Report

Program1

In program 1, we are asked to complete the implementation of the priority queue. The difference of this between the one shown on the slide is that the index starts from 0 rather than 1. Also, we use a queue without specific capacity in order to support automatic resizing. In addition, we add a comparator in order to let it be comparable according to the way the user want it to be compared. For test_pequeue, we have three unit tests for each kind of comparator to test its coverage.

Program 2

In the header file, we used the given GatBit function to implement ours GetChar and GetInt by the size of int and char, we used the binary operator left shift to see the specific bit of input is true or false (1 or 0). For the Put functions, I reuse the PutBit function as what I did in the GetBit. In the test case, I created four more test cases to see if the put function works. I first put bits, chars or ints and then use the get function to see if I get the right output as I put in the file. For the PubBit function, I have to put eight or its multiple number because of the size of buffer, but the char and int implementations will not have such problems because their size is already eight's multiples.

Program 3

For this program, in the compress part, we read the file char by char and calculate the frequency of each characters in to a map made of char and size. Then pass it to the pqueue we made under the restriction of (std::less). After that, we used the method from the prompt to form the tree we need. After that, we wrote the Read tree function to output the first part of our output. After that, we use the frequency of the top node to represent the second part of the output. After doing that, we wrote the function to follow the order of tree to find the trace of each node and then print that out in order.

In the decompress part, we decided to read the input file directly from the beginning of the file, we made a map to store the code (trace) for each characters and make a string to store the trace from the beginning. Once there is a '0', we add that to the string, once there is a '1', we also add it to string and the next 8 bits will be its char. After doing this, we sharing the number of 1 and 0s in the string to the last time of 1 's appearance and do the loop until the end. Then we use the map and the last part of the zap file to make the original file appear.