# References

## **DIGITAL RAIN REFERENCES**

### First steps:

The first thing I tried to do was using the GoToXY function given in the lab to print a column of characters. I wanted the characters to be random so I used a random number generator function to generate integers in the range of 32-126, then converted that into the relevant ascii character to be printed. 0-31 and 127 aren’t printable ascii characters so those values were omitted. I used an arbitrary 23 character long column as it natively fit within the created terminal window.

On the random numGenerator, in the class Michelle outlined how the rand() function in C wasn’t great and that there was a more efficient way of doing it in C++ so I used that example instead.

After it prints the full column, it goes back to the top and reprints a whitespace “ “ character to remove the trail.

A computer screen with text

Description automatically generated

A black background with white letters

Description automatically generated

Once I had that working, the next step was to add colour. You can’t have matrix/digital rain without the classic green text, and I wanted to make the trail of characters green while keeping the current character white. StackOverflow came to the rescue with helpfully supplying the function I needed to create my vibrant green text. A slight modification of my printing function to print the (y – 1) character again but in green, and I was already half-way there. I realized at this point that because I was using GotoXY to control where I was printing, I didn’t need the “endl” tag in my prints so I also removed those as they were redundant.

A computer screen shot of code

Description automatically generated

<https://stackoverflow.com/questions/4053837/colorizing-text-in-the-console-with-c>

It’s looking very good now!

A black background with green letters and numbers

Description automatically generated

### Remove blinking cursor:

I wanted to remove the blinking cursor as it was quite distracting, I found this immediately on my initial internet search. By using the handle key for the StdHandle in the windows library, I was able to retrieve the console cursor information and change the cursor visible flag to FALSE which hides the cursor. Quick and easy, StackOverflow gave me the packaged function so it was easy to implement.  
<https://stackoverflow.com/questions/18028808/remove-blinking-underscore-on-console-cmd-prompt>

A screenshot of a computer code

Description automatically generated

### Threading for multiple “rain” drops:

The next step was to create multiple columns. I thought about it for awhile and figured trying to control rows and columns of different positions (I want the rain-drops to be in different vertical positions and not all falling at the same level) would get incredibly messy and I didn’t want to spend hours working with vectors inside vectors. In the lab Patrick mentioned he was using threads and I thought that was a great idea. If I could put each raindrop column in its own thread then I wouldn’t have to worry about managing multiple rows of information, I could theoretically copy-paste my existing raindrop function and utilize the GotoXY function to control where the printing and colour changing was occurring due to the specified coordinates.

Thankfully I already understood how threads work from my experience in Embedded Systems and my own interest in the field, so it was more understanding the syntax than the concept which made this much easier. However the threading still took some effort to get working as there were quite a few things I needed to learn with how C++ handles them. One of the C++ reference docs had a good example, plus I found a very good simple explanation of how they work in StackOverflow.

<https://cplusplus.com/reference/thread/thread/thread/>

<https://stackoverflow.com/questions/266168/simple-example-of-threading-in-c>

I also wanted to have the “user” define how many raindrops to create and because this directly impacted how many threads were made, I needed a way to create and join the threads without explicitly creating each handle myself. I decided to try implementing this using a vector as from our previous classes, “for each” loops work great in this scenario.

The examples I found used vector.push\_back() when creating threads within a vector and while looking for supporting documentation about this functionality, I found this article referenced below that went into depth on vector.emplace\_back() which also compared it with the vector.push\_back() option. The article highlighted that emplace\_back() was more efficient in placing the elements at the back of the vector as it created the element in-place within the vector, while push\_back() created the object first, then copied it to the function which seemed like unnecessary extra steps. If I was creating large amounts of raindrop threads, this could be problematic as I didn’t want it doing extra creations + copies for every thread.

<https://www.geeksforgeeks.org/vectoremplace_back-c-stl/>

### Mutex for thread-safe printing:

I noticed a printing bug when I created multiple threads of RainEffect, very noticeable especially with the colour option for previous characters. The currently printing character is white, and it sets the previous character (using GoToXY function for y – 1 and re-printing the previous white character as green) before continuing on. The console can only print one character at a time despite multiple threads so each thread was attempting to print and making it very messy. It broke the logical flow of the program to generate the column of characters too as seen in the below screenshot.

We covered mutexes and thread-safe operations in Embedded Systems with Niall and remembering how the printing function can be interrupted, I looked for mutex functionality in C++ so see if I could make the printing thread-safe between the multiple threads.

<https://en.cppreference.com/w/cpp/thread/mutex>

Bug:

A computer screen shot of a program code

Description automatically generated

I was surprised at how easy it was to implement, I found the mutex page in the CPPreference docs and just by using the included example I was able to implement it. It did take some playing around with restricting where the mutex was applied as it was **crashing the program** + making it **very slow**. **Solution:** Limiting the scope of where the mutex is enabled helped to reduce the wait time, and it removed the Abort() termination errors I was getting. I learned that the mutex lock\_guard only gives back the mutex when it leaves the scope of where it was called, so I had to wrap each mutex within its own parenthesis for each print function.

To be extra “thread” safe I included a std::this\_thread::yield() at the end of the RainEffect class to yield the thread so other threads can take over, as there was a small bug where the first thread was taking priority and stopping the other threads from fully completing their code block.

<https://en.cppreference.com/w/cpp/thread/lock_guard>