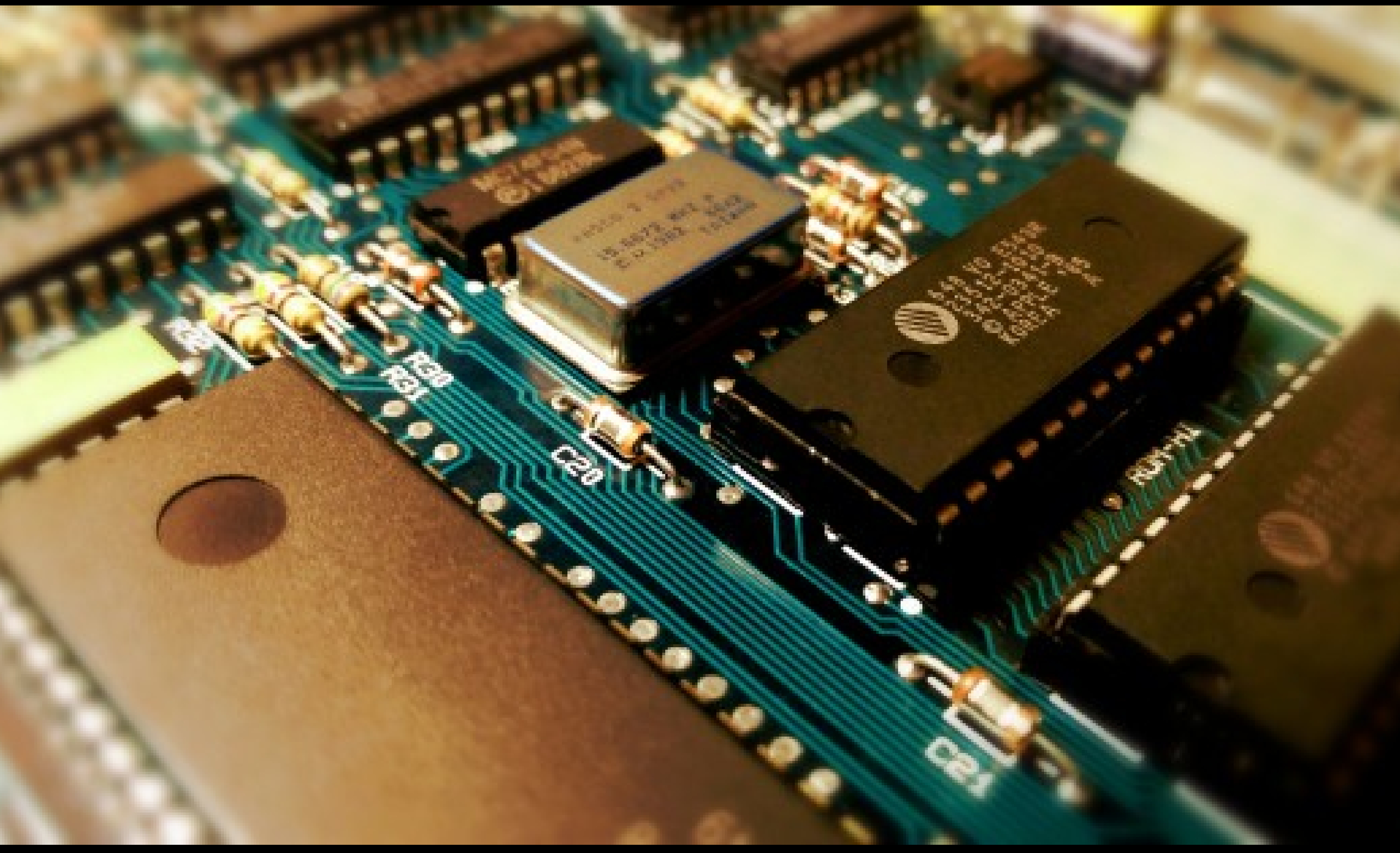
PROV LOGIC

ADVANCED C BASED PROGRAMMING INTERVIEW QUESTIONS

Real-world problems asked by Apple, AMD, NVIDIA, ARM, and Google

Targeted for VLSI, Embedded Systems, and Systems Programming Roles



Includes 50 hand-picked problems with fully working solutions, edge case handling, and optimized code for hardware-critical roles.

Prasanthi Chanda

1. Find the Position of Rightmost Set Bit (No Loops)

Solution:

```
int rightmostSetBit(int n) {
    return n & -n;
}
```

2. Count Number of 1s in Binary Using Lookup Table

Solution:

```
unsigned char table[256];

void initialize() {
    for (int i = 0; i < 256; i++)
        table[i] = (i & 1) + table[i >> 1];
}

int countSetBits(unsigned int x) {
    return table[x & 0xff] + table[(x >> 8) & 0xff] +
table[(x >> 16) & 0xff] + table[(x >> 24)];
}
```

3. Atomic Toggle of GPIO Bit (Memory-Mapped Register)

Solution:

```
#define GPIO_REG (*(volatile uint32_t*)0x40011000)

void atomicToggle(int bit) {
    __disable_irq();
    GPIO_REG ^= (1 << bit);
    __enable_irq();
}
```

4. Custom Fixed-Size Memory Allocator

Solution:

```
#define BLOCK_SIZE 32
#define TOTAL_BLOCKS 100

char memory[TOTAL_BLOCKS * BLOCK_SIZE];
char used[TOTAL_BLOCKS] = {0};

void* my_malloc() {
    for (int i = 0; i < TOTAL_BLOCKS; i++) {
        if (!used[i]) {
            used[i] = 1;
            return &memory[i * BLOCK_SIZE];
        }
    }
    return NULL;
}

void my_free(void* ptr) {
    int index = ((char*)ptr - memory) / BLOCK_SIZE;
    used[index] = 0;
}
```

5. Reverse Endianness of 32-bit Integer

Solution:

```
uint32_t reverseEndian(uint32_t x) {
    return ((x >> 24) & 0x000000FF) |
           ((x >> 8)  & 0x0000FF00) |
           ((x << 8)  & 0x00FF0000) |
           ((x << 24) & 0xFF000000);
}
```

6. Iterative Inorder Traversal of Binary Tree

Solution:

```
void inOrder(struct Node* root) {
    struct Node* stack[100];
    int top = -1;

    while (root || top != -1) {
        while (root)
            stack[++top] = root, root = root->left;

        root = stack[top--];
        printf("%d ", root->data);
        root = root->right;
    }
}
```

7. Circular Buffer for Byte Storage

Solution:

```
#define SIZE 256
uint8_t buffer[SIZE];
int head = 0, tail = 0;

void writeByte(uint8_t data) {
    int next = (head + 1) % SIZE;
    if (next != tail) {
        buffer[head] = data;
        head = next;
    }
}
```

```
uint8_t readByte() {
    if (head == tail) return 0; // Buffer empty
    uint8_t data = buffer[tail];
    tail = (tail + 1) % SIZE;
    return data;
}
```

8. Integer Square Root Using Bit Manipulation

Solution:

```
unsigned int intSqrt(unsigned int n) {
    unsigned int res = 0, bit = 1 << 30;
    while (bit > n)
        bit >>= 2;
    while (bit) {
        if (n >= res + bit) {
            n -= res + bit;
            res = (res >> 1) + bit;
        } else {
            res >>= 1;
        }
        bit >>= 2;
    }
    return res;
}
```

9. Fast Exponentiation (Binary Method)

Solution:

```
int power(int base, int exp) {
    int result = 1;
    while (exp > 0) {
```

```
    if (exp & 1)
        result *= base;
    base *= base;
    exp >>= 1;
}
return result;
}
```

10. Detect Loop in a Singly Linked List (Floyd's Algorithm)

Solution:

```
struct Node {
    int data;
    struct Node* next;
};

int hasLoop(struct Node* head) {
    struct Node *slow = head, *fast = head;
    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast) return 1;
    }
    return 0;
}
```

11. Implement memcpy() with Overlap Handling

Solution:

```
void* my_memcpy(void* dest, const void* src, size_t
n) {
    char* d = dest;
    const char* s = src;
```

```
if (d < s) {
    while (n--) *d++ = *s++;
} else {
    d += n;
    s += n;
    while (n--) *--d = *--s;
    }
    return dest;
}
```

12. Implement itoa() Function Without Using Standard Library

Solution:

```
void reverse(char* str, int len) {
    int i = 0, j = len - 1;
    while (i < j) {
        char tmp = str[i];
        str[i++] = str[j];
        str[j--] = tmp;
    }
}

char* itoa(int num, char* str, int base) {
    int i = 0, isNegative = 0;
    if (num == 0) {
        str[i++] = '0';
        str[i] = '\0';
        return str;
}
```

```
if (num < 0 && base == 10) {
    isNegative = 1;
    num = -num;
}

while (num) {
        int rem = num % base;
        str[i++] = (rem > 9) ? (rem - 10) + 'A' : rem +
'0';
        num /= base;
    }

    if (isNegative)
        str[i++] = '-';

    str[i] = '\0';
    reverse(str, i);
    return str;
}
```

13. Count Number of Trailing Zeros in Integer

Solution:

```
int countTrailingZeros(int n) {
    if (n == 0) return 32;
    int count = 0;
    while ((n & 1) == 0) {
        count++;
        n >>= 1;
    }
    return count;
}
```

14. Implement Thread-Safe Singleton in C (Without C++)

Solution:

```
typedef struct {
    int data;
} Singleton;

Singleton* getInstance() {
    static Singleton* instance = NULL;
    static pthread_mutex_t lock =
PTHREAD_MUTEX_INITIALIZER;

    if (!instance) {
        pthread_mutex_lock(&lock);
        if (!instance) {
            instance = malloc(sizeof(Singleton));
            instance->data = 42;
        }
        pthread_mutex_unlock(&lock);
    }
    return instance;
}
```

15. Bitwise Multiply Without * Operator

Solution:

```
int multiply(int a, int b) {
    int result = 0;
    while (b) {
```

```
        if (b & 1) result += a;
        a <<= 1;
        b >>= 1;
    }
    return result;
}
```

16. Implement Bit Rotation (Left and Right)

Solution:

```
uint32_t rotateLeft(uint32_t n, unsigned int d) {
    return (n << d) | (n >> (32 - d));
}

uint32_t rotateRight(uint32_t n, unsigned int d) {
    return (n >> d) | (n << (32 - d));
}
```

17. Find the Missing Number in an Array of 0 to n (XOR Method)

Solution:

```
int findMissing(int arr[], int n) {
    int res = 0;
    for (int i = 0; i < n; i++) res ^= arr[i];
    for (int i = 0; i <= n; i++) res ^= i;
    return res;
}
```

18. Set, Clear, Toggle Specific Bit

Solution:

```
unsigned int setBit(unsigned int n, int k) {
    return n | (1U << k);
}

unsigned int clearBit(unsigned int n, int k) {
    return n & ~(1U << k);
}

unsigned int toggleBit(unsigned int n, int k) {
    return n ^ (1U << k);
}
```

19. Simulate Mutex Using Test-And-Set (Spinlock)

Solution:

```
volatile int lock = 0;

int test_and_set(int *lock) {
    int old = *lock;
    *lock = 1;
    return old;
}
void acquire_lock() {
    while (test_and_set((int *)&lock));
}
void release_lock() {
    lock = 0;
}
```

20. Convert Float to Binary Without Using Union

Solution:

```
void floatToBinary(float f, char* out) {
    uint32_t asInt;
    memcpy(&asInt, &f, sizeof(float));
    for (int i = 31; i >= 0; i--)
        *out++ = (asInt & (1U << i)) ? '1' : '0';
    *out = '\0';
}
```

21. Lock-Free Stack (LIFO) Using Compare-And-Swap

Solution:

```
typedef struct Node {
    int data;
    struct Node* next;
} Node;

_Atomic(Node*) top = NULL;

void push(int val) {
    Node* newNode = malloc(sizeof(Node));
    newNode->data = val;
    Node* oldTop;
    do {
        oldTop = atomic_load(&top);
        newNode->next = oldTop;
    } while (!atomic_compare_exchange_weak(&top,
&oldTop, newNode));
}
```

```
int pop() {
    Node* oldTop;
    Node* newTop;
    do {
        oldTop = atomic_load(&top);
        if (oldTop == NULL) return -1;
        newTop = oldTop->next;
    } while (!atomic_compare_exchange_weak(&top,
&oldTop, newTop));
    int val = oldTop->data;
    free(oldTop);
    return val;
}
```

22. Fast Inverse Square Root (from Quake III Engine)

Solution:

```
float Q_rsqrt(float number) {
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y = number;
    i = *(long*)&y;
    i = 0x5f3759df - (i >> 1);  // initial guess
    y = *(float*)&i;
    y = y * (threehalfs - (x2 * y * y));  // Newton step
    return y;
}
```

23. Detect Stack Corruption with Canary (Stack Smashing Protector-like)

Solution:

```
#define CANARY_VALUE 0xDEADBEEF

void safe_function() {
    unsigned int canary = CANARY_VALUE;
    char buffer[64];
    // do operations on buffer...
    if (canary != CANARY_VALUE) {
        printf("Stack corruption detected!\n");
        exit(1);
    }
}
```

24. Efficient CRC32 Calculation (Polynomial 0xEDB88320)

Solution:

```
uint32_t crc32(const void* data, size_t n_bytes) {
    uint32_t crc = ~0U;
    const uint8_t* p = data;
    while (n_bytes--) {
        crc ^= *p++;
        for (int k = 0; k < 8; k++)
            crc = (crc >> 1) ^ (0xEDB88320 & -(crc & 1));
    }
    return ~crc;
}
```

25. Convert IEEE 754 Float to Decimal String Without sprintf()

Solution:

```
char* floatToString(float f, char* buffer) {
    int whole = (int)f;
    float frac = f - whole;
    int frac_scaled = (int)(frac * 10000);
    sprintf(buffer, "%d.%04d", whole, abs(frac_scaled));
    return buffer;
}
```

26. Custom Memory Allocator from Fixed Pool

Solution:

```
#define POOL_SIZE 1024
char mem_pool[POOL_SIZE];
char* mem_head = mem_pool;

void* pool_alloc(size_t size) {
    if (mem_head + size > mem_pool + POOL_SIZE)
        return NULL;
    void* p = mem_head;
    mem_head += size;
    return p;
}
```

27. Detect Integer Multiplication Overflow

Solution:

```
int mul_with_overflow_check(int a, int b, int* result) {
    *result = a * b;
    if (a != 0 && *result / a != b)
        return 1; // overflow
    return 0;
}
```

28. Write Your Own memcpy() Optimized with 4-byte Blocks

Solution:

```
void* fast_memcpy(void* dest, const void* src, size_t
n) {
    uint32_t* d = dest;
    const uint32_t* s = src;

    while (n >= 4) {
        *d++ = *s++;
        n -= 4;
    }

    uint8_t* db = (uint8_t*)d;
    const uint8_t* sb = (const uint8_t*)s;
    while (n--) {
        *db++ = *sb++;
    }
    return dest;
}
```

29. Reverse a Singly Linked List Using Recursion (No Loop Allowed)

Solution:

```
typedef struct Node {
    int data;
    struct Node* next;
} Node;
```

```
Node* reverse(Node* head) {
    if (!head || !head->next) return head;
    Node* rest = reverse(head->next);
    head->next->next = head;
    head->next = NULL;
    return rest;
}
```

30. Simulate Virtual Memory Paging (Simple Page Table Logic)

Solution:

```
#define PAGE_SIZE 4096
#define NUM_PAGES 16
void* page_table[NUM_PAGES];

void* translate_virtual_to_physical(uint32_t
virtual_addr) {
    int page_num = virtual_addr / PAGE_SIZE;
    int offset = virtual_addr % PAGE_SIZE;
    if (page_table[page_num])
        return (char*)page_table[page_num] + offset;
    else
        return NULL; // Page fault
}
```

31. Implement Atomic Fetch-And-Increment Using Inline Assembly (x86)

Solution:

```
int atomic_fetch_increment(int* ptr) {
    int old;
```

```
__asm__ __volatile__(
    "lock xaddl %0, %1"
    : "=r" (old), "+m" (*ptr)
    : "0" (1)
    : "memory"
);
return old;
}
```

32. Count Leading Zeros (CLZ) Without Using Compiler Intrinsic

Solution:

```
int clz(uint32_t x) {
    if (x == 0) return 32;
    int count = 0;
    for (int i = 31; i >= 0; i--) {
        if (x & (1U << i)) break;
        count++;
    }
    return count;
}
```

33. Implement a Circular Queue with Wrap-Around Without Modulo Operator

Solution:

```
#define SIZE 10
int queue[SIZE];
int front = 0, rear = 0;

int next_index(int index) {
    return (index + 1 == SIZE) ? 0 : index + 1;
}
```

34. Implement strstr() Without Using Library Functions

Solution:

```
char* my_strstr(const char* haystack, const char* needle) {
    if (!*needle) return (char*)haystack;
    for (; *haystack; haystack++) {
        const char* h = haystack;
        const char* n = needle;
        while (*h && *n && *h == *n) {
            h++; n++;
        }
        if (!*n) return (char*)haystack;
    }
    return NULL;
}
```

35. Implement a Lock-Free Stack Using Compare-And-Swap (CAS)

Solution:

```
#include <stdatomic.h>

typedef struct Node {
    int data;
    struct Node* next;
} Node;

_Atomic(Node*) top = NULL;

void push(int value) {
```

```
Node* new_node = malloc(sizeof(Node));
new_node->data = value;
Node* old_top;

do {
    old_top = atomic_load(&top);
    new_node->next = old_top;
} while (!atomic_compare_exchange_weak(&top,
&old_top, new_node));
}

int pop() {
    Node* old_top;
    do {
        old_top = atomic_load(&top);
        if (!old_top) return -1;
    } while (!atomic_compare_exchange_weak(&top,
&old_top, old_top->next));

    int value = old_top->data;
    free(old_top);
    return value;
}
```

36. Write a Byte-Aligned Bit Field Extractor

Solution:

```
unsigned int extract_bits(const unsigned char*
buffer, int start_bit, int bit_length) {
    unsigned int result = 0;
    for (int i = 0; i < bit_length; i++) {
        int byte_index = (start_bit + i) / 8;
```

```
        int bit_index = 7 - ((start_bit + i) % 8);
        result <<= 1;
        result |= (buffer[byte_index] >> bit_index) & 1;
    }
    return result;
}
```

37. Convert Float to IEEE 754 Binary Representation

Solution:

```
void print_float_bits(float f) {
    union {
        float f;
        unsigned int i;
    } u;
    u.f = f;

    for (int i = 31; i >= 0; i--) {
        printf("%d", (u.i >> i) & 1);
        if (i == 31 || i == 23) printf(" ");
    }
    printf("\n");
}
```

38. Implement a Minimal Custom Heap Allocator

Solution:

```
#define HEAP_SIZE 1024
char heap[HEAP_SIZE];
size_t offset = 0;

void* my_malloc(size_t size) {
```

```
    if (offset + size > HEAP_SIZE) return NULL;
    void* ptr = &heap[offset];
    offset += size;
    return ptr;
}
void my_free(void* ptr) {
    // No-op: Very basic, real free not supported
}
```

39. Implement Variable-Length Encoding (e.g., like UTF-8)

Solution:

```
int encode_varint(unsigned int value, unsigned char* out) {
    int i = 0;
    while (value >= 0x80) {
        out[i++] = (value & 0x7F) | 0x80;
        value >>= 7;
    }
    out[i++] = value;
    return i;
}
```

40. Simulate LRU Cache (Least Recently Used) Without Using Libraries

Solution:

```
#include <stdio.h>
#include <stdlib.h>
#define CACHE_SIZE 4
```

```
typedef struct Node {
    int key;
    struct Node* prev;
    struct Node* next;
} Node;

Node* head = NULL;
Node* tail = NULL;

void move_to_front(Node* node) {
    if (node == head) return;

    // Detach
    if (node->prev) node->prev->next = node->next;
    if (node->next) node->next->prev = node->prev;
    if (node == tail) tail = node->prev;

    // Move to front
    node->next = head;
    node->prev = NULL;
    if (head) head->prev = node;
    head = node;
}

void insert(int key) {
Node* temp = head;
while (temp) {
    if (temp->key == key) {
        move_to_front(temp);
        return;
    }
    temp = temp->next;
}
```

```
Node* new_node = malloc(sizeof(Node));
new_node->key = key;
new_node->prev = NULL;
new_node->next = head;
if (head) head->prev = new_node;
head = new_node;

// Enforce size
int count = 0;
temp = head;
while (temp) {
    count++;
    temp = temp->next;
}

if (count > CACHE_SIZE) {
    Node* del = tail;
    tail = tail->prev;
    tail->next = NULL;
    free(del);
}

if (!tail) {
    Node* p = head;
    while (p->next) p = p->next;
        tail = p;
    }
}
```

41. Write a Function That Executes Only Once (Thread-Safe)

Solution:

```
#include <stdatomic.h>

void run_once() {
    static atomic_flag flag = ATOMIC_FLAG_INIT;
    if (!atomic_flag_test_and_set(&flag)) {
        printf("Executed only once.\n");
    }
}
```

42. Print Binary Tree In-Order Without Recursion or Stack

Solution:

```
typedef struct Node {
    int data;
    struct Node *left, *right;
} Node;

void morris_traversal(Node* root) {
    Node* current = root;
    while (current) {
        if (!current->left) {
            printf("%d ", current->data);
            current = current->right;
        } else {
            Node* pre = current->left;
            while (pre->right && pre->right != current)
                pre = pre->right;
```

```
if (!pre->right) {
        pre->right = current;
        current = current->left;
} else {
        pre->right = NULL;
        printf("%d ", current->data);
        current = current->right;
                }
            }
        }
    }
```

43. Memory Copy with Overlap Handling (Like memmove)

Solution:

```
void* my_memmove(void* dest, const void* src,
size_t n) {
    unsigned char* d = dest;
    const unsigned char* s = src;
    if (d < s) {
        while (n--) *d++ = *s++;
    } else {
        d += n;
        s += n;
        while (n--) *(--d) = *(--s);
    }
    return dest;
}
```

44. Find the First Non-Repeated Character in a String (O(N))

Solution:

```
char first_non_repeated(const char* str) {
    int count[256] = {0};
    for (int i = 0; str[i]; i++)
        count[(unsigned char)str[i]]++;

    for (int i = 0; str[i]; i++)
        if (count[(unsigned char)str[i]] == 1)
            return str[i];
    return '\0';
}
```

45. Lock-Free Single Producer Single Consumer Queue

Solution:

```
#define QSIZE 64
int queue[QSIZE];
volatile int head = 0, tail = 0;

int enqueue(int val) {
    int next = (head + 1) % QSIZE;
    if (next == tail) return 0; // full
    queue[head] = val;
    head = next;
    return 1;
}
```

```
int dequeue(int* val) {
    if (head == tail) return 0; // empty
    *val = queue[tail];
    tail = (tail + 1) % QSIZE;
    return 1;
}
```

46. Count Set Bits Using Brian Kernighan's Algorithm

Solution:

```
int count_set_bits(unsigned int n) {
    int count = 0;
    while (n) {
        n &= (n - 1);
        count++;
    }
    return count;
}
```

47. Implement a Virtual Memory Page Table Entry Simulator

Solution:

```
typedef struct {
    unsigned int present : 1;
    unsigned int rw      : 1;
    unsigned int user    : 1;
    unsigned int frame   : 29;
} PTE;
void print_pte(PTE pte) {
    printf("Present: %u, RW: %u, User: %u, Frame: 0x%x\n",
        pte.present, pte.rw, pte.user, pte.frame);
}
```

48. Find Integer Square Root Without Using sqrt()

Solution:

```
unsigned int isqrt(unsigned int num) {
    unsigned int res = 0;
    unsigned int bit = 1 << 30;

    while (bit > num) bit >>= 2;

    while (bit != 0) {
        if (num >= res + bit) {
            num -= res + bit;
            res = (res >> 1) + bit;
        } else {
            res >>= 1;
        }
        bit >>= 2;
    }
    return res;
}
```

49. Reorder Array to Move All Zeros to the End

Solution:

```
void move_zeros(int* arr, int n) {
    int idx = 0;
    for (int i = 0; i < n; i++)
        if (arr[i] != 0)
            arr[idx++] = arr[i];
    while (idx < n)
        arr[idx++] = 0;
}
```

50. Implement Memory Leak Tracker

Solution:

```
typedef struct {
    void* ptr;
    size_t size;
} MemEntry;

#define MAX_ENTRIES 1000
MemEntry mem_table[MAX_ENTRIES];
int mem_index = 0;

void* my_malloc(size_t size) {
    void* p = malloc(size);
    mem_table[mem_index++] = (MemEntry){p, size};
    return p;
}

void my_free(void* p) {
    for (int i = 0; i < mem_index; ++i) {
        if (mem_table[i].ptr == p) {
            mem_table[i] = mem_table[--mem_index];
            break;
        }
    }
    free(p);
}

void report_leaks() {
    for (int i = 0; i < mem_index; ++i) {
        printf("Leak at %p (%zu bytes)\n",
mem_table[i].ptr, mem_table[i].size);
    }
}
```

PROV LOGIC

Excellence in World class VLSI Training & Placements

Do follow for updates & enquires

+91- 9182280927