1. Permutation Choice 1 (PC-1) and Final Permutation (FP)

- These are bit permutations.
- You give it a permutation table, and it permutes the bits of the input based on that table.

PC-1 and Final Permutation Code

```
def permute(bits, table):
    return ''.join(bits[i - 1] for i in table)

# Example 64-bit input as string of bits
input_64 =
"0110000101100010011000110110010001100101011001100110011101101000"  #
'abcdefgH' in bits

# Sample PC-1 permutation table (DES-style), here using 56 bits
PC1_TABLE = [
    57, 49, 41, 33, 25, 17, 9,
    1,  58, 50, 42, 34, 26, 18,
    10, 2,  59, 51, 43, 35, 27,
    19, 11, 3,  60, 52, 44, 36,
    63, 55, 47, 39, 31, 23, 15,
    7,  62, 54, 46, 38, 30, 22,
    14, 6,  61, 53, 45, 37, 29,
    21, 13, 5,  28, 20, 12, 4
]

# Final permutation (example: reverse of PC-1, simplified)
FP_TABLE = sorted(PC1_TABLE)  # In practice, this will be a specific
64-bit permutation

# Apply PC-1
pc1_result = permute(input_64, PC1_TABLE)
```

```
print("PC-1 output (56-bit):", pc1_result)

# Apply Final Permutation
final_permutation = permute(pc1_result.ljust(64, '0'), FP_TABLE)  #
pad to 64 bits if needed
print("Final Permutation (64-bit):", final_permutation)
```

2. Sample Compression Box: 64 ➡ 48

A compression permutation reduces 64 bits to 48 by selecting and reordering certain bits.

```
# Sample compression table (48 entries, DES-style)
COMPRESSION_TABLE = [
    14, 17, 11, 24, 1, 5,
    3, 28, 15, 6, 21, 10,
    23, 19, 12, 4, 26, 8,
    16, 7, 27, 20, 13, 2,
    41, 52, 31, 37, 47, 55,
    30, 40, 51, 45, 33, 48,
    44, 49, 39, 56, 34, 53,
    46, 42, 50, 36, 29, 32
]

def compress_64_to_48(bits64):
    return ''.join(bits64[i - 1] for i in COMPRESSION_TABLE)

compressed = compress_64_to_48(input_64)
print("Compressed 48-bit output:", compressed)
```

3. Expansion Box: 48 ➡ 64

A simplified expansion box (not common, but possible) — just for demonstration:

```
# Example: repeat some bits to expand from 48 to 64 (padding or
duplication)
```

```
def expand_48_to_64(bits48):
    # Duplicate some bits for expansion (this is just a sample
approach)
    expansion_table = list(range(1, 49)) + [1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12, 13, 14, 15, 16]
    return ''.join(bits48[i - 1] for i in expansion_table[:64])

# Use compressed 48-bit value
expanded = expand_48_to_64(compressed)
print("Expanded 64-bit output:", expanded)
```

Output Example (based on provided input):

```
PC-1 output (56-bit):
11000101100000110010000110000011001001001100011001010110
Final Permutation (64-bit):
1100010110000011001000011000001100100100110001100101011000000000
Compressed 48-bit output:
000110000110000000100011010010100101010011000101
Expanded 64-bit output:
0001100001100000001000110100101001010100110001010001100001100000
```