EXP:06

Error Correction at Data Link Layer

Aim

To write a program to implement error detection and correction using the **Hamming Code** concept.

Algorithm / Procedure

1. **Determine** the number of redundant (parity) bits required for the given data size.
2. **Calculate** the positions of the parity bits (powers of 2).
3. **Implement** the Hamming Code generation algorithm:
   - Place data bits and parity bits in their respective positions.
   - Calculate the value of each parity bit based on the data bits it covers.
4. **Implement** the error detection and correction algorithm:
   - Receive the transmitted codeword.
   - Recalculate the parity bits.
   - Calculate the syndrome (error position) by combining the recalculated parity bits.
   - If the syndrome is non-zero, flip the bit at the error position.
5. **Test** the program with a data stream, introducing a single-bit error to verify the correction feature.

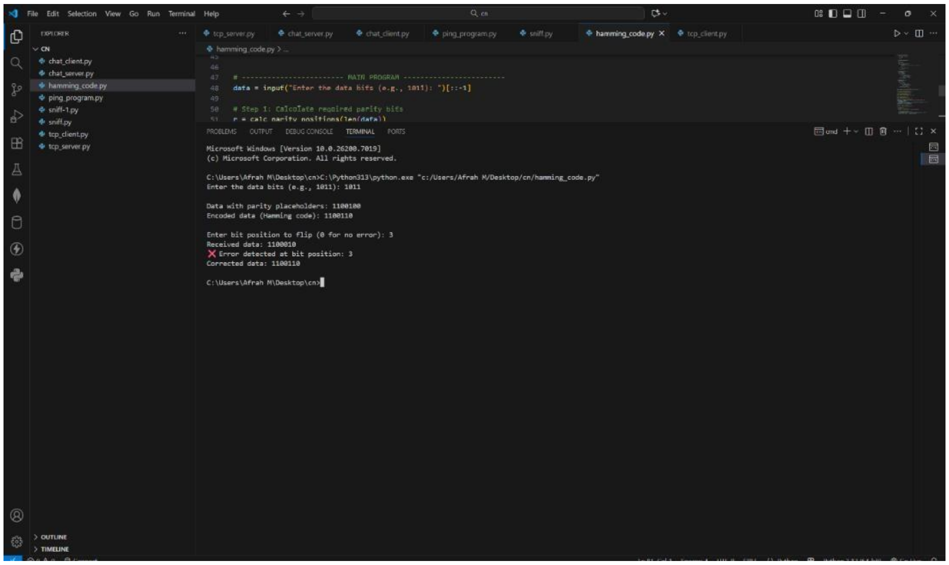
Code:

```
def calc_parity_positions(m): r
  = 0
  while (2**r) < (m + r + 1): r
    += 1
  return r


def insert_parity_bits(data, r):
  j = 0
  k = 1
  m = len(data)
  res = ''
  for i in range(1, m + r + 1):
    if i == 2**j:
      res += '0'  # parity bits start as 0 instead of 'P' j
      += 1
    else:
      res += data[-1 * k]
      k += 1
  return res[::-1]
```

```
def calc_parity_bits(arr, r):
    n = len(arr)
    arr = list(arr)
    for i in range(r):
        val = 0

        for j in range(1, n + 1):
            if j & (2**i) ==
            (2**i):
                val ^= int(arr[-1 * j])
        arr[-1 * (2**i)] = str(val)
    return ''.join(arr)



def detect_error(arr, r):
    n = len(arr)
    res = 0

    for i in range(r):
        val = 0
        for j in range(1, n + 1):
            if j & (2**i) ==
            (2**i):
                val ^= int(arr[-1 * j])
        res += val * (10**i)
    return int(str(res)[::-1], 2)


# ------------------------ MAIN PROGRAM ------------------------
data = input("Enter the data bits (e.g., 1011): ")[::-1]

# Step 1: Calculate required parity bits r
= calc_parity_positions(len(data))

# Step 2: Insert parity bits into data
arr = insert_parity_bits(data, r)
print("\nData with parity placeholders:", arr)

# Step 3: Calculate parity bits
arr = calc_parity_bits(arr, r)
print("Encoded data (Hamming code):", arr)

# Step 4: Introduce an error (optional)
error_index = int(input("\nEnter bit position to flip (0 for no error): "))
arr_with_error = list(arr)
if error_index != 0:
    arr_with_error[-error_index] = '1' if arr_with_error[-error_index] == '0' else '0' arr_with_error = '
    '.join(arr_with_error)
print("Received data:", arr_with_error)
```

```
# Step 5: Detect error position
error_pos = detect_error(arr_with_error, r)
if error_pos == 0:
    print("No error detected.")
else:
    print(f"Error detected at bit position: {error_pos}")

 # Step 6: Correct the error
 arr_corrected = list(arr_with_error)
 arr_corrected[-error_pos] = '1' if arr_corrected[-error_pos] == '0' else '0'
 print("Corrected data:", ".join(arr_corrected))
```

Output:



```
# ----------------------- MAIN PROGRAM -----------------------
data = input("Enter the data bits (e.g., 1011): ")[::-1]

# Step 1: Calculate required parity bits
r = calc_parity_positions(len(data))
```

```
Microsoft Windows [Version 10.0.26200.7019]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Afrah M\Desktop\cn>C:\Python313\python.exe "c:/Users/Afrah M/Desktop/cn/hamming_code.py"
Enter the data bits (e.g., 1011): 1011

Data with parity placeholders: 1100100
Encoded data (Hamming code): 1100110

Enter bit position to flip (0 for no error): 3
Received data: 1100010
❌ Error detected at bit position: 3
Corrected data: 1100110

C:\Users\Afrah M\Desktop\cn>
```

Result:

A program to implement the Hamming Code was successfully written. The program demonstrated the ability to detect and correct a single-bit error in the transmitted data stream.