Development of Arduino Programs to control Hardware

Module Title: Computer Programming

Module Code: KD4008

Module Tutor: Dr. Gary Wells

Student name: Katie McConville

Student ID: 17003410

Student email: [katie.mcconville@northumbria.ac.uk](mailto:katie.mcconville@northumbria.ac.uk)

Contents

[Introduction 1](#_Toc511315789)

[Purpose of the program 1](#_Toc511315790)

[Planning 1](#_Toc511315791)

[Method 3](#_Toc511315792)

## Introduction

This report is for a program in C that works on Arduino, and will take words entered into the system and translate them into Morse code, where the output is both an LED and a sounder. This report is to show the planning, writing and then the testing of the program.

### Purpose of the program

The program being written needs to fulfil the following criteria:

* Translate entered words into Morse code
* Display this Morse code on an LED and using a Piezo
* Display the Morse code on the serial monitor of Arduino

The program must also follow these rules:

* The duration of a dot is considered to be one time unit
* A dash is 3 time units
* A space between each component of a letter is one time unit
* A space between letters 3 time units
* A space between words is 5 time units

The recommended time unit is 50ms, but this will be tested later in the report to check its accuracy.

## Planning

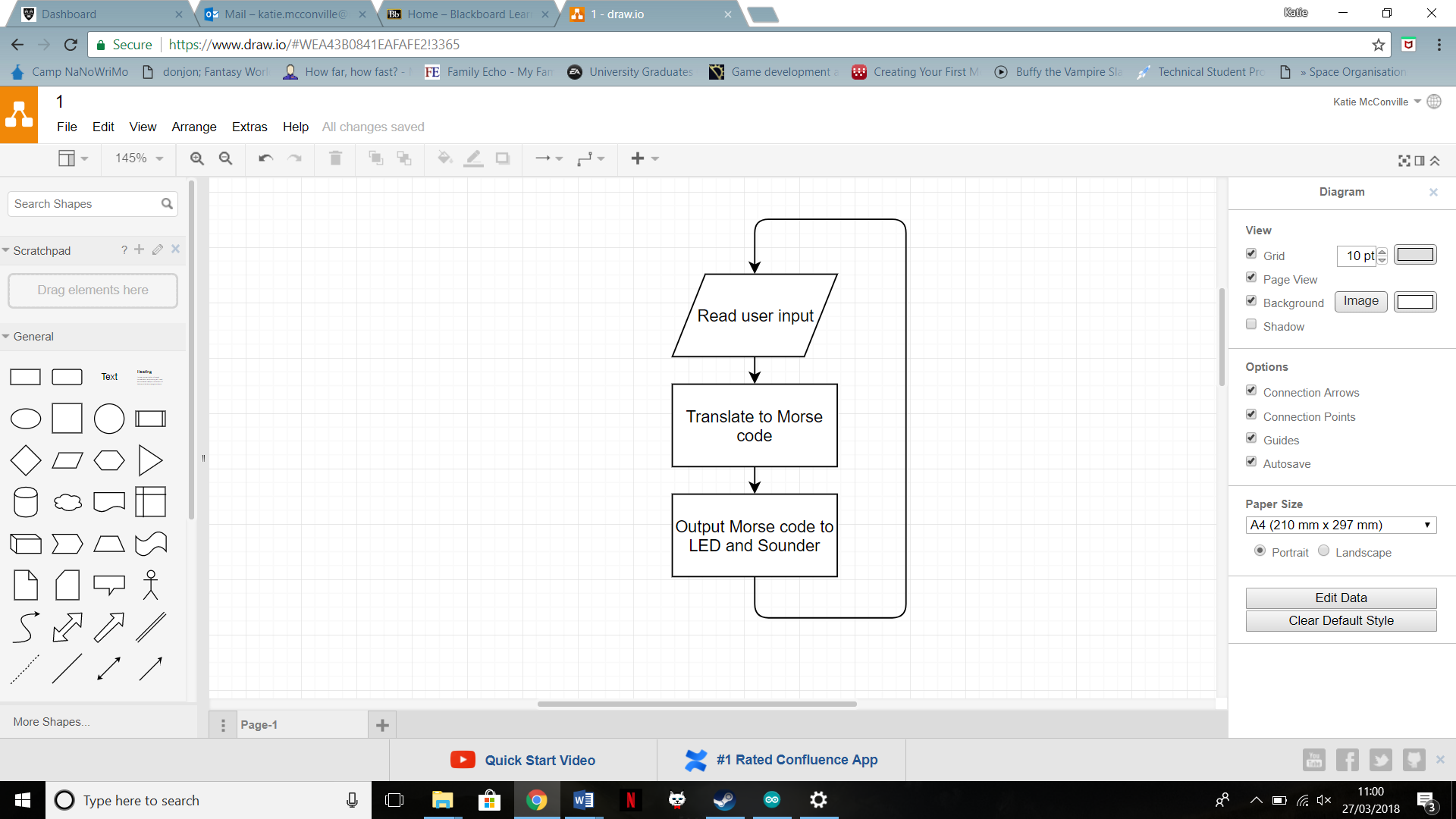


Figure - Simplest components flowchart

To start writing this program we had to begin by looking at the purposes and thinking of that in its most simple form of the input, processes and outputs. The fact that we are working with Arduino means that the program is looping constantly, so there is no definitive end point. The flowchart to the right (figure 1) is the beginning of this thought process, and shows the very simplest elements of what the program must consist of. From here, we can begin to break it down further.

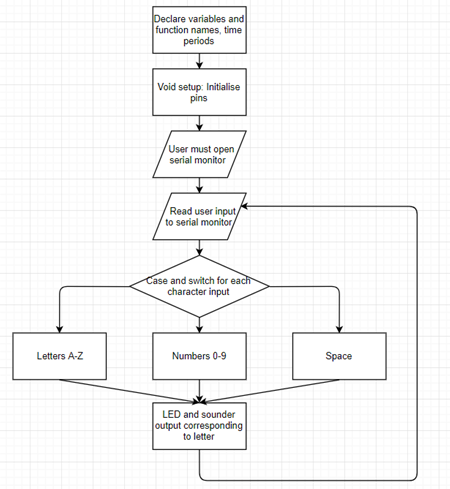
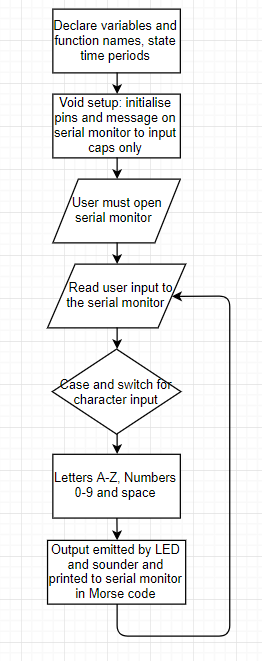
To expand this further, as seen in figure 2, we have to include the necessary functions for any program, which includes the declaration of all variables and the labelling for every term used. It is also where I will state the time period for a unit length, and then the time periods for dots, dashes, gaps between letters and gaps between words that were given to us in our initial brief. I will be specifying a unit length and then using that as a label in the formulas for each of the other functions so that as I am testing it I can change the time period in just one location rather than having to change it multiple times in the program and possibly leading to a discrepancy. In Arduino, what comes next is the setup, which is just the location for the initialisation to take place and tell the pic Arduino the function of the previously labelled legs, such as that in this case pin 7 is the output for the LED. From here, the user would have to open the serial monitor to be able to type in their message. This will then take us into the process part of the program, which is a continuous loop. It will take the typed input and work as a case and switch with each input character to find the correct output from the possible options. This output will then be emitted using the LED and the sounder, and the program will then loop back up to do a case and switch for the next character until the entered message is completed.

Figure 2 - An expanded flow chart

Figure 3 shows the further expansion of the previous flowchart, after an initial attempt at writing the program. The first change comes in that I’ve had to have the serial monitor inform the user that they can only print in capitals, as when there were lower case letters input there would not be an output at all. It is possible to include a conversion in the loop statement to enable the input of lower case letters, but this takes the usage of multiple libraries. I tried to just have all the letters again in the case and switch, this time lower case, but to do this required nested if statements in the loop that wouldn’t work. The next change made came in that the numbers, letters and space were now in one case and switch, rather than having an initial case and switch that would then break down into a further case and switch. The reason for this change is that if I had multiple words, or both letters and numbers the program would get stuck inside one of the subprograms and not be able to find these switches. The final change was that the actual output being emitted by the LED and the sounder would also be printed onto the serial monitor. This was to better help with testing the program, as it meant that I could be checking the output of the LED and sounder with the serial monitor, and then checking the output displayed on the monitor with the Morse code alphabet given with us to check everything was accurate.

Figure 3 - A further extended flow chart

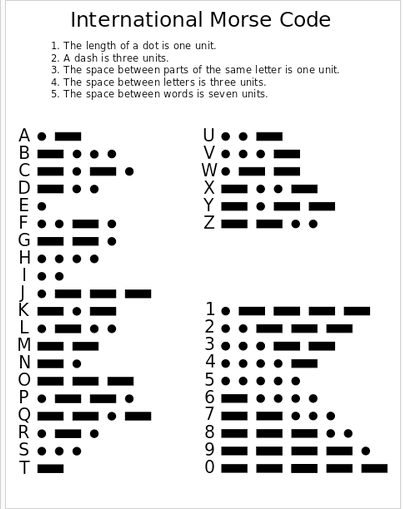


Figure 4 - The Morse code alphabet and numbers

Figure 4 shows the Morse code that will be used for this program, and will therefore be what the program will be tested against. It will also be the output into the serial monitor that will be visible to the user, written as the dots and dashes.

## Method

The program begins with declaration of the pin usage, and the frequency we want the piezo to sound at. Both the piezo and the LED will be on pin 7 of the Arduino.

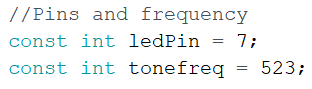


Figure 5 - The output pin and frequency

Before the initialisation, there is also the declaration all of the previously mentioned time periods, as shown in figure 6.

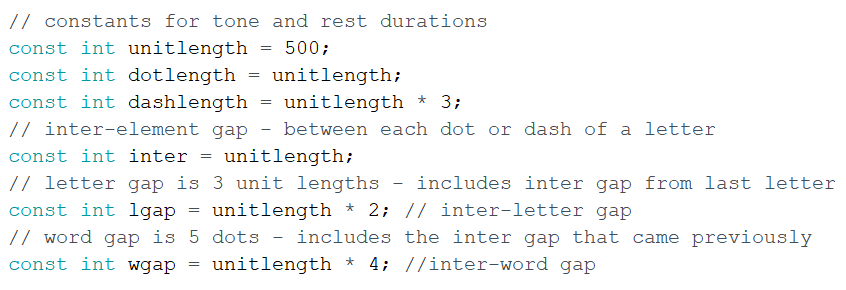


Figure 6 - Declaration of the time periods

In the comments you can see what each line relates to, and how for the word gap and letter gap I have come to the multiplier of the unit length to get the length we were given. The next part of the program is the void setup, which is where the initialisation of the necessary pins and the general system takes place. The program tells the chip to be ready to use the previously labelled ledPin as an output, for the LED and sounder. It then runs the command “Serial.begin” which initialises the pic so it will begin the continuous loop. I also have a “Serial.println” which prints to the serial monitor, so when the user opens the monitor it will inform them immediately that they can only use capital letters for the input. The time period shown in the picture is 500, as this was what was used during testing.

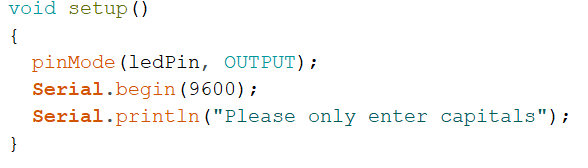


Figure 7 - Void setup, the initialisation

Figure 8 shows the next part of the program, the loop. Arduino works through a continuous loop so this is the main section of the program, and links to outside subprograms. First the program checks for an input in the serial monitor, and if it’s a character stated in the following if statement (A to Z and 0 to 9) then it assigns the value input to the label “thischar”. This is then used for the case and switch statement, searching through for the input character and then carrying out the subprogram. An example of one of the subprograms can be seen in figure 9. After completing this subprogram it will loop back, do a letter gap and then continue to go through the message character by character. When it hits a character that isn’t part of the values stated in the if statement it will then do the other option for the if statement, which is a word gap, as it is most likely a space.

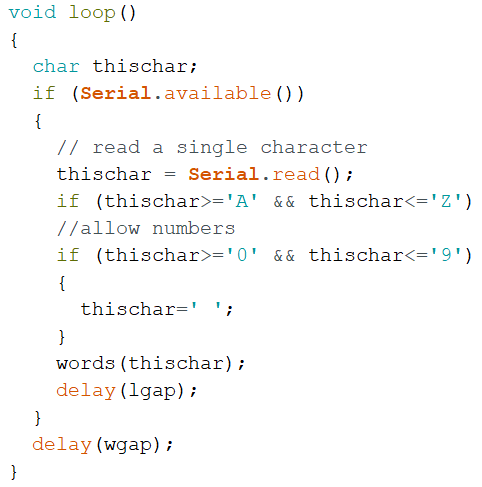


Figure 8 - the void loop

In figure 9 you can see the case and switch, with the letters A and B as an example. Dot and dash are both subprograms declared elsewhere in the program (seen in figure 10) so when the letter A is entered for the “thischar” value, it takes the program to this subprogram, and then it goes into each subprogram declared inside it. In the case of A, it will perform the dot function and then the dash function. It will then print the Morse code for the letter to the serial monitor, for the testing mentioned during the planning section. It will then return to the main loop program, to let the program continue through the rest of the entered message. Each letter and number is done in this format.

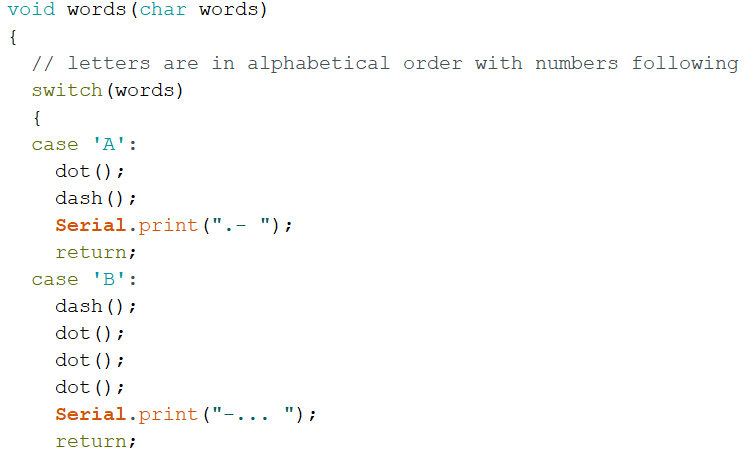


Figure 9 - The case and switch with letters A and B

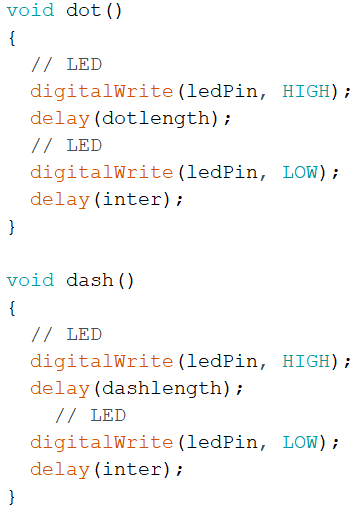


Figure 10 - The dot and dash functions

Figure 10 shows the subprograms for dot and dash that appear in the letter and number subprograms. They have generally the same format, with the only difference being the declared time for the delay to last, these are the previously stated time lengths relevant for the function. Both subprograms work by making the pin high, turning the LED on, and then doing the time delay for the relevant time. The pin is then made low, turning off the LED. The program then waits for the length of the inter letter gap previously declared before returning back to the letter subprogram it was part of.

## Testing

A large part of testing this program comes in the flashing of the LED and the sounder, both of which are unable to be shown in this report. However the output in the serial monitor can be shown, which is what the LED is being tested against to ensure its accuracy. This output is then checked against the Morse code translation shown in figure 4. I will test a few words to show that to the degree provable in this report the program is correct.

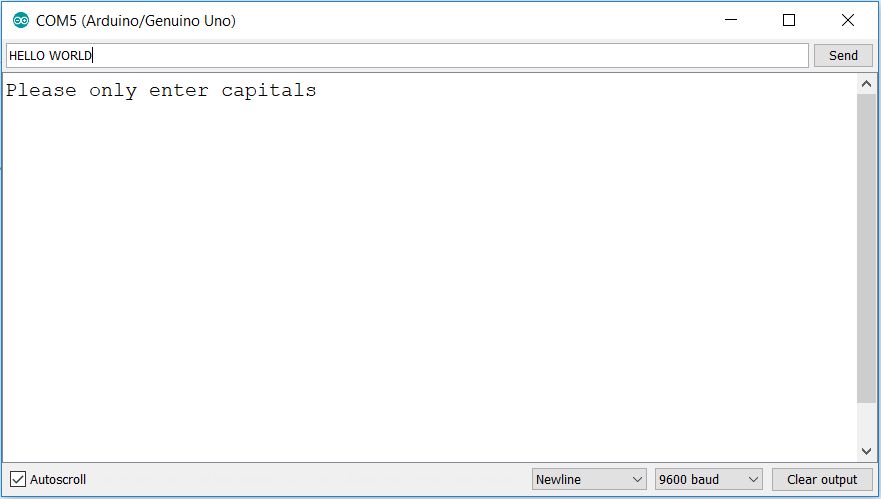


Figure 11 - "HELLO WORLD" test message

First I typed in the message “HELLO WORLD”, as seen in figure 11. The result of this is seen in figure 12, in Morse code on the serial monitor. I checked the LED with this and it was correct for what was printed on the screen. I can then compare this with the Morse code shown in figure 4, to check its accuracy.

Figure 4 showed me that the message that should’ve been printed was “…. . .-.. .-.. --- .-- --- .-. .-.. -..”. When looking at figure 12 we can see this is correct, and it was also the pattern shown on the LED.

This was the initial test of only a few letters, so after this, the whole alphabet must be checked. To do this, I entered the message “SPHINX OF BLACK QUARTZ JUDGE MY VOW”.

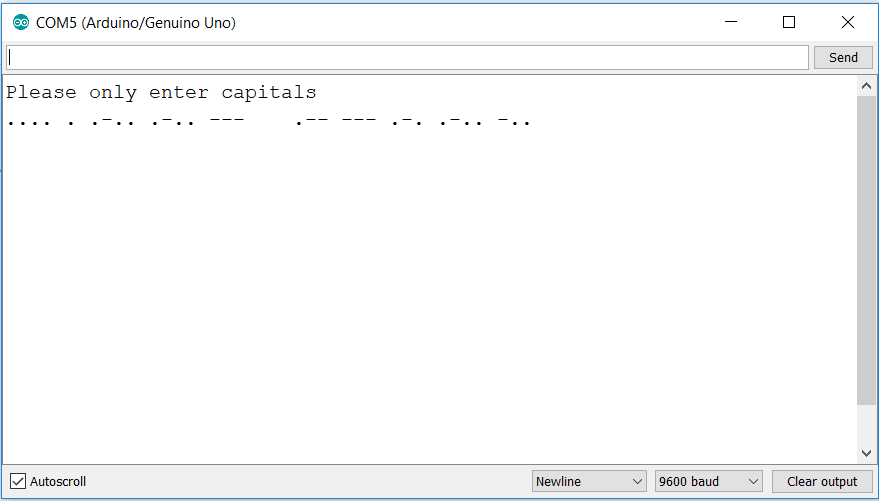


Figure 12 - "HELLO WORLD" message output on the serial monitor

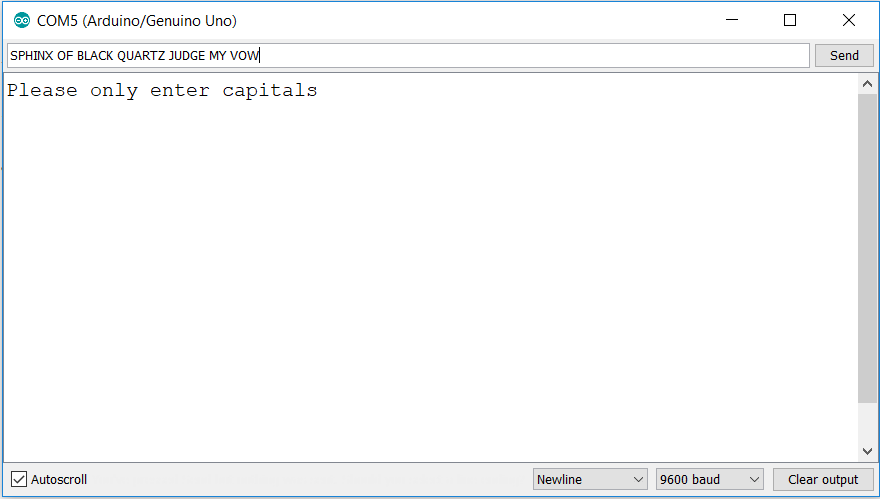


Figure 13 - Testing every letter of the alphabet

Going back to figure 4, the output that we should get for this message is “… .--. …. .. -. -..- --- ..-. -… .-.. .- -.-. -.- --.- ..- .- .-. - --.. .--- ..- -.. --. . -- -.-- …- --- .—". When we compare this with the output seen in figure 14, we can see this is correct. The LED pattern that went along with this message was also accurate to what was being printed on the serial monitor.

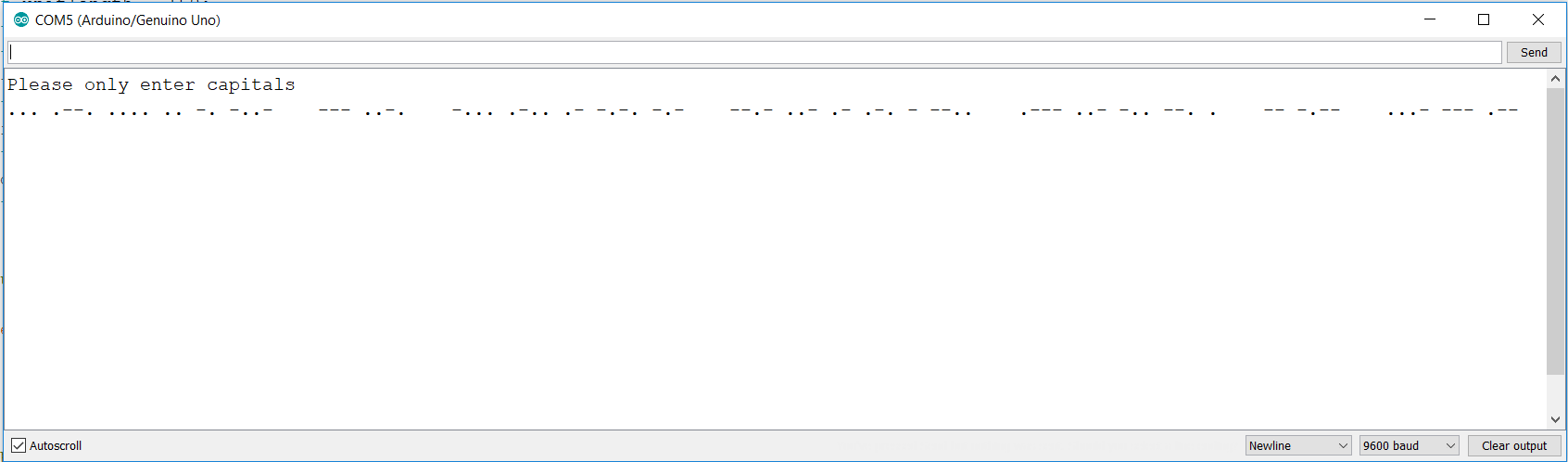


Figure 14 - The output for every letter of the alphabet

Now that every letter has been tested, numbers must also be tested. To do this I will just enter the message “0 1 2 3 4 5 6 7 8 9”. Figure 15 shows this message input.

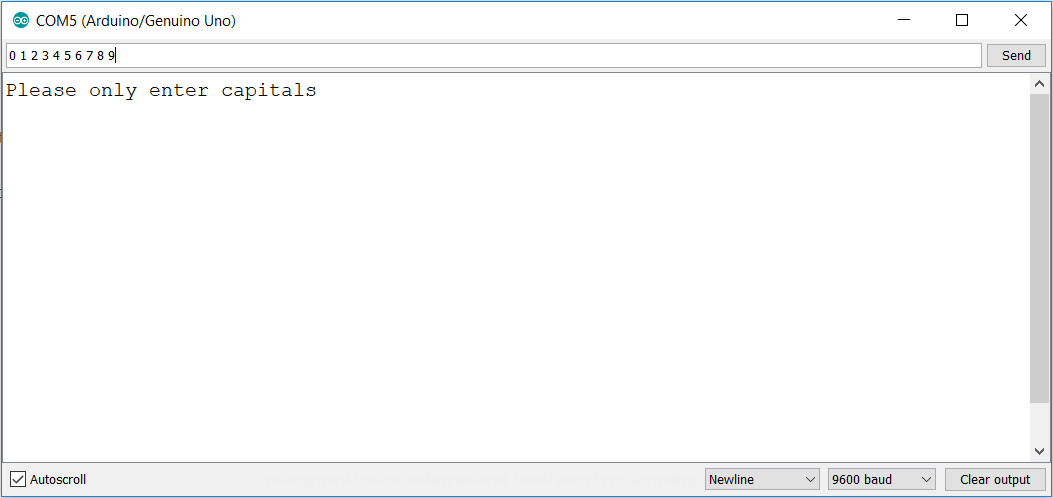


Figure 15 - Input of all numbers

For the output, figure 4 tells us we should get the output “----- .---- ..--- …-- ….- ….. -…. --… ---.. ----.”. As we can see in figure 16, the result of this message, this is correct.

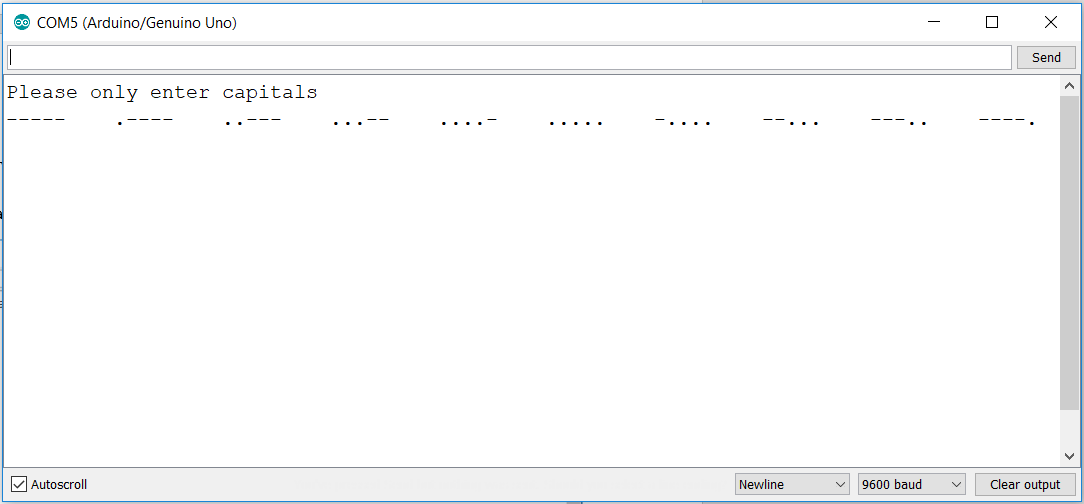


Figure 16 - Output for all numbers

The next thing to test after this is to ensure that a combination of numbers and letters in one message will work and is accurate. To do this, the message “H3LL0 W0RLD 123” was entered into the serial monitor, as seen in figure 17.

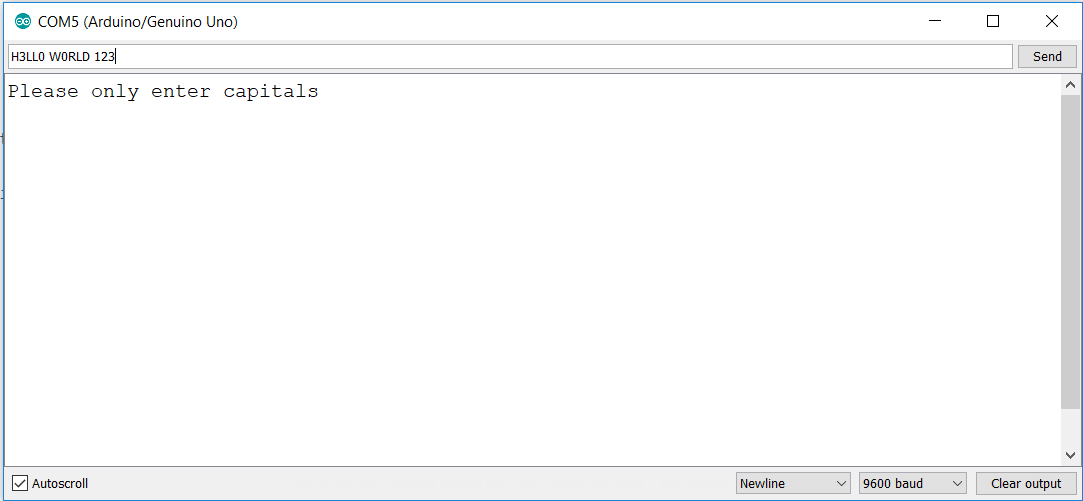


Figure 17 - Testing a combination of letters and numbers

Figure 18 shows the result of this message. From figure 4, we know that the program output should be “…. …-- .-.. .-.. ----- .-- ----- .-. .-.. -.. .---- ..--- …--“. When we compare the two, we can see that the program is once again correct and can cope with mixed messages.

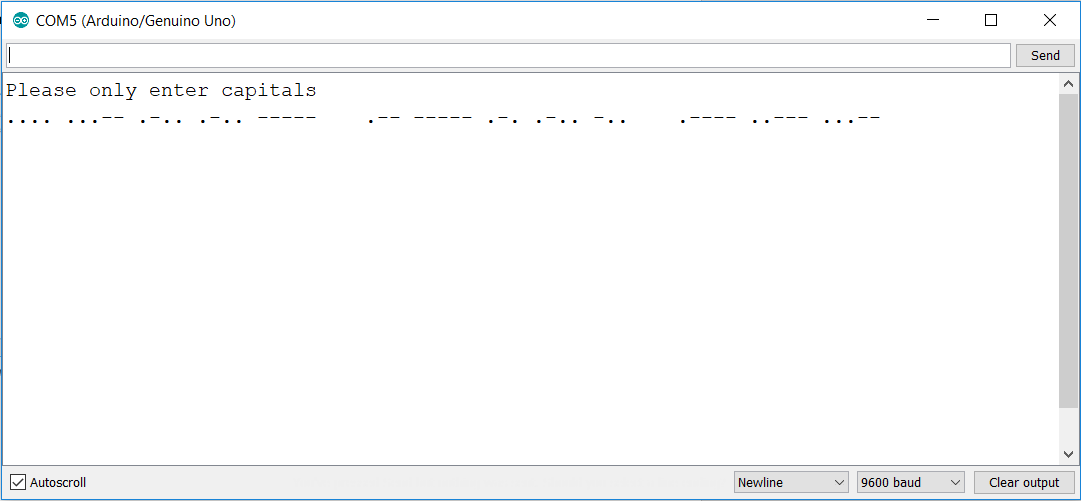


Figure 18 - The result of testing both letters and numbers

## Conclusion

The initial aims of this program were to:

• Translate entered words into Morse code

• Display this Morse code on an LED and using a Piezo

• Display the Morse code on the serial monitor of Arduino

I believe that during the testing process I have proved that my program can perform these functions to the degree that could be shown in this report. The program runs to all the time periods stated in the initial brief, and the translations produced are accurate to the messages input. To further develop this project, I would attempt to fix the capital letters only issue, either by inputting a conversion or just duplicating the case and switch with lower case letters. I would also input symbols such as “. , ? ! ‘ “ in order to enable more complex conversation, and try to develop a means of translating a Morse code input back into the English alphabet. It could also be developed further to work between two devices with some sort of Bluetooth system to enable encrypted conversation, as Morse code has previously been intended for. However, for the time span of this project I believe the project has reached the necessary potential and fulfils all the given aims.