CircuitMinds Software Instruction Manual

Arham Siddiqui

August 2023

Contents

1 Introduction

In pairs, you will be tasked to design, code, and test model vehicles to navigate an obstacle course as quickly as possible. Creativity in vehicle modifications and coding is encouraged and rewarded. This workshop will give you an idea of the real world applications of coding, and you will see your ideas come to life! The programming language being used for this workshop is C++, which is Arduino's inherent language. Please note that this manual will not cover everything regarding this programming language, rather it will go over what is necessary for this workshop in particular. Nevertheless, following along with this manual and understanding the implementation of this code will set up a great foundation for future programming endeavors.

1.1 Prerequisites & Materials

No previous coding experience is required. Here are the materials you will need to begin:

- CircuitMinds V2 Development Kit
- Chromebook or personal computer
- Connection to internet
- Construction and competition rules
- Paper and pencil for sketching designs
- Supply of legos (provided at workshop)
- Enthusiasm and curiosity!

1.2 Engineering Connection

This workshop requires you to reason with your knowledge of how kinetic energy works. The primary challenge is to code the vehicle to move through the obstacle course in the most efficient way possible given the constraints. Similar to how real-world engineers must use practical applications of code and hardware design to achieve a goal, you will learn how challenging this can be. For example, mechanical engineers may have to sacrifice the speed of a car in order to achieve the look that they want. They have to create a balance between what they want to see in the car and what they need to achieve mechanically. A car with fancy seats and controls may become too heavy to achieve the speed that the engineers want. Once the mechanical design is done, software engineers must work within the constraints presented by the hardware in order to create the most efficient code possible to control the car. In this activity, you will also experience the steps of the engineering design process as you brainstorm, prototype, test, and iterate on designs to create more successful solutions.

1.3 Learning Objectives

After this workshop, you will be able to:

- Identify how weight affects the momentum and energy of a vehicle
- Understand the basics of the engineering design process
- Explain what is meant by a design constraint
- Explain the logic of changes made to a design after a prototype test
- Understand the basics of code and how it is applied in the real world

2 Basic C++

2.1 Syntax

In programming, it is essential to understand the syntax of the language you are working with. The word syntax simply means the set of rules making up a language, or the linguistic characteristics that form together to create the language. You must keep this term in mind whenever writing code in your program. Every statement, function, loop, or procedure you write must follow proper syntactical practice or else you will get errors instead of working code. For this workshop, this manual will be your guide to proper syntax, so make sure to pay attention to the spacing, special characters, and spelling that you see in this manual when writing the code by yourself. Programming languages are notorious for having challenging syntax, but over time you'll get the hang of it!

2.2 Data Types

Data types are a straightforward concept in programming: they refer to the type of data of a particular value. In programming, common data types include boolean (true or false), int (integers), float (decimals), char (characters), and string (sequence of characters). In this workshop, you will only need to work with the int - or integer - data type.

2.3 Variables

A variable is a designated memory location or "slot" within your program where data can be stored. Its name "variable" stems from its ability to be altered as the program runs. Instead of dealing with actual memory addresses, we assign meaningful names to these variables to stay organized in our program.

In order to declare a variable for our program to understand, we must initialize it with its name and corresponding datatype. In our program, we will start by initializing integer variables to store corresponding pin numbers on our physical engineering board. View the following code snippet and make sure you understand how these variables are being declared.

```
1 int enA = 7;
2 int in1 = 9;
3 int in2 = 8;
4 int enB = 6;
5 int in3 = 5;
6 int in4 = 4;
```

It is important that you copy this code down in your own IDE (Integrated Development Environment), or "coding area". This will be the beginning of our program to make the car move!

3 Control Structures

3.1 Conditionals

When it comes to tackling real world problems, such as a car navigating through an obstacle course, it's essential for our program to factor in data, or user input, for the purpose of decision making. To manage such data, programmers use Control Structures, which rely on the concept of comparison to operate. Control Structures allow us to add logic to our program and make it act based on decisions!

The most common type of control structure used in programming is the if-else statement. The syntax for this statement in C++ is quite straightforward.

```
if ( boolean-value is true ) {
    executeStatement;
}
else {
    executeAlternateStatement;
}
```

In the above code snippet, the word if starts the statement, followed by a boolean comparison (remember this means either true or false) in parenthesis, curly brackets , and the statement you want executed between these brackets. In the example above, if the boolean value is True, "executeStatement" will be ran by the program, otherwise "executeAlternateStatement" will be ran by the program. Mostly, the boolean comparison comes in the form of a function which returns either True or False when referenced. Functions will be discussed later. The following code snippet incorporates data types, variables, and an if-else statement. Can you tell what the value for each variable will be when the program is executed? (Answers are given below)

```
int x = 1;
int y = 2;
int z = 3;

z = 1;

if ( x == y ) {
    x = 5;
}
else {
    z = 9;
}
```

Answer: x=1, y=2, z=9

In this example, interpret the "==" as "is equal to". If you are still having trouble understanding the code above, discuss with others around you or call one of the helpers over. Also, you don't need to write any code that isn't explicitly stated as part of the program. This example was just for practice.

3.2 Switch Case

The switch statement is a more efficient way to compare a variable to multiple different numbers at the same time instead of using long if statements. It's like a fork in the road with many paths. Being another Control Structure, the switch statement allows you to change the functionality of your program depending on a value. The syntax for the switch statement can be seen in the code below.

```
switch (expression)  {
    case expressionA:
        // code to be executed if
        // expression is equal to expressionA;
        break;

    case expressionB:
        // code to be executed if
        // expression is equal to expressionB;
        break;
        .
        .
        .
    default:
        // code to be executed if
        // expression doesn't match any cases
}
```

It is important to note that in the switch statement, what is labeled expression in paranthesis is not a boolean return, as what was in the parenthesis in the if statement. Instead, it is simply an expression (think of a word or number), and the following cases are possible values for that expression. With this information, which case would be executed in the code snippet below? (Answer is given below)

```
switch ("abc")  {
    case "abcd": // case 1
        break;

    case "ab": // case 2
        break;

    case "a": // case 3
        break;

    case "abc": // case 4
        break;

    case "b": // case 5
        break;

    default: // case 6 (note that default isn't technically a case)
        break
}
```

Answer: case 4

3.3 Loops

A loop in programming allows a set of instructions to be executed repeatedly. For instance, if you want to print a message multiple times or call a function repeatedly, using a loop is much more efficient than writing the same statement repeatedly or calling the function over and over. Loops improve program readability and efficiency by simplifying repetitive tasks.

There are three main loops in C++: for loop, while loop, and do-while loop. In this workshop, you will be working with the for loop. Any loop has four main components: the initialization expression, test expression, update expression, and body of the loop. Take a look at an example for loop below.

```
for (int i = 1; i <= 5; i++) {
    statement;
}
```

In this code snippet, "int i = 1" is the initialization expression, "i ¡= 5" is the test expression, "i++" is the update expression, and "statement;" is the body of the loop. Not that the initialization expression is initializing a new variable "i" that is being used in the test expression. "i++" simply means "i = i + 1", which increments the value of "i" by 1. In summary, the for loop will execute "statement;" as long as the variable "i" is less than or equal to 5, and "i" will increase by 1 every iteration of the loop.

In the example below, can you tell how many times the statement will execute? (Answer given below)

```
for (int j = 0; j <= 7; j++) {
    statement;
}
```

"statement;" will be executed 8 times

Remember that "statement;" is place holder value representing the body of your for loop. This can be replaced with function calls, if-else statements, and even more loops!

4 Functions

Functions - sometimes called methods - are blocks of code that perform a specific task. They help organize your code, make it easier to understand, and, as the name implies, add functionality to your program. For the purpose of this workshop, you should be familiar with defining and calling a void function. Void simply means that the function doesn't return a value, though it still adds functionality to the program. An example of a function being defined can be seen in the snippet below.

```
void moveForward() {
    moveLeftMotor();
    moveRightMotor();
}
```

The void keyword is used to tell the program you are defining a function. "moveForward" is the name of the function followed by parenthesis. The curly brackets indicate the start of the body of the function. "moveLeftMotor()" and "moveRightMotor()" are two function calls that make up the body of this function.

Function calls are how we access the functions that we defined earlier. A function call is made up of the name of the function followed by a pair of parenthesis. This tells the program that we want to execute the code inside that function on the particular line we are calling the function. In the following example, you should be able to identify where the function is being called and defined. At which line is the function defined at the top being called? (Answer is given below)

```
void moveForward() {
    moveLeftMotor();
    moveRightMotor();
}

switch (userInput) {
    case "f":
        moveForward();
        break;
    case "b":
        moveBackward();
        break;
    default:
        break;
}
```

Answer: Line 8