# C++ Project

## Concept of the Program

So the idea stemmed from a discussed problem that both I and an interviewer had whilst coding a messaging service for an embedded system. I thought this would be a fun little challenge to optimise in my spare time. It can also show a few potential skill

## The Problem

With embedded IoT systems an important feature is the ability to handle incoming messages from multiple sources and various different types of messages. The purpose of this program is to demonstrate handling of several different message types (will be eventually created by threads) and then queued and handled based on required protocol e.g., TCP, MQTT, BLE.

## Skills demonstrated in C++ (and OOP)

1. Abstract Class – inheritance, polymorphism, Abstraction, Encapsulation (set, get), difference between private, public and protected - constructor
2. Pure virtual functions – show understanding
3. Threads
4. Understanding of network topics
5. Queues
6. Data Parsing
7. Memory Management – New and Delete
8. Data Objects – Queues, arrays, vectors

## Walk through

The code is commented as it helps for me to go through as we go but here is a description of what things are and why it has been done.

### Classes

So first thing I am going to do is make a class in a separate cpp and h file. Now when putting class in the cpp

So when defining classes in separate files – all the definitions of members e.g. the data goes in the .h file whereas the cpp is where all the member functions go.

### Abstract Class

The messageClass is an abstract function – it is designed to never be called but be the basis of many other child classes that will define the message type (e.g. TCP, MQTT, BLE).

### Derived Classes

The two child message classes shown in this example is a MQTT message and a BLE message to show an active example of different IoT devices and the communications protocols. Each have their own unique properties for example an MQTT has a topic and may require different actions depending on that.

### Encapsulation

So with regards to the classes – each have private data – it is important that the members remain private so that only classes of that type can access them. If any sensitive data needs processing it should be done inside the class – i.e. inside the class encrypts/ decrypts data and places it as public member variable.

### Queues

Two functions BLE queue and MQTT queue as these may require separate processing, encrypting etc.

As the messages come in, we want to make sure that the Message Processing gets placed in the relative queues.

So as they come in we want to place the messages in the relative queues and then pop them off and on when the processing can occur The purpose of this is to queue messages as they come in and then we can process them in the correct way – this may mean certain time constraints.

So queues in c++ are data structures that operate on a first in first out (FIFO or first come first served) operation. The idea here is that we can get multiple messages at once and make sure they are queued in the right order – the queue will be a shared resource in memory. The idea is that the messages coming in might need to be processed in a certain order – e.g. data packets with an required output order.

We check for an empty queue and then process each queue message individually – this way we can have the queues in separate threads and do separate threads of processing.

### Data Structures and Incoming Messages

Inside the class the message is of a type vector (of strings). The size of the incoming data may vary and as a result a vector can dynamically allocated memory rather than depending on a fixed size as you do not know the size of the incoming data – the user may also need to do some processing, add other things to the data. So the ability to expand the data mid process (rather than declaring new arrays every time might be important).

### Threading

So here we are going to set up threads to create the classes and then add the messages to a relevant queue. This will simulate new incoming messages – to show the usability in a real life scenario.