Introduction to Programming

Binary Logic

- Modern computers store data and perform operations using Boolean/binary logic
  - Boolean Logic values can take 2 values: **True or False**
  - Binary numbers have 2 possible values: **1 or 0**
- A single true or false value is a basic unit of information called a bit

- In modern computing, individual bits are grouped in sets of 8 bits, called a byte

Example of a byte: 0 0 1 0 1 1 0 1

  - A series of bytes can be used to represent numbers, characters, images, video, etc.

Decimal vs Binary

- The numbering system we are used to is called **decimal**, base 10
  - 10 possible values: 0,1,2,3,4,5,6,7,9
  - Rightmost digit is the least significant (1s place)
  - Each digit to the left is a higher power of 10

    $9=9*10^{0}$, $90=9*10^{1}$, $900=9*10^{2}$, $9000=9*10^{3}$…
- Compared to binary, **base 2**
  - 2 possible values: 0,1
  - Rightmost digit is the least significant (2^{0} place)
  - Each digit to the left is a higher power of 2

    $1=1*2^{0}$(1), $10=1*2^{1}$ (2), $100=1*^{2}$(4), $1000=1*2^{3}$ (8)…
- These are just ways of representing the numbers; the underlying numbers and operations(+,-,x,/) are the same

Reading a Binary number

- To convert from binary to decimal, use a base of 2 and powers beginning with 0 at the left, counting upwards to the right
- This technique is the same in decimal systems; it is similar to how “10” is 10 times larger than 1
- For example, “2” in decimal can be represented in binary as “0010”

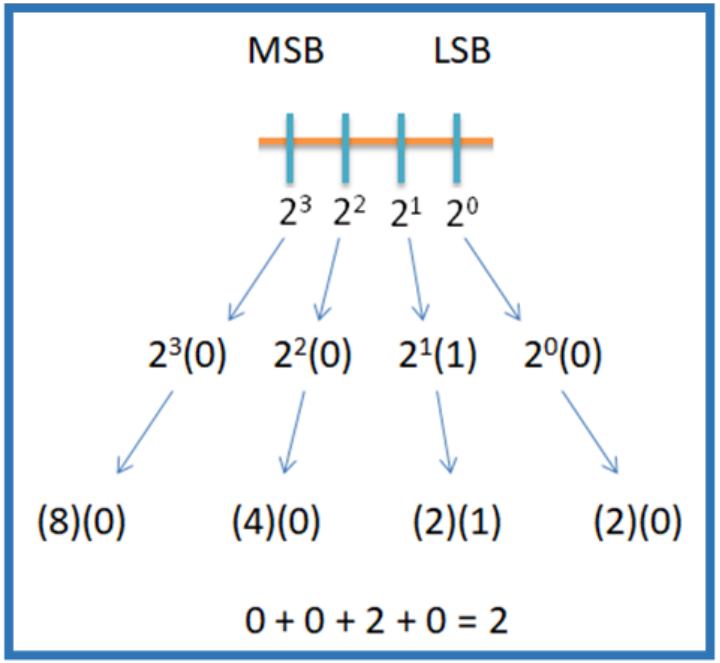


Figure 1: Convert binary to decimal

Example: Convert $(0010)_2$ to decimal = $2^3(0) + 2^2(0) + 2^1(1) + 2^0(0)$
= (8)(0) + (4)(0) + (2)(1) + (2)(0)
= $(2)_{10}$

Hexadecimal System

- Sometimes your data may not be a meaningful representation other than the base binary
- A string of 1s and 0s is not easily read by humans
- To display in a more readable format, 4 bits are grouped together
  - 4 bits gives you 2^4 values
  - 16 values in 0-15
  - This is called **Hexadecimal** or **hex**
- 2 hex numbers make a byte
  - A single byte goes from 0(0) to FF (255)

Hex Representation	Binary Representation	Decimal Representation
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Hexadecimal number system

Negative Numbers

- To represent negative numbers in binary, you take the left-most bit and use it for the sign, 0 for +, and 1 for -
  - There are common ways of doing this called **One's Complement** and **Two's Complement**
  - Two's complement is the most common since it allows subtraction to be performed correctly, subtracting a number is the same as adding a negative number
- Because computer variables have a finite number of digits, this can lead to errors
  - If you increase a binary number so that the leftmost bit becomes a 1, the computer may interpret it as negative **(called an overflow)**
  - So we either tell the computer the number can only be positive (**unsigned**) or make sure the variable has enough bits
  - This is a common error when using a timer or loop counter
    - A common type is an **int,** which stores values from 32767 to - 32768, so if you are storing milliseconds, it will overflow after ~30 seconds

Common Programming Languages

- There are many different languages
  - Vary in usage, strengths, and format
- Common programming languages:
  - C and derivatives, C#, C++
  - Java, JavaScript
  - Python
- Arduino uses a variation of C++
- We will focus on learning Arduino C++, the programming language for Arduino hardware

```
#include <SPI.h>
#include <SD.h>

const int chipSelect = 4;

void setup() {
  Serial.begin(9600);
  while (!Serial) {
    ;
  }
  Serial.print("Initializing SD card...");
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    while (1);
  }
  Serial.println("card initialized.");
}
```

Figure 2: An example of Arduino Code

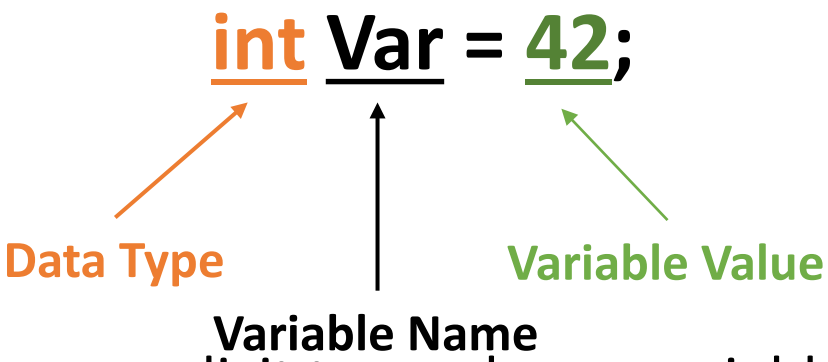
Basic Parts of a Computer Program

- Variables – The data you are working with
- Operators – The way you change the data (simple)
- Control Structure – How you control the flow of your program
- Syntax – Markup characters and language
- Functions – More complex combinations of the above elements

Variables

https://docs.arduino.cc/language-reference/#variables

- Variables are data values saved in memory to be changed or used during code execution
  - Usually changeable but can be declared as CONSTANT
- Variables consist of three primary parts – the data type, the variable name, and the variable value

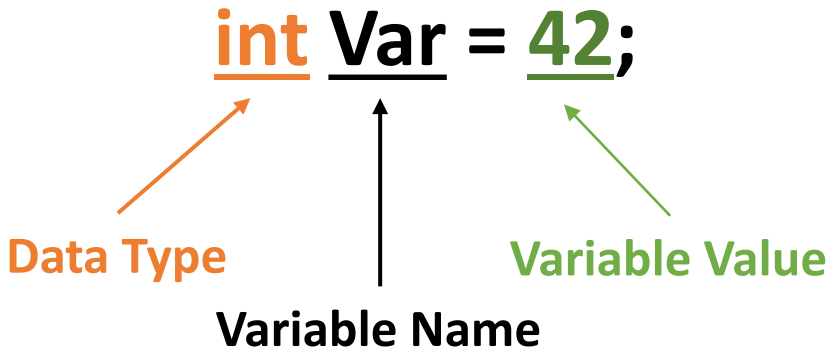


C++ requires an explicit type when a variable is created

Assigning a value at creation is not required

Variables Scope

https://docs.arduino.cc/language-reference/#variables



- Where a variable is created is called a definition
- Variables can be defined inside of a function (called local) or outside of all functions (called global)
  - Global variables can be accessed by the whole program
  - Local variables can only be accessed by their function

Common Data Types

https://docs.arduino.cc/language-reference/#variables

Data Type	Keyword	Size in Memory (bytes)	Range of Values	Note:
Signed Integer	int	2	-32768 to 32767	Notice max value, above that it would roll over to negative, use **unsigned int** or **long** for larger number
Character	char	1	0 to 255	Each value corresponds to a printable character or keyboard operation called **ASCII**
String	String	varies	varies	This is different than an array of chars and behaves differently
Boolean	bool	1	True or False	Even though only 1 bit is required the minimum variable size is 1 byte
Floating decimal point number	float	4	-3.4×10^{38} to 3.4×10^{38}	Precision is limited to 6-7 digits, on Arduino Mega **Double** has same precision
Array	type[size]	varies	varies	An ordered group of multiple values of the same type, ex. int array_of_ints[5] is 5 integers

Commonly used C++ data types

Operators

https://docs.arduino.cc/language-reference/#structure

- Operators usually perform a single functional operation, such as adding or subtracting
- Operator Types include:
  - Assignment
  - Arithmetic
  - Comparison
  - Logical
  - Bitwise (not covered here)
  - Compound (not covered here)
- Operations must be defined for the variable types, and some operations may be undefined for some types

Assignment Operator

https://docs.arduino.cc/language-reference/#structure

=

Performing an operation does not necessarily change the value of a variable

The assignment operator stores the result of the code to the right of the = in the variable to the left

X=7;

Stores the value 7 in the variable X

Nothing prevents you from trying to assign differing variable types to each other (like putting a char in an int) but sometimes the results may not be the expected.

Arithmetic Operators

https://docs.arduino.cc/language-reference/#structure

- Arithmetic operators are mathematical functions that take two operands, perform a calculation, and provide a result

Operator	Description	Example
+	Adds two operands	A + B
-	Subtracts second operand from first	A – B
*	Multiplies operands	A * B
/	Divides dividend by divisor	B / A
%	Modulus operator: Remainder of quotient	B % C

Arithmetic operators

Comparison Operators

https://docs.arduino.cc/language-reference/#structure

- Comparison operators are used to compare two values (called operands)
- Produced a Boolean (True/False) result
- Usually used in combination with a control structure

Operator	Description	Example
==	Equal to	x == y (x is equal to y)
!=	Not equal to	x != y (x is not equal to y)
<	Less than	x < y (x is less than y)
>	Greater than	x > y (x is greater than y)
<=	Less than or equal to	x <= y (x is less than or equal to y)
>=	Greater than or equal to	x > y (x is greater than or equal to y)

Comparison operators

Logical Operators

https://docs.arduino.cc/language-reference/#structure

- Logical operators manipulate True and False results into more complex operations
- All Nonzero values are considered True (1 is true, but so is 617, 32, 100542)

Operator	Description	Example
&&	AND – If both operands are True, the condition is True; otherwise, it is false	If A = 1 and B = 0, then A && B = false
||	OR – If either operand is True, the condition is true; otherwise, it is false	If A = 1 and B = 0, then A || B = true
!	NOT – If a condition is true, then !condition is false	If A || B = true, then !(A || B) = false

Logical operators

Control Structure

https://docs.arduino.cc/language-reference/#structure

- Control Structure is used to run code in a more complex sequence than simple sequential steps
  - Run the same code multiple times with a loop
    - For, while, and do while
  - Only run a portion of code if some condition is true
    - If … else
- Uses comparison and logical operators to determine if a section of code will be executed or skipped over

Conditional Statements: If/Else

- An if statement performs a logical check and executes the following code if true
- May be combined with **Else** which executes code when the If is false

```
void setup() {
    Serial.begin(9600);   }
void loop() {
    int A = 15;
    int B = 10;
    if (A >= B) {
        Serial.println (A – B);
    }
    else {
        Serial.println (B + A);
    }
```

In this example, since A is greater than or equal to B, 5 will print out (A – B), but the second print will not be executed.

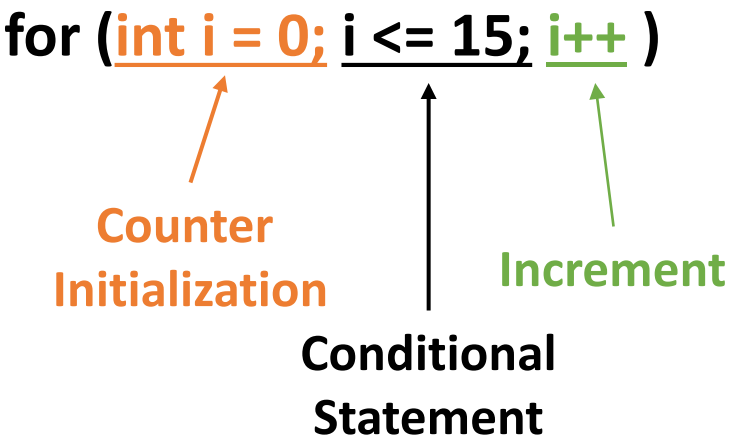
```
void setup() {
    Serial.begin(9600);   }
void loop() {
    int A = 5;
    int B = 20;
    if (A >= B) {
        Serial.println (A - B);
    }
    else {
        Serial.println (B + A);
    }
```

Now since B is greater than A, the first print will be skipped, and 25 will be printed (B + A)

For Loops

A **for loop** executes repeatedly and increments a counter variable until the conditional statement is no longer true

- The initialization happens once when the loop starts
- The increment happens at the end of the loop
- The logical check happens at the beginning of the loop (pretest condition)




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

void loop() {
    for (int i = 0; i <= 15; i++)  {
        Serial.println(i);
    }
}
```

Output → 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

A for loop that prints 0 to 15. The variable i starts at 0 and increments every loop until i=16 and then the loop ends.
Note:++ is a compound operator i++ is the same as i=i+1

While Loops

A **while loop** will only run when the conditional statement is true, but it does not automatically increment or break the loop unless something in the loop changes a value in the conditional statement

while (carrier < 0) { Serial.print (carrier++) }

Conditional Statement **Loop Execution**

```
int carrier = 0;

void setup() {
    Serial.begin(9600);
}

void loop() {
    while (carrier < 20) {
        Serial.println(carrier++);
}
```

A while loop that counts from 0 to 19

A **do while loop** is similar but performs the logical check at the end of the loop

Functions

https://docs.arduino.cc/language-reference/#functions

- A function is a code segment written to be executed elsewhere in the program
- A function:
  - Can return a result, the function has type matching what it returns (void means no return)
  - Must have a unique name and cannot be defined inside another function
  - Any variables you want to pass into the function (called arguments) are used placed inside the parentheses
- Functions can be predefined in libraries or written by the user

```
int giveme_seven (int argument) {
      int local_var;
      return 7;
}
```

A very simple function that always returns the number seven. Normally argument and local_var would be used in the function.

```
void loop () {
      int value=2;
      int return_value;
      return_value=giveme_seven(value);
      Serial.print (return_value);
}
```

A call to the function, when called it returns 7, which is assigned to the variable return_value and then printed

setup() and loop()

- Arduinos uses two special void functions in every program
- setup() runs one time when the program begins
- loop() that runs continuously thereafter

```
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

Initial functions of a new sketch created automatically by the IDE

Syntax

- Some code language is instructions to the compiler, which translates your code into machine code, or use for grouping
  - #include is used to add libraries of prewritten code to your program
  - {} curly braces are used to group code together for loops, functions, and if statements
  - ; terminates a line
  - // and /* */ are used to indicate comments (text the computer will ignore, for documentation and notes)

Leaving Comments

- Comments are used to explain what your code does, leave notes, document changes made, etc.
  - The clearer and more detailed your comments, the more useful they will be
- If you can fit your comment on one line, then simply type two backslashes followed by your text

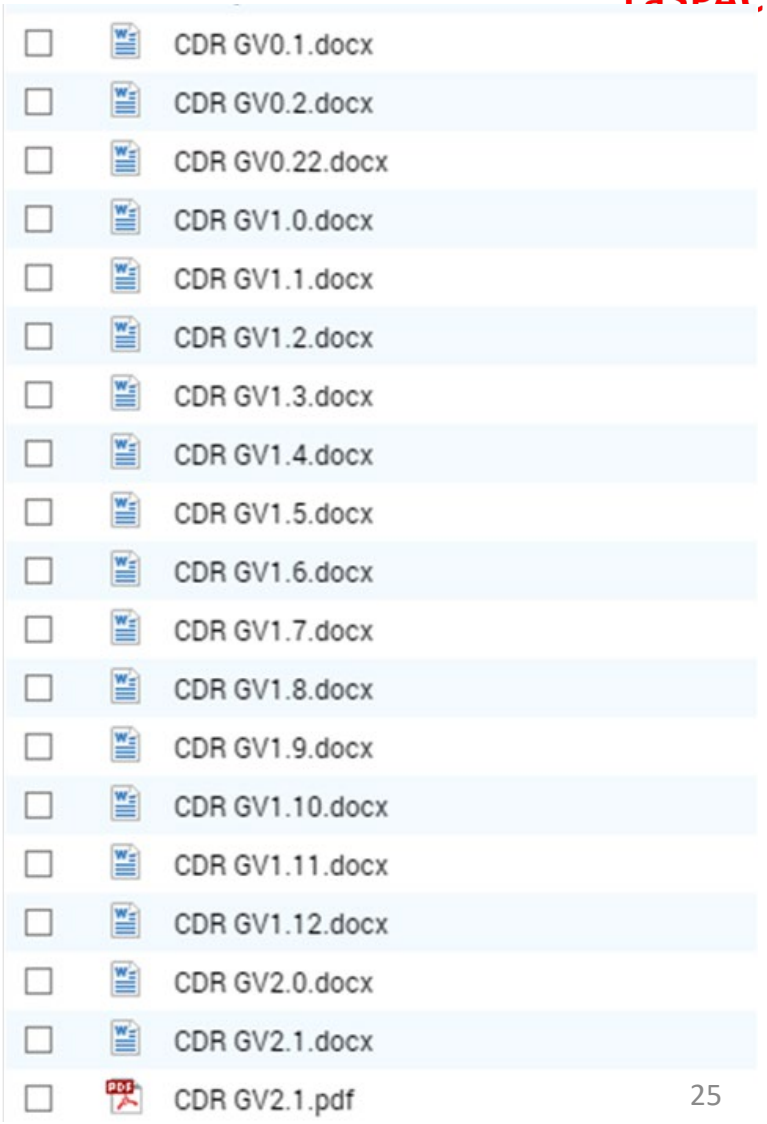
```
// Leave a one-line comment like this
```

- If you need more room, then use a backslash and asterisk combination to comment over multiple lines

```
/* Use as many lines as needed in order to provide
enough information for someone else to understand
your code */
```

Version Control

- While developing software, it is important to track the changes made within your code
- For your code, it is recommended that you provide a version number (or date and time) in your filename
  - Change the name when you make changes to the code
  - Keep a log of the changes you make in your lab notebook or an outside document
  - Be aware that the Arduino IDE will automatically save when you compile or upload
- git is a common version control system, but complex for LaACES



Troubleshooting Your Code

- Check syntax and variable names (Upper and lower case matter)
  - Check punctuation: semicolons, brackets, and parentheses must be placed correctly
- Ensure correct placement of conditional statements and loops
- Use correct data types
- Make sure global and local variables are accessible to the appropriate functions
- Use print statements at intermediate steps to make sure the program is running the steps you think it should and calculating the value you expect