text to binary.
2. Apply hamming code concept on the binary data and add redundant bits to it.
3. Save the output in a file called channel.

Create a receiver program with below features

1. Receiver program should read the input from channel file.
2. Apply hamming code on the binary data to check for errors.
3. If there is an error, display the position of the error.
4. Else remove the redundant bits and convert the binary data to ascii and display the output

Program:-

```
def string_to_binary(input_string):
    return ''.join(format(ord(c), '08b')
                   for c in input_string)
def binary_to_string(binary_data):
    chars = []
    for i in range(0, len(binary_
        byte = binary_data[i:i+8]
        chars.append(chr(int(byte,2)))
    return ''.join(chars)

def calculate_parity_bits(data):
    n = len(data)
    r = 0
    while(2**r) < (n+r+1):
        r += 1
    return r

def insert_parity_bits(data, r):
    n = len(data)
    j = 0
    k = 0
    m = n+r
    hamming_code = []

for i in range(1, m+1):
    if i == 2**j:
```

```
        hamming_code.append(0)
        j+=1
    else:
        hamming_code.append(int(data[k]))
        k+=1
    return hamming_code.
def calculate_parity_values(hamming_code,r):
    n=len(hamming_code)
    for i in range(r):
        parity_pos = 2**i
        parity_val=0
        for j in range(1,n+1):
            if j & parity_pos and j != parity_pos:
                parity_val ^= hamming_code[j-1]
        hamming_code[parity_pos-1] = parity_val
    return hamming_code.
def detect_and_correct_error(hamming_code,r):
    n = len(hamming_code)
    error_position=0
    for i in range(r):
        parity_pos = 2**i
```

```
        parity_val = 0
        for j in range (1, n+1):
            if j & parity_pos:
                parity_val ^= hamming_code
                                [j-1]

        if parity_val != 0:
            error_position += parity_pos

    if error_position:
        print(f" Error detected at position:
                        {error_position}")
        hamming_code[error_position-1] ^= 1
        print(f" Corrected Hamming code:
                        {hamming_code}")

    else:
        print(" No error detected.")
    return hamming_code.

def extract_data_from_hamming
                    (hamming_code, r):
    j = 0
    data = []
for i in range(1, len(hamming_code)
                        +1):
    if i != 2**j:
        data.append(hamming_code[i-1])
```

```
else:
    j += 1
return ''.join(map(str, data))

def main():
    input_string = "jayashree"
binary_data = string_to_binary(input_
                               string)
print(f"Binary representation of
      {input_string}: {binary_data}")
r = calculate_parity_bits(binary_data)
hamming_code = insert_parity_bits
               (binary_data, r)
hamming_code = calculate_parity_values
               (hamming_code, r)
print(f"Hamming Code with parity
      bits: {hamming_code}")
print("\n Introducing a single-bit
      error for demonstration...")
while True:
    try:
        error_bit = int(input(f"Enter the
            bit position (1-{len
(hamming_code)}) to introduce an
                error: "))
```

```
if error_bit & (error_bit - 1) == 0:
    raise ValueError("Error bit position
Cannot be a power of 2. Please enter
a valid position.")
    break
except ValueError as e:
    print(e)
hamming_code = Calculate_parity_
               values(hamming_code, r)
print(f"Hamming Code with parity
      bits: {hamming_code}")
print("\nIntroducing a single-bit
      error for demonstration...")
while True:
  try:
    error_bit = int(input(f"Enter
    the bit position (1-{len(hamming_
    code)}) to introduce an error: "))
if error_bit & (error_bit - 1) == 0:
    raise ValueError("Error bit
    position cannot be a power of
    2. Please enter a valid
    position.")
```

```
        break
    except ValueError as e:
        print(e)
hamming_code[error_bit - 1] ^= 1
print(f"Hamming Code with
        error : {hamming_code}")
hamming_code = detect_and_correct_
        error(hamming_code, r)

corrected_binary_data = extract_
        data_from_hamming
    (hamming_code, r)

corrected_string = binary_to_string
        (corrected_binary_data)

print(f"Final output after
    Correcting Hamming Code:
        {corrected_string}")

if __name__ == "__main__":
    main()
```

output:

Binary representation of 'Jayashree':

01101010011000010111001011000010
11100110110100001110010011001010110010

Error!

Hamming Code with parity bits:

[0,0,0,1,1,0,0,1,0,1,0,0,1,1,0,0,0,
0,0,1,0,1,1,1,1,0,0,1,0,1,0,1,0,0,
0,0,1,0,1,1,1,1,0,0,1,0,1,0,0,0,0,
1,1,1,0,0,1,0,0,0,1,1,0,0,1,0,
1,0,1,1,0,0,1,0,1]

Introducing a single-bit error for demonstration...

Enter the bit position (1-79) to introduce an error: 5

Error!

Hamming Code with error: [0,0,0,
0,0,1,0,0,1,0,1,0,0,1,1,0,0,0,0,0,1
,0,1,1,1,1,0,0,1,0,1,0,0,0,0,1,0,
1,1,1,0,0,1,1,0,1,1,0,1,0,0,0,0,1,1,
1,0,0,1,0,0,0,1,1,0,0,1,0,1,0,1,1,0
0,1,0,1]

error detected at position: 5

Corrected Hamming Code: [0,0,0,0,1,1,
0,0,1,0,1,0,0,1,1,0,0,0,0,0,1,0,1,
1,1,0,0,1,0,1,0,1,0,0,0,0,0,1,0,1,1,1,0,
0,0,1,1,0,1,1,0,1,0,0,0,0,1,1,1,0,0,
1,0,0,0,1,1,0,0,1,0,1,0,1,1,0,0,1,0,1]

final output after correcting Hamming Code: 'jayashree'.

20/8/24

Result:

Thus the program is Successfully executed and verified.