain garbage values

At declaration

```
int x = 10;
```

After declaration

```
int x;

void setup() {
    x = 10;
}
```

Variable Types

Data Types

- Three major types
  a. Logic (true/false)
  b. Numeric
     - Integer
     - Decimal (Floating Point)
  c. Alphanumeric

- Others include
  a. Pointers
  b. Void
  c. Arrays

Logical

- Two values
    - True   1
    - False   0
- bool (boolean)
    - Should use bool as it is the standard c++ type
- bools are technically 1 byte

Numeric - Integers

Integers are whole numbers

- Optional modifier unsigned

Data Type	Bytes	Range
char	1	-128 to 127
int	2	-32,768 to 32,767
long	4	-2,147,483,648 to 2,147483,647
unsigned char/byte	1	0 to 255
unsigned int	2	0 to 65,535
unsigned long	4	0 to 4,294,967,295

Numeric - Floating Point

Floating Point numbers are fractional (having a decimal)

- Offers 6-7 decimal places of precision
- Very computationally expensive
- Not natively handled by Arduino

Data Type	Bytes	Range
float	4	3.4028235E-38 to 3.4028235E+38
double	4	3.4028235E-38 to 3.4028235E+38

Alphanumeric

For storing values other than numbers

- When using char to store alphanumeric; use quotes
    - char charValue = ‘3’; will store the character 3 or the numeric 51
    - Using alphanumeric might produce unexpected results

Data Type	Bytes	Range
char	1	Holds a single Char or 0 to 255
string	1+(1*# of chars)	N/A

Arrays

Arrays are collections of variables

- Typically of one data type but can be multiple types
- An example might be to hold a number of sensor readings
  - Rather than create 20 variables, you hold one array of 20 readings

Data Type	Bytes	Range
any	1 byte + (sizeOfType * # of elements)	N/A

Variable Rollover

When a variable exceeds it’s range, it rolls over to the other end of its range.

- This occurs in both directions

```
unsigned char x = 255;          // x is 255
x = x + 1;                      // x is 0
x = x - 1;                      // x is 255 again

char y = 127;               // y is 127
y = y + 1;                  // y is -128
y = y - 1;                  // y is 127 again
```

Modifiers

Some variables have optional modifiers

- const
  - Defines a variable as constant (never changing)
  - Not *needed* but explicitly tells the compiler to not allow changing
  - Can be used on all variable types
- unsigned
  - Available only for some variable types (most integer based)
  - Used to provide more flexibility while maintaining a smaller type

```
const char LEFT_MOTOR_PIN = 9;
char LEFT_MOTOR_PIN = 9;
unsigned char leftMotorSpeed = 225;
const unsigned char MIN_MOTOR_SPEED = 200;
```

Choosing a Variable Type

- Be sure that its range is within the bounds you need
- char/unsigned char is good for pins
- int is not much more expensive, may be safer
- Use floating point **only** when necessary (very expensive)

Variable Scope

Variable Scope

- Variable scope determines where a variable can be “seen”
- To some extent, it helps keep storage space down
- Variables should use the smallest scope necessary

Global Scope

- Accessible by the entire program
  - Any function
- Generally defined at the top (outside of a function)
- Can get unwieldy with too many
- Use only when necessary

Global is like storing files in a public network folder

Local Scope (member variables)

- Visible by any code within that function

```
void setup() {
    int memberVariable x = 0;  // Only visible within setup()
}

void loop() {
    memberVariable = 4;        // This will result in an error
}
```

Local scope is like storing files on your personal computer

Formal Parameters

- Like member variables but provided from outside
    - Changing the value locally doesn't affect the source*

```
int addIntegers(int this, int that) {
    // I can see this and that because they were provided
    // I can't change their values outside of my own function
    return this + that;
}
```

A formal parameter passed by value is like an email attachment

Loops

- Declaring within any loop
  - Recreated with each iteration
  - Not visible outside of the loop
  - Generally bad practice
- for loops (only within)

```
void setup() {
    for(int i = 0; i < 10; i++) {
        // i is only visible in here
    }
    i = 12;             // This will result in an error
}
```

⚠ Variable declarations within a non for-loop are generally a bad idea

Arithmetic

Basic Operators

Operator	Action
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo - Returns the remainder of dividing two numbers

Compound Operators

Operator	Action	Expression	Equivalent
++	Increment by 1	x = x + 1;	**x++; OR ++x;**
--	Decrement by 1	x = x - 1;	**x--; OR --x;**
+=	Increment by	x = x + 10	x += 10;
-=	Decrement by	x = x - 5;	x -= 5;
*=	Multiply by	x = x * 2;	x += 2;
/=	Divide by	x = x / 2;	x /= 2;

~~Order~~ Precedence of Operations

Operator	Action
()	Parenthesis
++ --	Increment, Decrement
* / %	Multiplication, Division, Modulus
+ -	Addition, Subtraction
< > <= >=	Less than, Greater than comparisons
== !=	Is/Is Not Equal to
&&	Logical AND
||	Logical OR
= += -= *= /=	Assignment and compound assignment operators

Parting Thoughts

- Always initialize during declaration
- Try to use the smallest data type
- Understand the scope of a variable
- Compound operators while handy, can be confusing

Questions?