### **1. Problem Statement:**

* The problem of data compression arises in various domains, from file storage to communication systems. Compressing messages reduces bandwidth usage and storage requirements, making it a useful skill to implement. This particular project likely addresses compression using Huffman coding—a widely-used algorithm.

### **2. Tech Stack:**

* **C++**:

C++ is very fast, which is important for compression and decompression tasks that need to be efficient. It also has built-in tools (like STL) that make it easy to manage data structures such as trees and heaps, which are necessary for Huffman encoding.

There’s a large C++ community with lots of tutorials and documentation, making it easy to find help if you run into problems.

C++ gives you direct control over memory and makes it easy to work with complex data structures, which is perfect for this project.

### **3. High-Level Diagram:**

The following steps represent the high-level process of how the project solves message compression and decompression:

* **Input Message**: Take the original message as input.
* **Frequency Analysis**: Analyze the frequency of characters in the message.
* **Build Huffman Tree**: Construct a Huffman Tree based on character frequencies.
* **Generate Codes**: Assign binary codes to each character using the Huffman Tree.
* **Compression**: Compress the message by replacing characters with their corresponding Huffman codes.
* **Decompression**: Use the Huffman Tree to decode the compressed message and restore the original message.

### **6. Challenges and Solutions:**

* **Memory Management**: Efficient memory handling for large messages and trees was achieved by carefully using pointers and managing heap memory in C++.
* **Encoding & Decoding**: Ensuring that both the encoding and decoding functions operate consistently was a challenge, resolved through extensive testing with different input messages.