1)parity

```
#include <stdio.h>

int main() {
    int num, type, ones = 0;

    printf("Enter the bit pattern (as an integer): ");
    scanf("%d", &num);

    printf("Enter parity type (0 for even, 1 for odd): ");
    scanf("%d", &type);

    int temp = num;
    while (temp) {
        if (temp % 2 == 1) ones++;  // count 1s
        temp /= 2;
    }

    int parity = (type == 0) ? (ones % 2) : !(ones % 2);
    printf("Generated Parity Bit: %d\n", parity);

    int sent = (num << 1) | parity;
    printf("Transmitted Data (Including Parity Bit): %d\n", sent);

    int recv_data = sent >> 1;
    int recv_parity = sent & 1;

    ones = 0;
    temp = recv_data;
    while (temp) {
        if (temp % 2 == 1) ones++;
        temp /= 2;
    }
    ones += recv_parity;

    int ok = (type == 0) ? (ones % 2 == 0) : (ones % 2 != 0);
    printf(ok ? "No error detected.\n" : "Error detected in transmission!\n");

    return 0;
}
```

---

3)hamming code

```
#include <stdio.h>

// Insert parity bits at correct positions
void generateHammingCode(int data[], int m, int hamming[], int r) {
    int i, j = 0, k = 0;

    for (i = 1; i <= m + r; i++) {
        if ((i & (i - 1)) == 0)  // Power of 2 = parity bit
            hamming[i - 1] = 0;
        else
            hamming[i - 1] = data[j++];
    }

    // Calculate parity bits
    for (i = 0; i < r; i++) {
        int pos = 1 << i;
        int parity = 0;
        for (j = pos - 1; j < m + r; j++) {
            if (((j + 1) & pos) != 0)
```

```
                parity ^= hamming[j];
            }
            hamming[pos - 1] = parity;
        }
    }

    // Detect and correct single-bit error
    void detectAndCorrect(int hamming[], int size, int r) {
        int i, errorPos = 0;

        for (i = 0; i < r; i++) {
            int pos = 1 << i;
            int parity = 0;
            for (int j = 0; j < size; j++) {
                if ((j + 1) & pos)
                    parity ^= hamming[j];
            }
            if (parity != 0)
                errorPos += pos;
        }

        if (errorPos == 0)
            printf("No error detected.\n");
        else {
            printf("Error at position %d. Correcting it...\n", errorPos);
            hamming[errorPos - 1] ^= 1;
        }
    }

    // Print Hamming Code
    void printCode(int code[], int size) {
```

```c
    printf("Hamming Code: ");
    for (int i = size - 1; i >= 0; i--)
        printf("%d ", code[i]);
    printf("\n");
}

int main() {
    int m;
    printf("Enter number of data bits: ");
    scanf("%d", &m);

    int r = 0;
    while ((1 << r) < m + r + 1)
        r++;

    int data[m];
    printf("Enter %d data bits (MSB to LSB): ", m);
    for (int i = 0; i < m; i++)
        scanf("%d", &data[i]);

    int totalBits = m + r;
    int hamming[totalBits];

    generateHammingCode(data, m, hamming, r);

    printf("\nGenerated Hamming Code:\n");
    printCode(hamming, totalBits);

    int error;
    printf("\nEnter position (1 to %d) to introduce an error, or 0 for none: ", totalBits);
    scanf("%d", &error);
    if (error > 0 && error <= totalBits) {
        hamming[error - 1] ^= 1;
        printf("Error introduced at position %d!\n", error);
    }

    printf("\nReceived Hamming Code:\n");
    printCode(hamming, totalBits);

    detectAndCorrect(hamming, totalBits, r);

    printf("\nCorrected Hamming Code:\n");
    printCode(hamming, totalBits);

    return 0;
}
```

---

8)
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

int is_prime(long int num) {
    if (num < 2) return 0;
    for (int i = 2; i <= sqrt(num); i++) {
        if (num % i == 0)
            return 0;
    }
    return 1;
}
```

```
void compute_keys(long int p, long int q, long int e[], long int d[], int *count, long int *n, long int *t) {
    *n = p * q;
    *t = (p - 1) * (q - 1);
    int k = 0;

    for (long int i = 2; i < *t; i++) {
        if (*t % i == 0) continue;
        if (is_prime(i) && i != p && i != q) {
            e[k] = i;
            long int k1 = 1;
            while (1) {
                k1 += *t;
                if (k1 % e[k] == 0) {
                    d[k] = k1 / e[k];
                    break;
                }
            }
            k++;
        }
    }
    *count = k;
}

void encrypt(char msg[], long int e, long int n, long int temp[], long int en[]) {
    printf("\nTHE ENCRYPTED MESSAGE IS:\n");
    int i = 0;
    while (msg[i] != '\0') {
        long int pt = msg[i] - 96;
        long int k = 1;
        for (int j = 0; j < e; j++) {
            k = (k * pt) % n;
        }
        temp[i] = k;
        long int ct = k + 96;
        en[i] = ct;
        printf("%c", (char)ct);
        i++;
    }
    en[i] = -1;
}

void decrypt(long int temp[], long int en[], long int d, long int n) {
    printf("\n\nTHE DECRYPTED MESSAGE IS:\n");
    int i = 0;
    while (en[i] != -1) {
        long int ct = temp[i];
        long int k = 1;
        for (int j = 0; j < d; j++) {
            k = (k * ct) % n;
        }
        long int pt = k + 96;
        printf("%c", (char)pt);
        i++;
    }
    printf("\n");
}

int main() {
    long int p, q, n, t;
    long int e[100], d[100], temp[100], en[100];
    int count;
    char msg[100];

    printf("ENTER FIRST PRIME NUMBER: ");
    scanf("%ld", &p);
    if (!is_prime(p)) {
        printf("WRONG INPUT\n");
        return 1;
    }

    printf("ENTER ANOTHER PRIME NUMBER: ");
    scanf("%ld", &q);
    if (!is_prime(q) || p == q) {
        printf("WRONG INPUT\n");
        return 1;
    }

    printf("ENTER MESSAGE (lowercase only): ");
    scanf("%s", msg);

    compute_keys(p, q, e, d, &count, &n, &t);

    printf("\nPOSSIBLE VALUES OF e AND d ARE:\n");
    for (int i = 0; i < count; i++)
        printf("%ld\t%ld\n", e[i], d[i]);

    encrypt(msg, e[0], n, temp, en);
    decrypt(temp, en, d[0], n);

    return 0;
}
```