Quantization:

```
for (j = 0; j < height; j++) {
        for (i = 0; i < width; i++) {
            b = pInput[(i + j * width) * 3 + 0];     // Blue Color Component
            g = pInput[(i + j * width) * 3 + 1];     // Green Color Component
            r = pInput[(i + j * width) * 3 + 2];     // Red Color Component
            rgb16 = ((r >> 3) << 11) | ((g >> 2) << 5) | (b >> 3);
            quantizedImageData[(i + j * width) * 2 + 0] = rgb16 & 0xFF;
            quantizedImageData[(i + j * width) * 2 + 1] = (rgb16 >> 8) & 0xFF;
        }
    }
```

To quantize the 8-8-8 image to 5-6-5, first we extract the color component and save them in variable r,g,b. Then cut the lowest 3 bits in r and b, cut the lowest 2 bits in g. After this operation the 24 bits images is converted to 16bits.

Compression:

```
unsigned char *CAppCompress::Compress(int &cDataSize) {
    unsigned char *compressedData ;
    LAB lab;
    Tuple *C, *Cnext;
    lab.data = pInput;
    lab.sb_len = 4095;
    lab.lab_len = 15;
    lab.curr = 0;
    lab.len = width * height * 3;
    C = Encode_LZ77(lab);
    Cnext = C;
    int count = 0;
    unsigned char *tempData;
    tempData = new unsigned char[width * height * 6];
```

```
        while (Cnext) {
                tempData[count * 3] = ( Cnext->p << 4 |(Cnext->l)) & 0xFF;
                tempData[count * 3 + 1] = (Cnext->p >> 4) & 0xFF;
                tempData[count * 3 + 2] = Cnext->c & 0xFF;
                count++;
                Cnext = Cnext->next;
        }
        cDataSize = count * 3;
        compressedData = new unsigned char[cDataSize];
        for (int i = 0; i < cDataSize; i++)
                compressedData[i] = tempData[i];

        return compressedData ;              // return the compressed data
}
```

We use the LZ77 algorithm to compress the image. We find when the length of search buffer is 4096 and the length of look ahead buffer is 16, the compression ratio is better than other lengths. We compress all the color component and then save every token to 3 unsigned char variables. In our algorithm, the maximum of position variable(p) is 4095 and the maximum of length variable(l) is 15. So the first char variable save the lowest 4 bits of p and all bits in l. The second char save the highest 8 bits of p and the third char save the variable c. After this operation we successfully compress the image. And the decompress process is just transform the string data into token format, then decompress the token through Decode_LZ77 function.

Result:

Image				
Compression Ratio	**1.07**	**1.15**	**2.57**	**1.44**
Compression Ratio after Quantization	**2.32**	**3.04**	**4.00**	**2.66**

As we can see from the result, the compression ratio is not very high in complex image while it's high in the simple image such as the third image. After quantization and compression, the compression ratio can be very high. Even though the quantization part contributes 1.5 compression ratio, the compression ratio after two steps is higher than that just multiply them. The reason is that the algorithm LZ77 is good at compressing similar data in a continuous sequence. After quantization we cut the lowest 3 bits of the color component, which makes the maximum number of color component reduce from 256 to 32. With only 32 numbers, the possibility of repetition is higher and LZ77 can compress more data in a token.