**Problem 1:**

***Brief Explanation:***

For this problem, we had to use DES to encrypt a message about Sir Lewis Hamilton and ultimately decrypt the encrypted message to reproduce the message. First, we determined whether the program call used either -e or -d, which identified if the DES was using encryption or decryption. Once the appropriate algorithm is identified, a BitVector object is created with the file pointer created in Python. Thus, 64-bit blocks will be split accordingly with the read\_bits\_from\_file() function. We start analyzing the block if the block has a nonzero size. Furthermore, we account for padding of the BitVectors if the block is not 64 bits long by using the pad\_from\_right() function. Next, we carry out the Feistel Function by first splitting the 64-bit BitVector into 2 32-bit blocks (RE and LE, for right block and left block). We first expand the right block from 32 bits to 48 bits using expansion permutation. Using the result from this operation, we XOR the result with current round key. The result of the XORing operation is inputted into the substitute function to generate an s-box. Then, we do a permutation using the sbox BitVector object using the p-box permutation table as a parameter. This will help us generated the modified RE, and allow us to set LE to RE and RE to the modified RE. After all rounds are complete, we join the 2 32-bit BitVectors back to a 64-bit BitVector by concatenating RE with LE. Thus, we can write the resulting BitVector in hex format to the output file using a file pointer. This algorithm is the exact same in decryption. The major difference between encryption and decryption is that the round keys will be reversed. Other exceptions include reading the encrypted text file and constructing the BitVector object using the hexstring parameter. Because we are not using the filename when constructing the BitVector, we need to manually index the 64-bit blocks (bitvec = bv[i \* 64 : (i + 1) \* 64]) to go through each block. After these modifications, we carry out the rest of the operations just like in encryption. The resulting decryption creates a BitVector that will be rendered in ASCII format and written to the decrypted.txt file with the message mentioned earlier.

***Encrypted Output:*** \_image.py566b33456851c5966303a2053f6a22d41581fa810f1668eb7761db9206b466a8a65e50171f030c680a971cffd17e583060cd6e32ec5bd4ba1f9bda5976a883327bada116974b7e8220290949d5315cd4d308e297b7789bcf7466c433e6effef150ea4a44df492f449509044104c47b32351b272672fc599ea6926482920a08dd08cfdfdd19ae50585efebe84f51afbd7487e04b5e127457e37e615da2b55fafc317fecebf59a

***Decrypted Output:***

In the unforgiving world of Formula One, Lewis Hamilton abides at the top. He's the man to beat, the top earner, the most important voice, the most prominent figure  - a Black man alone at the summit of motorsports' highest echelon. England's knight in Mercedes armor. Over the past 15 years, the 36-year-old Briton has won seven world championships, tying the record set by Ferrari's Michael Schumacher  - the German F1 driver who was regarded as the greatest of all time until Hamilton broadsided him from that perch. At Sunday's Russian Grand Prix, Hamilton rallied through a late rain shower to claim the checkered flag on the way to becoming the first driver in the sport's history with 100 career victories. And that's besides his 100 career pole positions. As achievements go in racing, this is beyond otherworldly.

**Problem 2:**

***Brief Explanation:***

Encrypting an image using DES is like encrypting text files. The major idea in this problem is realizing that the PPM files are in binary format, so we need to read the input image file in binary mode. This also means the encrypted PPM file will be binary format. PPM files have a header that describes the attributes of the file, which take up 3 lines. Thus, we store the 3 lines into a variable and write them immediately after reading them in the output PPM file. Next, we construct a BitVector object using the filename attribute, and carry out the DES encryption algorithm using Feistel structures like what we did in Problem 1. After the rounds have been complete, we write the BitVector object to a file pointer that references to the output file.

***Encrypted PPM image:***

A picture containing fabric

Description automatically generated