Project Specification for DES Implementation

Objective: The goal is to implement the **Data Encryption Standard (DES)** algorithm using **C/C++** and a **Linux-based compiler (gcc/g++)**. The implementation must use bitwise operations and direct binary representation without relying on any external libraries or threads. It should handle encryption and decryption based on the command-line arguments, process files for input/output, and follow the specified key format.

Functional Requirements:

1. **DES Algorithm Implementation:**
   - **Initial Permutation (IP):**
     - The program must apply the **Initial Permutation** (IP) as defined in the DES standard, rearranging the bits of the 64-bit plaintext.
   - **Feistel Structure (16 Rounds):**
     - **16 Feistel Rounds:** The encryption process must consist of 16 rounds as described in the DES standard.
     - **Expansion (E) Permutation:** Expand the 32-bit right half of the data block to 48 bits using the **E expansion table**.
     - **Key Mixing:** XOR the expanded 48-bit right half with the 48-bit subkey for the current round.
     - **S-Box Substitution:** Use the **S-boxes** to substitute the 48-bit data block, resulting in a 32-bit output.
     - **Permutation (P):** Apply the **P permutation** to the output of the S-box substitution.
     - **Feistel Function:** XOR the result with the left half of the data block and swap the left and right halves after each round.
   - **Key Generation:**
     - **Permuted Choice 1 (PC1):** Apply **PC1** to the 64-bit key from the `key.txt` file, discarding 8 parity bits to obtain a 56-bit key.
     - **Key Splitting:** Divide the 56-bit key into two 28-bit halves.
     - **Key Shifts:** Perform **left circular shifts** based on the predefined shift schedule for each of the 16 rounds.
     - **Permuted Choice 2 (PC2):** After shifting, apply **PC2** to obtain the 48-bit round keys for the 16 rounds.
   - **Final Permutation (FP):**

- After completing the 16 Feistel rounds, apply the **Final Permutation (FP)** to obtain the 64-bit ciphertext.

2. **Command-line Interface:**
   - The program must be run via the command line with two options: `encrypt` or `decrypt`.
   - The program should not print any output to the console. Instead, it should read from and write to the specified files as shown below:
     - **Encryption:**
       `./team001 encrypt plain_text.txt key.txt cipher_text.dat`
       * `plain_text.txt`: The file containing the plaintext.
       * `key.txt`: A file containing the 64-bit key in hexadecimal format (e.g., `4A45B36C89948598`).
       * `cipher_text.dat`: The binary file to which the encrypted ciphertext should be written.
     - **Decryption:**
       `./team001 decrypt cipher_text.dat key.txt plain_text.txt`
       * `cipher_text.dat`: The binary file containing the ciphertext.
       * `key.txt`: A file containing the 64-bit key.
       * `plain_text.txt`: The file to which the decrypted plaintext should be written.

3. **Input/Output Files:**
   - **Input Key:** The key will be provided in a text file (`key.txt`) in **hexadecimal format** (e.g., `4A45B36C89948598`). The program should read this key and convert it to binary.
   - **Plaintext Input/Output:** The plaintext will be provided in a file (`plain_text.txt`), and the output ciphertext will be written to a file (`cipher_text.dat`) in **binary format**.
   - **Ciphertext Input/Output:** For decryption, the program will read the binary ciphertext from a file (`cipher_text.dat`), and the decrypted plaintext will be written to a file (`plain_text.txt`).

4. **Single File Implementation:**
   - All code must be written in a single `.cpp` file (e.g., `student_code.cpp`) without using any external header files or libraries.
   - The entire DES algorithm (including permutations, S-boxes, key generation, and encryption/decryption logic) must reside within this single file.

5. **Binary and Bitwise Operations:**
   - The implementation must use **direct binary representation** and **bitwise operations** to process the data.
   - No external libraries or threading should be used.

6. **No Console Output:**

- The program must not produce any console output. All input and output operations must be handled through files.

Non-Functional Requirements:

1. **Performance:**
   - The implementation should be efficient, leveraging bitwise operations and binary file handling to minimize overhead.
2. **Portability:**
   - The program must be compatible with **Linux** and compile using the `gcc/g++` compiler.
3. **Code Quality:**
   - Ensure consistent formatting and readable code, following good practices for naming variables and structuring functions.

Deliverables:

1. **Single `.cpp` File:**
   - The entire DES implementation must be in a single file named `student_code.cpp` (or `team001.cpp`, as per the naming convention provided).
   - This file should contain the complete implementation of DES encryption and decryption, including key generation, permutations, and file input/output.
2. **No Output to Console:**
   - The program must not print anything to the console. All output (plaintext or ciphertext) must be written to the appropriate files.
3. **Key File (`key.txt`):**
   - The provided key must be stored in a file in hexadecimal format.
4. **Submission:**
   - Submit the `.cpp` file directly to the LMS system as instructed.