```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>

// Function to compute (a^b) % mod using fast exponentiation
int mod_exp(int base, int exp, int mod) {
    int result = 1;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % mod;
        }
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}

// Function to check if a number is primitive in F_p
bool is_primitive(int g, int p) {
    for (int i = 1; i < p - 1; i++) {
        if (mod_exp(g, i, p) == 1) {
            return false;
        }
    }
    return true;
}

// Find a primitive element in F_p
int find_primitive(int p) {
    for (int g = 2; g < p; g++) {
```

```
        if (is_primitive(g, p)) {
            return g;
        }
    }
    return -1;
}

// Polynomial multiplication modulo an irreducible polynomial
int poly_mult_mod(int a, int b, int mod_poly, int p) {
    int result = 0;
    while (b) {
        if (b & 1) {
            result ^= a;
        }
        b >>= 1;
        a <<= 1;
        if (a & (1 << 4)) {  // Assuming degree 4 field extension
            a ^= mod_poly;
        }
    }
    return result;
}

// Find a primitive element in F_{p^m}
bool is_primitive_ext(int g, int mod_poly, int p, int field_size) {
    int value = g;
    for (int i = 1; i < field_size - 1; i++) {
        value = poly_mult_mod(value, g, mod_poly, p);
        if (value == 1) return false;
    }
    return true;
```

```
}

int find_primitive_ext(int mod_poly, int p, int field_size) {
    for (int g = 2; g < field_size; g++) {
        if (is_primitive_ext(g, mod_poly, p, field_size)) {
            return g;
        }
    }
    return -1;
}

// Check if a polynomial is irreducible in F_2[x]
bool is_irreducible(int poly, int degree) {
    for (int div = 2; div < (1 << (degree / 2 + 1)); div++) {
        int remainder = poly;
        int divisor = div;
        while (remainder >= divisor) {
            int shift = (int)log2(remainder) - (int)log2(divisor);
            remainder ^= (divisor << shift);
        }
        if (remainder == 0) {
            return false;
        }
    }
    return true;
}

// Find an irreducible monic polynomial of given degree over F_2
int find_irreducible_monic(int degree) {
    for (int poly = (1 << degree) | 1; poly < (1 << (degree + 1)); poly += 2) {
        if (is_irreducible(poly, degree)) {
```

```
            return poly;
        }
    }
    return -1;
}

int main() {
    int p = 7;  // Prime field F_p
    printf("Primitive element in F_%d: %d\n", p, find_primitive(p));

    int mod_poly = 0b10011; // x^4 + x + 1 (irreducible over F_2)
    int field_size = 16;  // 2^4
    printf("Primitive element in F_%d^4: %d\n", p, find_primitive_ext(mod_poly, 2, field_size));

    int degree = 4;
    int irreducible_poly = find_irreducible_monic(degree);
    printf("Irreducible monic polynomial of degree %d over F_2: %04b\n", degree, irreducible_poly);

    return 0;
}
```