24/02/24

The matrix values and then find the saddle points

```
#include <stdio.h>

#define ROW 3
#define COL 3

// Function to find saddle points
void findSaddlePoints(int matrix[ROW][COL]) {
    int i, j, k, min_in_row, max_in_col;
    for (i = 0; i < ROW; i++) {
        min_in_row = matrix[i][0];
        for (j = 1; j < COL; j++) {
            if (matrix[i][j] < min_in_row) {
                min_in_row = matrix[i][j];
            }
        }
        for (k = 0; k < COL; k++) {
            max_in_col = matrix[0][k];
            for (j = 1; j < ROW; j++) {
                if (matrix[j][k] > max_in_col) {
                    max_in_col = matrix[j][k];
                }
            }
            if (min_in_row == max_in_col) {
                printf("Saddle Point found at (%d, %d): %d\n", i, k, min_in_row);
            }
        }
    }
}

int main() {
    int matrix[ROW][COL];
    printf("Enter matrix elements (%d x %d):\n", ROW, COL);
    for (int i = 0; i < ROW; i++) {
        for (int j = 0; j < COL; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }
    printf("Given matrix:\n");
    for (int i = 0; i < ROW; i++) {
        for (int j = 0; j < COL; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
    printf("Saddle points:\n");
    findSaddlePoints(matrix);
    return 0;
}
```

OUTPUT:

```
Enter matrix elements (3 x 3):
1 2 3
4 5 6
7 8 9
Given matrix:
1 2 3
4 5 6
7 8 9
Saddle points:
Saddle Point found at (2, 0): 7
```

Write a program to swap the values of two variables using pointers.

```
#include <stdio.h>
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
int main() {
    int x = 10, y = 20;
    printf("Before swapping: x = %d, y = %d\n", x, y);
    swap(&x, &y);
    printf("After swapping: x = %d, y = %d\n", x, y);
    return 0;
}
```

OUTPUT:

```
Before swapping: x = 10, y = 20
After swapping: x = 20, y = 10
```

Write a program to find the length of a string using pointers.

```
#include <stdio.h>
int stringLength(char *str) {
    char *ptr = str;
    int length = 0;
    while (*ptr != '\0') {
        length++;
        ptr++;
    }
    return length;
}
int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);
    int length = stringLength(str);
    printf("Length of the string: %d\n", length);
    return 0;
}
```

OUTPUT:

```
Enter a string: YERRISWAMY
Length of the string: 10
```

Write a program to reverse a string using pointers.

```
#include <stdio.h>
void reverseString(char *str) {
    char *start = str;
    char *end = str;
    while (*end != '\0') {
        end++;
    }
    end--;
    while (start < end) {
        char temp = *start;
        *start = *end;
        *end = temp;
        start++;
        end--;
    }
}
int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);
    reverseString(str);
    printf("Reversed string: %s\n", str);
    return 0;
}
```

OUTPUT:

```
Enter a string: 123456
Reversed string: 654321
```

Count the number of occurrences of a character in a string using pointers

```
#include <stdio.h>
int countOccurrences(char *str, char ch) {
    int count = 0;
    while (*str != '\0') {
        if (*str == ch) {
            count++;
        }
        str++;
    }
    return count;
}
int main() {
    char str[100];
    char ch;
    printf("Enter a string: ");
    scanf("%s", str);
    printf("Enter the character to search for: ");
    scanf(" %c", &ch);
    int occurrences = countOccurrences(str, ch);
    printf("Number of occurrences of '%c' in the string: %d\n", ch, occurrences);
    return 0;
}
```

OUTPUT:

```
Enter a string: VEERANJINEYULU
Enter the character to search for: E
Number of occurrences of 'E' in the string: 3
```

Find the maximum and minimum values in an array using pointers.

```
#include <stdio.h>
int findMax(int *arr, int size) {
    int max = *arr;
    for (int i = 1; i < size; i++) {
        if (*(arr + i) > max) {
            max = *(arr + i);
        }
    }
    return max;
}
int findMin(int *arr, int size) {
    int min = *arr;
    for (int i = 1; i < size; i++) {
        if (*(arr + i) < min) {
            min = *(arr + i);
        }
    }
    return min;
}
int main() {
    int arr[100];
    int size;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &size);
    printf("Enter the elements of the array: ");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }
    int max = findMax(arr, size);
    int min = findMin(arr, size);
    printf("Maximum value in the array: %d\n", max);
    printf("Minimum value in the array: %d\n", min);
    return 0;
}
```

OUTPUT:

```
Enter the number of elements in the array: 4
Enter the elements of the array: 5
7
9
25
Maximum value in the array: 25
Minimum value in the array: 5
```

To sort an array of integers in ascending order using pointers

```
#include <stdio.h>
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
void bubbleSort(int *arr, int size) {
    int i, j;
    for (i = 0; i < size - 1; i++) {
        for (j = 0; j < size - i - 1; j++) {
            if (*(arr + j) > *(arr + j + 1)) {
                swap(arr + j, arr + j + 1);
            }
        }
    }
}
int main() {
    int arr[100];
    int size;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &size);
    printf("Enter the elements of the array: ");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }
    bubbleSort(arr, size);
    printf("Sorted array in ascending order: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

OUTPUT:

```
Enter the number of elements in the array: 5
Enter the elements of the array: 24
876
94847
576
46
Sorted array in ascending order: 24 46 576 876 94847
```

To find the factorial of a number using pointers.

```
#include <stdio.h>
void factorial(int num, int *result) {
    *result = 1;
    for (int i = 1; i <= num; ++i) {
        *result *= i;
    }
}
int main() {
    int number;
    printf("Enter a number: ");
    scanf("%d", &number);
    int result;
    factorial(number, &result);
    printf("Factorial of %d is: %d\n", number, result);
    return 0;
}
```

OUTPUT:

```
Enter a number: 6
Factorial of 6 is: 720
```

To check if a given number is prime using pointers.

```
#include <stdio.h>

#include <stdbool.h>

bool is_prime(int *num) {

    if (*num <= 1) {

        return false;

    }

    for (int i = 2; i * i <= *num; ++i) {
        if (*num % i == 0) {
            return false;
        }
    }
    return true;
}
int main() {
    int number;
    printf("Enter a number: ");
    scanf("%d", &number);
    int *ptr = &number;
    bool prime = is_prime(ptr);
    if (prime) {
        printf("%d is a prime number.\n", number);
    } else {
        printf("%d is not a prime number.\n", number);
    }
    return 0;
}
```

OUTPUT:

```
Enter a number: 225
225 is not a prime number.
```

To concatenate two strings using pointers.

```
#include <stdio.h>
void concat_strings(char *dest, const char *src) {
    while (*dest != '\0') {
        dest++;
    }
    while (*src != '\0') {
        *dest = *src;
        dest++;
        src++;
    }
    *dest = '\0';
}
int main() {
    char str1[100], str2[100];
    printf("Enter the first string: ");
    scanf("%s", str1);
    printf("Enter the second string: ");
    scanf("%s", str2);
    concat_strings(str1, str2);
    printf("Concatenated string: %s\n", str1);
    return 0;
}
```

OUTPUT:

```
Enter the first string: 1,2,3,4,5,

Enter the second string: 6,7,8,9,10

Concatenated string: 1,2,3,4,5,6,7,8,9,10
```

To find the sum of elements in an array using pointers.

```
#include <stdio.h>

int array_sum(int *arr, int size) {

    int sum = 0;

    for (int i = 0; i < size; ++i) {

        sum += *(arr + i);

    }

    return sum;

}

int main() {

    int size;

    printf("Enter the size of the array: ");

    scanf("%d", &size);

    int arr[size];

    printf("Enter the elements of the array:\n");

    for (int i = 0; i < size; ++i) {

        scanf("%d", &arr[i]);

    }

    int *ptr = arr;

    int sum = array_sum(ptr, size);
    printf("Sum of the elements: %d\n", sum);
    return 0;
}
```

OUTPUT:

```
Enter the size of the array: 5
Enter the elements of the array:
25
46
78
90
84
Sum of the elements: 323
```

To delete an element from an array using pointers.

```
#include <stdio.h>
void delete_element(int *arr, int *size, int position) {
    if (position < 0 || position >= *size) {
        printf("Invalid position\n");
        return;
    }
    for (int i = position; i < *size - 1; ++i) {
        arr[i] = arr[i + 1];
    }
    (*size)--;
}
int main() {
    int size;
    printf("Enter the size of the array: ");
    scanf("%d", &size);
    int arr[size];
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < size; ++i) {
        scanf("%d", &arr[i]);
    }
    int position;
    printf("Enter the position of the element to delete (0 to %d): ", size - 1);
    scanf("%d", &position);
    delete_element(arr, &size, position);
    printf("Array after deletion:\n");
    for (int i = 0; i < size; ++i) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

OUTPUT

```
Enter the size of the array: 5

Enter the elements of the array:

35

57

79

20

51

Enter the position of the element to delete (0 to 4): 2

Array after deletion:

35 57 20 51
```

To reverse an array using pointers:

```
#include <stdio.h>


void reverse_array(int *arr, int size) {

    int *left = arr;          // Pointer to the first element

    int *right = arr + size - 1; // Pointer to the last element


    // Swap elements pointed to by left and right pointers

    while (left < right) {

        int temp = *left;

        *left = *right;

        *right = temp;

        left++;
        right--;
    }
}

int main() {
    int size;
    printf("Enter the size of the array: ");
    scanf("%d", &size);

    int arr[size];

    printf("Enter the elements of the array:\n");
    for (int i = 0; i < size; ++i) {
        scanf("%d", &arr[i]);
    }

    // Reverse the array
    reverse_array(arr, size);

    printf("Reversed array:\n");
    for (int i = 0; i < size; ++i) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

OUTPUT:

```
Enter the size of the array: 4

Enter the elements of the array:

24

42

67

76

Reversed array:

76 67 42 24
```

ANALYTICAL SESSION

To implement the Dutch National Flag problem, which sorts an array of 0s, 1s, and 2s

```
#include <stdio.h>
void swap(int *a, int *b) {
   int temp = *a;
   *a = *b;
   *b = temp;
}
void sortColors(int* nums, int numsSize) {
   int low = 0, mid = 0, high = numsSize - 1;
   while (mid <= high) {
      switch(nums[mid]) {
         case 0:
            swap(&nums[low], &nums[mid]);
            low++;
            mid++;
            break;
         case 1:
            mid++;
            break;
         case 2:
            swap(&nums[mid], &nums[high]);
            high--;
            break;
      }
   }
}
int main() {
   int size;
   printf("Enter the size of the array: ");
   scanf("%d", &size);
   int nums[size];
   printf("Enter the elements of the array (0s, 1s, and 2s only):\n");
   for (int i = 0; i < size; i++) {
      scanf("%d", &nums[i]);
```

```
    }
    printf("Before sorting: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", nums[i]);
    }

    sortColors(nums, size);

    printf("\nAfter sorting: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", nums[i]);
    }

    return 0;
}
```

OUTPUT:

```
Enter the size of the array: 5
Enter the elements of the array (0s, 1s, and 2s only):
1
0
2
1
0
Before sorting: 1 0 2 1 0
After sorting: 0 0 1 1 2
```

Implements a function to check whether a given string is an anagram of another string

```
#include <stdio.h>
#include <string.h>
#define MAX_CHAR 256
int areAnagrams(const char *str1, const char *str2) {
    int len1 = strlen(str1);
    int len2 = strlen(str2);
    if (len1 != len2)
        return 0;
    int count[MAX_CHAR] = {0};
    for (int i = 0; i < len1; i++)
        count[str1[i]]++;
    for (int i = 0; i < len2; i++) {
        count[str2[i]]--;
        if (count[str2[i]] < 0)
            return 0;
    }
    return 1;
}
int main() {
    char str1[100], str2[100];
    printf("Enter the first string: ");
    scanf("%s", str1);
    printf("Enter the second string: ");
    scanf("%s", str2);
    if (areAnagrams(str1, str2))
        printf("%s and %s are anagrams.\n", str1, str2);
    else
        printf("%s and %s are not anagrams.\n", str1, str2);
    return 0;
}
```

OUTPUT:

```
Enter the first string: MANGO
Enter the second string: GOMAN
MANGO and GOMAN are anagrams.
```

Implement a function to find the median of two sorted arrays of equal size

```
#include <stdio.h>
double findMedianSortedArrays(int arr1[], int arr2[], int n) {
    int merged[2 * n];
    int i = 0, j = 0, k = 0;
    while (i < n && j < n) {
        if (arr1[i] < arr2[j])
            merged[k++] = arr1[i++];
        else
            merged[k++] = arr2[j++];
    }
    while (i < n)
        merged[k++] = arr1[i++];
    while (j < n)
        merged[k++] = arr2[j++];
    if (2 * n % 2 == 0)
        return (merged[n - 1] + merged[n]) / 2.0;
    else
        return merged[n];
}
int main() {
    int n;
    printf("Enter the size of the arrays: ");
    scanf("%d", &n);
    int arr1[n], arr2[n];
    printf("Enter the elements of the first sorted array:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr1[i]);
    }
    printf("Enter the elements of the second sorted array:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr2[i]);
    }
    double median = findMedianSortedArrays(arr1, arr2, n);
    printf("Median of the two sorted arrays is: %.2lf\n", median);
    return 0;
}
```

OUTPUT:

```
Enter the size of the arrays: 4
Enter the elements of the first sorted array:
6
4
```

```
9
7
Enter the elements of the second sorted array:
8
5
3
2
Median of the two sorted arrays is: 4.00
```

Implements a function to find the minimum number of jumps needed to reach the end of an array

```
#include <stdio.h>
int minJumps(int arr[], int size) {
    if (size <= 1)
        return 0;
    int maxReach = arr[0];
    int steps = arr[0];
    int jumps = 1;
    for (int i = 1; i < size; i++) {
        if (i > maxReach)
            return -1;
        if (i == size - 1)
            return jumps;
        maxReach = (i + arr[i] > maxReach) ? i + arr[i] : maxReach;
        steps--;
        if (steps == 0) {
            jumps++;
            if (i >= maxReach)
                return -1;
            steps = maxReach - i;
        }
    }
    return -1;
}
int main() {
    int size;
    printf("Enter the size of the array: ");
    scanf("%d", &size);
    int arr[size];
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }
    int minJumpsNeeded = minJumps(arr, size);
    if (minJumpsNeeded != -1)
        printf("Minimum number of jumps needed: %d\n", minJumpsNeeded);
    else
        printf("Cannot reach the end of the array.\n");
    return 0;
}
```

OUTPUT:

```
Enter the size of the array: 4
Enter the elements of the array:
87
57
69
20
Minimum number of jumps needed: 1
```

Implements a function to find the length of the longest substring containing only distinct characters:

```
#include <stdio.h>
#include <string.h>
#define MAX_CHARS 256
int longestSubstringLength(char *str) {
    int n = strlen(str);
    int maxLength = 0;
    int start = 0;
    int visited[MAX_CHARS];
    memset(visited, -1, sizeof(visited));
    start = 0;
    for (int i = 0; i < n; i++) {
        if (visited[str[i]] != -1) {
            start = (start > visited[str[i]] + 1) ? start : visited[str[i]] + 1;
        }
        visited[str[i]] = i;
        maxLength = (i - start + 1 > maxLength) ? i - start + 1 : maxLength;
    }
    return maxLength;
}
int main() {
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);
    int length = longestSubstringLength(str);
    printf("Length of the longest substring with distinct characters: %d\n", length);
    return 0;
}
```

OUTPUT:

```
Enter a string: ABCCBABBAC
Length of the longest substring with distinct characters: 3
```

Finds the kth largest element in an unsorted array using the quickselect algorithm

```
#include <stdio.h>
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
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) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}
int quickSelect(int arr[], int low, int high, int k) {
    if (low <= high) {
        int partitionIndex = partition(arr, low, high);
        if (partitionIndex == k)
            return arr[partitionIndex];
        else if (partitionIndex > k)
            return quickSelect(arr, low, partitionIndex - 1, k);
        else
            return quickSelect(arr, partitionIndex + 1, high, k);
    }
    return -1;
}
int main() {
    int size;
    printf("Enter the size of the array: ");
    scanf("%d", &size);
    int arr[size];
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }
    int k;
    printf("Enter the value of k: ");
    scanf("%d", &k);
    if (k >= 1 && k <= size) {
        int kthLargest = quickSelect(arr, 0, size - 1, size - k);
        if (kthLargest != -1)
            printf("The %dth largest element is: %d\n", k, kthLargest);
        else
            printf("Invalid input!\n");
    } else {
        printf("Invalid value of k!\n");
    }
    return 0;
}
```

OUTPUT:

```
Enter the size of the array: 5
Enter the elements of the array:
46
57
68
79
80
Enter the value of k: 3
The 3th largest element is: 68
```

The maximum sum subarray within a circular array, we can utilize the Kadane's algorithm, which is typically used to find the maximum subarray sum in a linear array

```
#include <stdio.h>
int max(int a, int b) {
    return (a > b) ? a : b;
}
int kadane(int arr[], int size) {
    int maxSum = arr[0];
    int currentMax = arr[0];
    for (int i = 1; i < size; i++) {
        currentMax = max(arr[i], currentMax + arr[i]);
        maxSum = max(maxSum, currentMax);
    }
    return maxSum;
}
int maxCircularSum(int arr[], int size) {
    int maxLinearSum = kadane(arr, size);
    int arraySum = 0;
    for (int i = 0; i < size; i++) {
        arraySum += arr[i];
        arr[i] = -arr[i];
    }
    int maxCircularSum = arraySum + kadane(arr, size);
    return (maxCircularSum > 0) ? max(maxCircularSum, maxLinearSum) :
maxLinearSum;
}
int main() {
    int size;
    printf("Enter the size of the circular array: ");
    scanf("%d", &size);
    int arr[size];
    printf("Enter the elements of the circular array:\n");
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }
    int maxSum = maxCircularSum(arr, size);
    printf("Maximum sum of a subarray within the circular array: %d\n", maxSum);
    return 0;
}
```

OUTPUT:

```
Enter the size of the circular array: 4
Enter the elements of the circular array:
8
-1
3
4
Maximum sum of a subarray within the circular array: 15
```

Implement a function that finds the longest palindromic substring in a given string

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char* longestPalindrome(char* s) {
    int n = strlen(s);
    if (n == 0) return "";
    int dp[n][n];
    memset(dp, 0, sizeof(dp));
    int start = 0;
    int maxLength = 1;
    for (int i = 0; i < n; i++)
        dp[i][i] = 1;
    for (int i = 0; i < n - 1; i++) {
        if (s[i] == s[i + 1]) {
            dp[i][i + 1] = 1;
            start = i;
            maxLength = 2;
        }
    }
    for (int len = 3; len <= n; len++) {
        for (int i = 0; i <= n - len; i++) {
            int j = i + len - 1;
            if (dp[i + 1][j - 1] && s[i] == s[j]) {
                dp[i][j] = 1;
                if (len > maxLength) {
                    start = i;
                    maxLength = len;
                }
            }
        }
    }
    char* result = malloc(maxLength + 1);
    strncpy(result, s + start, maxLength);
    result[maxLength] = '\0';
    return result;
}
int main() {
    char s[100];
    printf("Enter a string: ");
    scanf("%s", s);
    char* longestPalindromicSubstring = longestPalindrome(s);
    printf("Longest palindromic substring: %s\n", longestPalindromicSubstring);
    free(longestPalindromicSubstring);
    return 0;
```

```
}
```

OUTPUT:

```
Enter a string: BABAD
Longest palindromic substring: BAB
```

Implements a function to count the number of subarrays with a sum less than a given value

```
#include <stdio.h>
#include <stdlib.h>
int countSubarrays(int arr[], int n, int target) {
    int count = 0;
    for (int i = 0; i < n; i++) {
        int sum = 0;
        for (int j = i; j < n; j++) {
            sum += arr[j];
            if (sum < target)
                count++;
            else
                break;
        }
    }
    return count;
}
int main() {
    int n, target;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);
    int *arr = (int *)malloc(n * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed!");
        return 1;
    }
    printf("Enter the elements of the array: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter the target sum: ");
    scanf("%d", &target);
    int result = countSubarrays(arr, n, target);
    printf("Number of subarrays with sum less than %d: %d\n", target, result);
    free(arr);
    return 0;
}
```

OUTPUT:

```
Enter the number of elements in the array: 5
Enter the elements of the array: 1
2
3
4
6
Enter the target sum: 10
Number of subarrays with sum less than 10: 10
```

Write a C program to implement a function that Returns the count of distinct substrings of a given string

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

#define MAX_LENGTH 100

int countDistinctSubstrings(char *str) {
    int len = strlen(str);
    int count = 0;
    int hash[MAX_LENGTH] = {0}; // Assuming the maximum length of the string is
MAX_LENGTH

    // Iterate through all substrings
    for (int i = 0; i < len; i++) {
        for (int j = i; j < len; j++) {
            int hashValue = 0;
            for (int k = i; k <= j; k++) {
                hashValue = hashValue * 10 + (str[k] - 'a' + 1); // Assuming lowercase letters
only
            }
            if (!hash[hashValue]) {
                hash[hashValue] = 1;
                count++;
            }
        }
    }
    return count;
}

int main() {
    char str[MAX_LENGTH];
    printf("Enter a string: ");
    scanf("%s", str);

    int result = countDistinctSubstrings(str);
    printf("Count of distinct substrings: %d\n", result);

    return 0;
}
```

OUTPUT:

```
Enter a string: BANANA
Segmentation fault
```