COLLECTION OF C CODE EXAMPLES FOR DATA STRUCTURES AND ALGORITHMS

1️⃣ STATIC STACK USING ARRAY WITH PUSH() & DISPLAY()

```
#include <stdio.h>
#define MAX 100

int stack[MAX];  // array to store stack elements
int top = -1;    // top of stack

// push an element onto the stack
void push(int x) {
    if (top == MAX - 1) {      // check overflow
        printf("Stack overflow\n");
        return;
    }
    stack[++top] = x;          // insert element
}

// display all elements of stack
void display() {
    if (top == -1) {           // check underflow
        printf("Stack is empty\n");
        return;
    }
    printf("Stack (top->bottom): ");
    for (int i = top; i >= 0; i--) printf("%d ", stack[i]);
    printf("\n");
}

int main() {
    push(10); push(20); push(30);  // insert elements
    display();                     // show stack
    return 0;
}
```

2️⃣ BUBBLE SORT

```
#include <stdio.h>

// bubble sort function
void bubbleSort(int a[], int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - 1 - i; j++)
            if (a[j] > a[j + 1]) {   // swap if in wrong order
                int t = a[j];
                a[j] = a[j + 1];
                a[j + 1] = t;
            }
}

int main() {
    int a[] = {64, 25, 12, 22, 11};
    int n = sizeof(a) / sizeof(a[0]);
    bubbleSort(a, n);   // call sort function
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

3️⃣ POSTFIX EVALUATION USING ARRAY STACK

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define MAX 100

long long stack[MAX];
int top;

// initialize stack
void init() { top = -1; }

// push element
void push(long long v) { stack[++top] = v; }
```

```
// pop element
long long pop() { return stack[top--]; }

// evaluate postfix expression
long long evalPostfix(const char *expr) {
    init();
    char *token = strtok(strdup(expr), " ");
    while (token) {
        if (isdigit(token[0])) push(atoll(token));     // if number, push
        else {                                          // if operator, po
            long long b = pop(), a = pop();
            switch (token[0]) {
                case '+': push(a + b); break;
                case '-': push(a - b); break;
                case '*': push(a * b); break;
                case '/': push(a / b); break;
            }
        }
        token = strtok(NULL, " ");
    }
    return pop();   // final result
}

int main() {
    const char *expr = "5 1 2 + 4 * + 3 -";
    printf("Result = %lld\n", evalPostfix(expr));
    return 0;
}
```

4️⃣ STATIC LINKED LIST WITH CREATE() & INSERT()

```
#include <stdio.h>
#include <stdlib.h>

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

struct Node *head = NULL;
```

```
// create linked list from array
void create(int arr[], int n) {
    struct Node *tail = NULL;
    for (int i = 0; i < n; i++) {
        struct Node *p = malloc(sizeof(*p));
        p->data = arr[i];
        p->next = NULL;
        if (!head) head = tail = p;
        else { tail->next = p; tail = p; }
    }
}

// insert new node at position
void insert(int data, int pos) {
    struct Node *p = malloc(sizeof(*p));
    p->data = data;
    if (pos == 0) { p->next = head; head = p; return; }
    struct Node *cur = head;
    for (int i = 0; cur && i < pos - 1; i++) cur = cur->next;
    p->next = cur->next;
    cur->next = p;
}

// display list
void display() {
    struct Node *p = head;
    while (p) { printf("%d -> ", p->data); p = p->next; }
    printf("NULL\n");
}

int main() {
    int arr[] = {10, 20, 30};
    create(arr, 3);
    display();
    insert(25, 2);
    display();
    return 0;
}
```

5️⃣ INSERTION SORT

```
#include <stdio.h>

// insertion sort function
void insertionSort(int a[], int n) {
    for (int i = 1; i < n; i++) {
        int key = a[i];
        int j = i - 1;
        while (j >= 0 && a[j] > key) {   // shift larger elements right
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = key;  // insert key at right place
    }
}

int main() {
    int a[] = {12, 11, 13, 5, 6};
    int n = sizeof(a) / sizeof(a[0]);
    insertionSort(a, n);
    printf("Sorted: ");
    for (int i = 0; i < n; i++) printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

6️⃣ STATIC QUEUE WITH ISFULL() & INSERT()

```
#include <stdio.h>
#define MAX 5

int q[MAX];
int front = 0, rear = -1, size = 0;

// check full
int isFull() { return size == MAX; }

// insert element
void insert(int x) {
    if (isFull()) { printf("Queue full\n"); return; }
```

```
    rear = (rear + 1) % MAX;
    q[rear] = x;
    size++;
}

// display queue
void display() {
    if (size == 0) { printf("Queue empty\n"); return; }
    for (int i = 0; i < size; i++) printf("%d ", q[(front + i) % MAX]);
    printf("\n");
}

int main() {
    insert(10); insert(20); insert(30);
    display();
    return 0;
}
```

7️⃣ STATIC STACK WITH ISEMPTY() & POP()

```
#include <stdio.h>
#define MAX 50

int stack[MAX];
int top = -1;

// check empty
int isEmpty() { return top == -1; }

// push
void push(int x) { stack[++top] = x; }

// pop
int pop() {
    if (isEmpty()) { printf("Empty\n"); return -1; }
    return stack[top--];
}

int main() {
    push(5); push(15);
    printf("Popped: %d\n", pop());
```

```
    printf("Popped: %d\n", pop());
    printf("Popped: %d\n", pop());
    return 0;
}
```

8️⃣ QUICK SORT

```
#include <stdio.h>

// swap two numbers
void swap(int *a, int *b) { int t = *a; *a = *b; *b = t; }

// partition array
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++)
        if (arr[j] < pivot)
            swap(&arr[++i], &arr[j]);
    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}

// quick sort function
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    for (int i = 0; i < n; i++) printf("%d ", arr[i]);
    printf("\n");
    return 0;
}
```

9️⃣ SINGLY LINKED LIST WITH CREATE() & DISPLAY()

```
#include <stdio.h>
#include <stdlib.h>

struct Node { int data; struct Node *next; };
struct Node *head = NULL;

// create linked list
void create(int arr[], int n) {
    struct Node *tail = NULL;
    for (int i = 0; i < n; i++) {
        struct Node *p = malloc(sizeof(*p));
        p->data = arr[i]; p->next = NULL;
        if (!head) head = tail = p;
        else { tail->next = p; tail = p; }
    }
}

// display list
void display() {
    struct Node *p = head;
    while (p) { printf("%d -> ", p->data); p = p->next; }
    printf("NULL\n");
}

int main() {
    int a[] = {1, 2, 3, 4};
    create(a, 4);
    display();
    return 0;
}
```

🔟 DYNAMIC QUEUE WITH INSERT() & DISPLAY()

```
#include <stdio.h>
#include <stdlib.h>

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

```
struct Node *front = NULL, *rear = NULL;

// insert into queue
void insert(int x) {
    struct Node *p = malloc(sizeof(*p));
    p->data = x; p->next = NULL;
    if (!rear) front = rear = p;
    else { rear->next = p; rear = p; }
}

// display queue
void display() {
    struct Node *p = front;
    if (!p) { printf("Queue empty\n"); return; }
    while (p) { printf("%d ", p->data); p = p->next; }
    printf("\n");
}

int main() {
    insert(10); insert(20); insert(30);
    display();
    return 0;
}
```

1️⃣1️⃣ LINEAR SEARCH

```
#include <stdio.h>

// simple linear search
int linearSearch(int a[], int n, int key) {
    for (int i = 0; i < n; i++)
        if (a[i] == key) return i;
    return -1;
}

int main() {
    int a[] = {2, 4, 0, 1, 9};
    int key = 1;
    int pos = linearSearch(a, 5, key);
    if (pos != -1) printf("Found at index %d\n", pos);
    else printf("Not found\n");
```

```
    return 0;
}
```

1️⃣2️⃣ DYNAMIC STACK USING LINKED LIST WITH PUSH() & DISPLAY()

```
#include <stdio.h>
#include <stdlib.h>

struct Node { int data; struct Node *next; };
struct Node *top = NULL;

// push onto stack
void push(int x) {
    struct Node *p = malloc(sizeof(*p));
    p->data = x; p->next = top; top = p;
}

// pop element
int pop() {
    if (!top) { printf("Empty\n"); return -1; }
    int v = top->data;
    struct Node *t = top; top = top->next; free(t);
    return v;
}

// display stack
void display() {
    struct Node *p = top;
    while (p) { printf("%d ", p->data); p = p->next; }
    printf("\n");
}

int main() {
    push(5); push(10); push(15);
    display();
    printf("Popped: %d\n", pop());
    display();
    return 0;
}
```

1️⃣3️⃣ STATIC QUEUE WITH INIT() & ISEMPTY()

```
#include <stdio.h>
#define MAX 5

int q[MAX];
int front, rear, size;

// initialize
void init() { front = 0; rear = -1; size = 0; }

// check empty
int isEmpty() { return size == 0; }

// insert element
void insert(int x) {
    if (size == MAX) { printf("Full\n"); return; }
    rear = (rear + 1) % MAX;
    q[rear] = x;
    size++;
}

// display queue
void display() {
    if (isEmpty()) { printf("Empty\n"); return; }
    for (int i = 0; i < size; i++) printf("%d ", q[(front + i) % MAX]);
    printf("\n");
}

int main() {
    init();
    insert(10); insert(20);
    display();
    printf("Empty? %s\n", isEmpty() ? "Yes" : "No");
    return 0;
}
```

1️⃣4️⃣ DOUBLY LINKED LIST WITH CREATE() & DISPLAY()

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node *prev, *next;
};
struct Node *head = NULL;

// create list
void create(int a[], int n) {
    struct Node *tail = NULL;
    for (int i = 0; i < n; i++) {
        struct Node *p = malloc(sizeof(*p));
        p->data = a[i];
        p->prev = p->next = NULL;
        if (!head) head = tail = p;
        else { tail->next = p; p->prev = tail; tail = p; }
    }
}

// display list forward
void display() {
    struct Node *p = head;
    while (p) { printf("%d <-> ", p->data); p = p->next; }
    printf("NULL\n");
}

int main() {
    int a[] = {1, 2, 3, 4};
    create(a, 4);
    display();
    return 0;
}
```

1️⃣5️⃣ STATIC QUEUE WITH ISEMPTY() & INSERT()

```
#include <stdio.h>
#define MAX 4
```

```
int q[MAX], front = 0, rear = -1, size = 0;

// check empty
int isEmpty() { return size == 0; }

// insert element
void insert(int x) {
    if (size == MAX) { printf("Full\n"); return; }
    rear = (rear + 1) % MAX; q[rear] = x; size++;
}

// display
void display() {
    if (isEmpty()) { printf("Empty\n"); return; }
    for (int i = 0; i < size; i++) printf("%d ", q[(front + i) % MAX]);
    printf("\n");
}

int main() {
    insert(7); insert(8);
    display();
    printf("Empty? %s\n", isEmpty() ? "Yes" : "No");
    return 0;
}
```

1️⃣6️⃣ STATIC STACK WITH INIT() & INSERT()

```
#include <stdio.h>
#define MAX 50

int stack[MAX], top;

// initialize
void init() { top = -1; }

// insert = push
void insert(int x) {
    if (top == MAX - 1) { printf("Overflow\n"); return; }
    stack[++top] = x;
}
```

```
// pop element
int pop() {
    if (top == -1) { printf("Empty\n"); return -1; }
    return stack[top--];
}

int main() {
    init();
    insert(1); insert(2);
    printf("Popped: %d\n", pop());
    return 0;
}
```

1️⃣7️⃣ DYNAMIC QUEUE WITH DELETE() & DISPLAY()

```
#include <stdio.h>
#include <stdlib.h>

struct Node { int data; struct Node *next; };
struct Node *front = NULL, *rear = NULL;

// insert
void insert(int x) {
    struct Node *p = malloc(sizeof(*p));
    p->data = x; p->next = NULL;
    if (!rear) front = rear = p;
    else { rear->next = p; rear = p; }
}

// delete front
int deleteQ() {
    if (!front) { printf("Empty\n"); return -1; }
    int v = front->data;
    struct Node *t = front; front = front->next;
    if (!front) rear = NULL;
    free(t);
    return v;
}

// display
```

```
void display() {
    struct Node *p = front;
    if (!p) { printf("Empty\n"); return; }
    while (p) { printf("%d ", p->data); p = p->next; }
    printf("\n");
}

int main() {
    insert(10); insert(20); insert(30);
    display();
    printf("Deleted: %d\n", deleteQ());
    display();
    return 0;
}
```

1️⃣8️⃣ POSTFIX EVALUATION USING LINKED LIST STACK

```
#include <stdio.h>
#include <ctype.h>   // for isdigit()

#define SIZE 50
int stack[SIZE];
int top;

// Function to initialize the stack
void init() {
    top = -1;  // stack is empty initially
}

// Function to push an element into the stack
void push(int val) {
    if (top == SIZE - 1)
        printf("Stack Overflow\n");
    else
        stack[++top] = val;
}

// Function to pop an element from the stack
int pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
```

```
        return -1;
    } else
        return stack[top--];
}

int main() {
    char exp[SIZE];
    char *e;
    int num1, num2, res;

    printf("Enter a postfix expression: ");
    scanf("%s", exp);   // example: 23*54*+9-

    init();             // initialize stack
    e = exp;

    while (*e != '\0') {
        if (isdigit(*e)) {
            // if character is a number, push it
            push(*e - '0');
        } else {
            // if operator, pop two operands and apply operator
            num1 = pop();
            num2 = pop();
            switch (*e) {
                case '+': res = num2 + num1; break;
                case '-': res = num2 - num1; break;
                case '*': res = num2 * num1; break;
                case '/': res = num2 / num1; break;
            }
            // push result back to stack
            push(res);
        }
        e++;
    }

    // final result will be on top of stack
    printf("Result of Postfix Expression = %d\n", pop());
    return 0;
}
```