1. Array Reversal with Pointers
Write a function `void reverseArray(int *arr, int len)` that reverses the elements of an array using only pointer arithmetic (no subscript notation).

---

2. Summation via Pointers
Create a function `int sumElements(int *arr, int len)` that computes the sum of elements in an array using pointer arithmetic. Avoid using array subscripts.

---

3. First Occurrence Pointer
Implement `int* findFirstOccurrence(int *arr, int len, int target)` to return a pointer to the first occurrence of `target` in the array. Return `NULL` if not found.

---

4. Matrix Column Sum
Write a function `void columnSum(int (*matrix)[N], int rows, double *result)` that calculates the sum of each column in a 2D matrix using pointers. `result` stores the sums.

---

5. String Length via Pointers
Write `int pointerStrlen(char *str)` to compute the length of a string using pointer arithmetic (do not use `strlen`).

---

6. Dynamic Array Statistics
Dynamically allocate an array of `n` integers. Use pointers to compute the mean and variance. Return results via pointers in `void stats(int *arr, int n, double *mean, double *variance)`.

---

7. Swap Arrays via Pointers
Implement `void swapArrays(int *a, int *b, int len)` to swap elements between two arrays using pointers. Assume arrays are of equal length.

---

8. Circular Shift Left
Write `void shiftLeft(int *arr, int len, int k)` to perform a circular left shift by `k` positions using pointers. Avoid array subscripts.

---

9. Pointer-Based Binary Search
Create `int* binarySearch(int *arr, int len, int key)` that returns a pointer to the found element using binary search. Use pointer arithmetic only.

---

10. Merge Sorted Arrays
Write `void mergeSorted(int *a, int aLen, int *b, int bLen, int *result)` to merge two sorted arrays into `result` using pointers.

---

11. Remove Duplicates in Place
Implement `int removeDuplicates(int *arr, int len)` to remove duplicates from a sorted array using pointers. Return the new length.

---

12. Two-Pointer Two Sum
Solve the two-sum problem with `int* twoSum(int *arr, int len, int target)` using two pointers. Return indices via a dynamically allocated array.

---

13. Rotate Array via Pointers
Write `void rotateArray(int *arr, int len, int k)` to rotate the array right by `k` positions using pointer manipulation.

---

14. Check Sorted Array
Create `int isSorted(int *arr, int len)` to check if an array is sorted in ascending order using pointers. Return 1 if sorted, 0 otherwise.

---

15. String Concatenation with Pointers
Implement `void pointerStrcat(char *dest, char *src)` to concatenate `src` to `dest` using only pointers. Handle overflow.

---

16. Matrix Transpose via Pointers
Write `void transposeMatrix(int (*matrix)[N], int rows)` to transpose a square matrix using pointer arithmetic.

---

17. Pointer-Based Substring Search
Create `char* findSubstring(char *str, char *substr)` to find the first occurrence of `substr` in `str` using pointers. Return the address or `NULL`.

---

18. Palindrome Check for Integers
Write `int isIntPalindrome(int *arr, int len)` to check if an integer array is a palindrome using two pointers (start and end).

---

19. Function Pointers for Operations
Define a function `void applyOperation(int *arr, int len, int (*op)(int))` that applies a function (e.g., square, increment) to each element using function pointers.

---

20. Largest Subarray Sum via Pointers
Implement `int maxSubarraySum(int *arr, int len)` to find the maximum sum of any contiguous subarray using a two-pointer approach.

1)

```
#include <stdio.h>

void reverseArray(int *arr, int len) {
    int *start = arr;
    int *end = arr + len - 1;
    while (start < end) {
```

```
        int temp = *start;
        *start = *end;
        *end = temp;
        start++;
        end--;
    }
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int len = sizeof(arr) / sizeof(arr[0]);
    reverseArray(arr, len);
    for (int i = 0; i < len; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}

2)
#include <stdio.h>

int sumElements(int *arr, int len) {
    int sum = 0;
    int *ptr = arr;
    for (int i = 0; i < len; i++) {
        sum += *ptr;
        ptr++;
    }
    return sum;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int len = sizeof(arr) / sizeof(arr[0]);
    printf("Sum: %d\n", sumElements(arr, len));
    return 0;
}

3)
#include <stdio.h>

int* findFirstOccurrence(int *arr, int len, int target) {
    int *ptr = arr;
    for (int i = 0; i < len; i++) {
```

```
        if (*ptr == target) {
            return ptr;
        }
        ptr++;
    }
    return NULL;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int len = sizeof(arr) / sizeof(arr[0]);
    int target = 3;
    int *result = findFirstOccurrence(arr, len, target);
    if (result) {
        printf("Found at address: %p\n", result);
    } else {
        printf("Not found\n");
    }
    return 0;
}
```

4)

```
#include <stdio.h>
#define N 3

void columnSum(int (*matrix)[N], int rows, double *result) {
    for (int col = 0; col < N; col++) {
        result[col] = 0;
        for (int row = 0; row < rows; row++) {
            result[col] += *(*(matrix + row) + col);
        }
    }
}

int main() {
    int matrix[][N] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    double result[N];
    columnSum(matrix, 3, result);
    for (int i = 0; i < N; i++) {
        printf("Column %d sum: %.2f\n", i, result[i]);
    }
    return 0;
}
```

5)

```
#include <stdio.h>

int pointerStrlen(char *str) {
    int len = 0;
    while (*str != '\0') {
        len++;
        str++;
    }
    return len;
}

int main() {
    char str[] = "Hello, World!";
    printf("Length: %d\n", pointerStrlen(str));
    return 0;
}
```

6)

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

void stats(int *arr, int n, double *mean, double *variance) {
    double sum = 0;
    for (int i = 0; i < n; i++) {
        sum += arr[i];
    }
    *mean = sum / n;

    double varSum = 0;
    for (int i = 0; i < n; i++) {
        varSum += (arr[i] - *mean) * (arr[i] - *mean);
    }
    *variance = varSum / n;
}

int main() {
    int n = 5;
    int *arr = (int *)malloc(n * sizeof(int));
    arr[0] = 1; arr[1] = 2; arr[2] = 3; arr[3] = 4; arr[4] = 5;
    double mean, variance;
    stats(arr, n, &mean, &variance);
    printf("Mean: %.2f, Variance: %.2f\n", mean, variance);
    free(arr);
```

```
    return 0;
}
```

7)

```
#include <stdio.h>

void swapArrays(int *a, int *b, int len) {
    for (int i = 0; i < len; i++) {
        int temp = *(a + i);
        *(a + i) = *(b + i);
        *(b + i) = temp;
    }
}

int main() {
    int a[] = {1, 2, 3};
    int b[] = {4, 5, 6};
    int len = sizeof(a) / sizeof(a[0]);
    swapArrays(a, b, len);
    printf("Array A: ");
    for (int i = 0; i < len; i++) printf("%d ", a[i]);
    printf("\nArray B: ");
    for (int i = 0; i < len; i++) printf("%d ", b[i]);
    return 0;
}
```

8)

```
#include <stdio.h>

void shiftLeft(int *arr, int len, int k) {
    k = k % len;
    for (int i = 0; i < k; i++) {
        int temp = *arr;
        for (int j = 0; j < len - 1; j++) {
            *(arr + j) = *(arr + j + 1);
        }
        *(arr + len - 1) = temp;
    }
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int len = sizeof(arr) / sizeof(arr[0]);
    shiftLeft(arr, len, 2);
```

```
    for (int i = 0; i < len; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

9)

```
#include <stdio.h>

int* binarySearch(int *arr, int len, int key) {
    int *low = arr;
    int *high = arr + len - 1;
    while (low <= high) {
        int *mid = low + (high - low) / 2;
        if (*mid == key) return mid;
        else if (*mid < key) low = mid + 1;
        else high = mid - 1;
    }
    return NULL;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int len = sizeof(arr) / sizeof(arr[0]);
    int key = 3;
    int *result = binarySearch(arr, len, key);
    if (result) {
        printf("Found at address: %p\n", result);
    } else {
        printf("Not found\n");
    }
    return 0;
}
```

10)

```
#include <stdio.h>

void mergeSorted(int *a, int aLen, int *b, int bLen, int *result) {
    int *ptrA = a, *ptrB = b, *ptrResult = result;
    while (ptrA < a + aLen && ptrB < b + bLen) {
        if (*ptrA < *ptrB) {
            *ptrResult++ = *ptrA++;
        } else {
            *ptrResult++ = *ptrB++;
```

```
        }
    }
    while (ptrA < a + aLen) *ptrResult++ = *ptrA++;
    while (ptrB < b + bLen) *ptrResult++ = *ptrB++;
}

int main() {
    int a[] = {1, 3, 5};
    int b[] = {2, 4, 6};
    int result[6];
    mergeSorted(a, 3, b, 3, result);
    for (int i = 0; i < 6; i++) {
        printf("%d ", result[i]);
    }
    return 0;
}
```

11)

```
#include <stdio.h>

int removeDuplicates(int *arr, int len) {
    if (len == 0) return 0;
    int *ptr = arr;
    int uniqueIndex = 0;
    for (int i = 1; i < len; i++) {
        if (*(arr + i) != *(arr + uniqueIndex)) {
            uniqueIndex++;
            *(arr + uniqueIndex) = *(arr + i);
        }
    }
    return uniqueIndex + 1;
}

int main() {
    int arr[] = {1, 1, 2, 2, 3, 4, 4, 5};
    int len = sizeof(arr) / sizeof(arr[0]);
    int newLen = removeDuplicates(arr, len);
    for (int i = 0; i < newLen; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

12)

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

int* twoSum(int *arr, int len, int target) {
    int *result = (int *)malloc(2 * sizeof(int));
    int *left = arr;
    int *right = arr + len - 1;
    while (left < right) {
        int sum = *left + *right;
        if (sum == target) {
            result[0] = left - arr;
            result[1] = right - arr;
            return result;
        } else if (sum < target) {
            left++;
        } else {
            right--;
        }
    }
    return NULL;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int len = sizeof(arr) / sizeof(arr[0]);
    int target = 7;
    int *result = twoSum(arr, len, target);
    if (result) {
        printf("Indices: %d, %d\n", result[0], result[1]);
        free(result);
    } else {
        printf("No solution\n");
    }
    return 0;
}
```

13)

```
#include <stdio.h>

void rotateArray(int *arr, int len, int k) {
    k = k % len;
    int temp[k];
    for (int i = 0; i < k; i++) {
```

```
        temp[i] = *(arr + len - k + i);
    }
    for (int i = len - 1; i >= k; i--) {
        *(arr + i) = *(arr + i - k);
    }
    for (int i = 0; i < k; i++) {
        *(arr + i) = temp[i];
    }
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int len = sizeof(arr) / sizeof(arr[0]);
    rotateArray(arr, len, 2);
    for (int i = 0; i < len; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

14)

```
#include <stdio.h>

int isSorted(int *arr, int len) {
    for (int i = 1; i < len; i++) {
        if (*(arr + i) < *(arr + i - 1)) {
            return 0;
        }
    }
    return 1;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int len = sizeof(arr) / sizeof(arr[0]);
    printf("Is sorted: %d\n", isSorted(arr, len));
    return 0;
}
```

15)

```
#include <stdio.h>

void pointerStrcat(char *dest, char *src) {
    while (*dest) dest++;
```

```
    while (*src) {
        *dest = *src;
        dest++;
        src++;
    }
    *dest = '\0';
}

int main() {
    char dest[100] = "Hello, ";
    char src[] = "World!";
    pointerStrcat(dest, src);
    printf("Result: %s\n", dest);
    return 0;
}
```

16)

```
#include <stdio.h>
#define N 3

void transposeMatrix(int (*matrix)[N], int rows) {
    for (int i = 0; i < rows; i++) {
        for (int j = i + 1; j < N; j++) {
            int temp = *(*(matrix + i) + j);
            *(*(matrix + i) + j) = *(*(matrix + j) + i);
            *(*(matrix + j) + i) = temp;
        }
    }
}

int main() {
    int matrix[N][N] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    transposeMatrix(matrix, N);
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

17)

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

```
char* findSubstring(char *str, char *substr) {
    while (*str) {
        char *start = str;
        char *pattern = substr;
        while (*str && *pattern && *str == *pattern) {
            str++;
            pattern++;
        }
        if (!*pattern) {
            return start; // Substring found
        }
        str = start + 1; // Move to the next character in the main string
    }
    return NULL; // Substring not found
}

int main() {
    char str[] = "Hello, World!";
    char substr[] = "World";
    char *result = findSubstring(str, substr);
    if (result) {
        printf("Substring found at address: %p\n", result);
    } else {
        printf("Substring not found\n");
    }
    return 0;
}
```

18)

```
#include <stdio.h>

int isIntPalindrome(int *arr, int len) {
    int *start = arr;
    int *end = arr + len - 1;
    while (start < end) {
        if (*start != *end) {
            return 0; // Not a palindrome
        }
        start++;
        end--;
    }
    return 1; // Palindrome
}
```

```
int main() {
    int arr[] = {1, 2, 3, 2, 1};
    int len = sizeof(arr) / sizeof(arr[0]);
    if (isIntPalindrome(arr, len)) {
        printf("The array is a palindrome\n");
    } else {
        printf("The array is NOT a palindrome\n");
    }
    return 0;
}
```

19)

```
#include <stdio.h>

// Function to square a number
int square(int x) {
    return x * x;
}

// Function to increment a number
int increment(int x) {
    return x + 1;
}

// Function to apply an operation to each element of the array
void applyOperation(int *arr, int len, int (*op)(int)) {
    for (int i = 0; i < len; i++) {
        *(arr + i) = op(*(arr + i));
    }
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int len = sizeof(arr) / sizeof(arr[0]);

    // Apply square operation
    applyOperation(arr, len, square);
    printf("After squaring: ");
    for (int i = 0; i < len; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
```

```
    // Apply increment operation
    applyOperation(arr, len, increment);
    printf("After incrementing: ");
    for (int i = 0; i < len; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

20)

```
#include <stdio.h>

int maxSubarraySum(int *arr, int len) {
    int maxSum = *arr; // Initialize maxSum with the first element
    int currentSum = *arr; // Initialize currentSum with the first element
    int *ptr = arr + 1; // Start from the second element

    for (int i = 1; i < len; i++) {
        // Decide whether to start a new subarray or continue the current one
        currentSum = (*ptr > currentSum + *ptr) ? *ptr : currentSum + *ptr;
        // Update maxSum if currentSum is greater
        maxSum = (currentSum > maxSum) ? currentSum : maxSum;
        ptr++; // Move to the next element
    }

    return maxSum;
}

int main() {
    int arr[] = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
    int len = sizeof(arr) / sizeof(arr[0]);
    printf("Maximum subarray sum: %d\n", maxSubarraySum(arr, len));
    return 0;
}
```