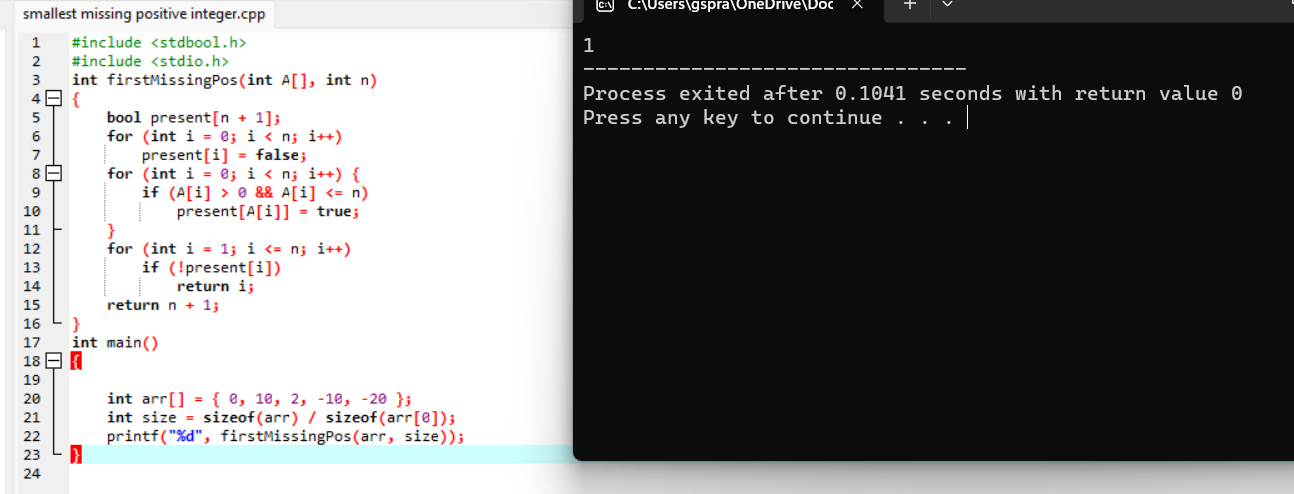
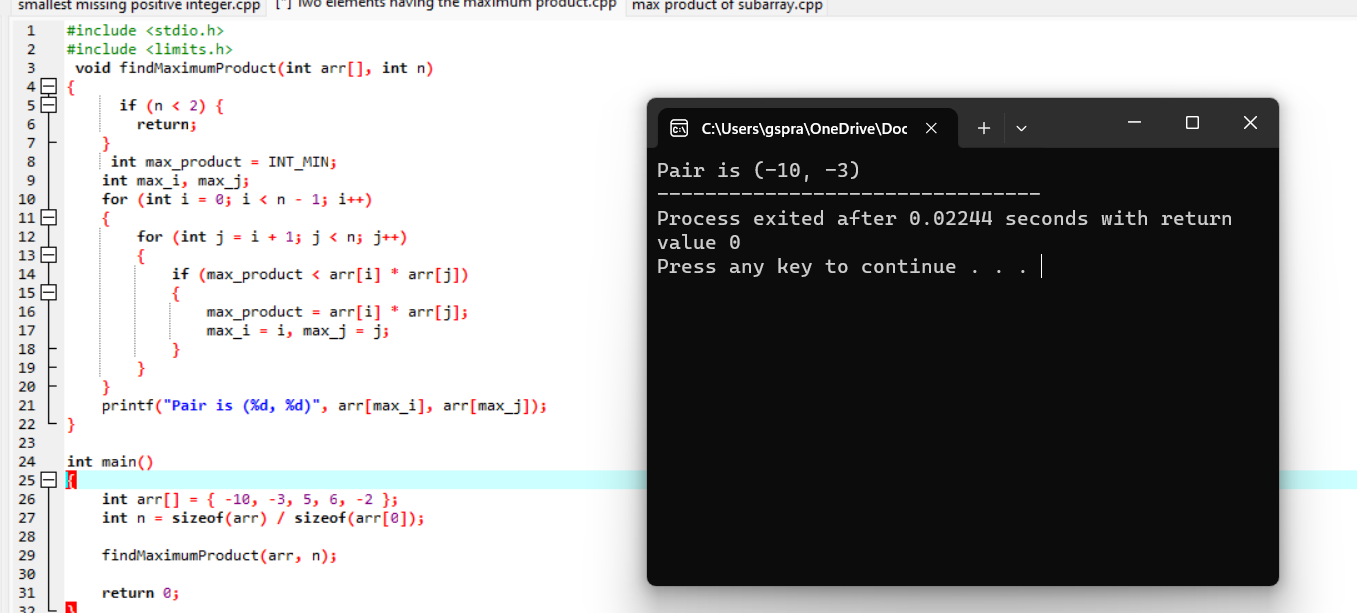
8-9

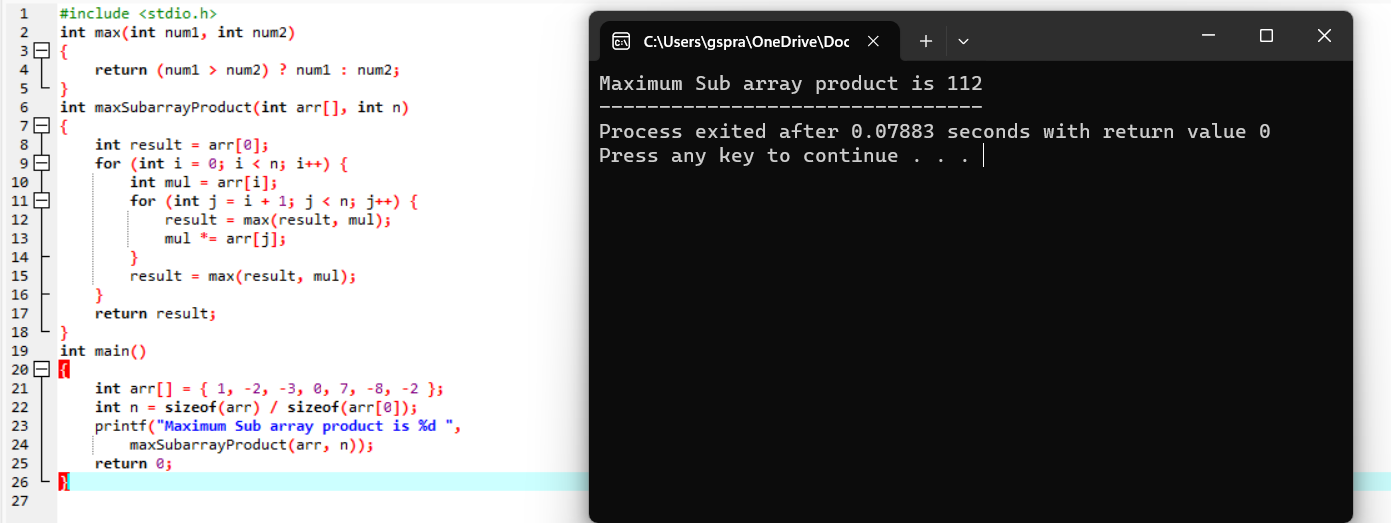
1. Smallest missing possitive integer.



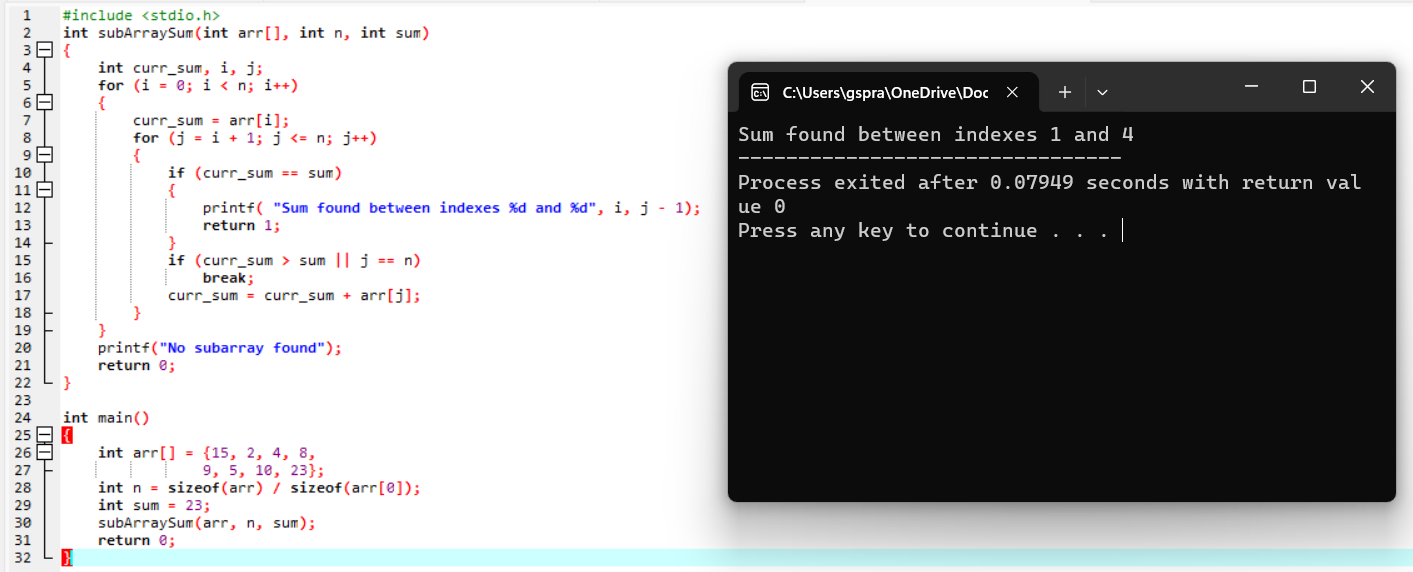
2.The two elements that have the maximum product.



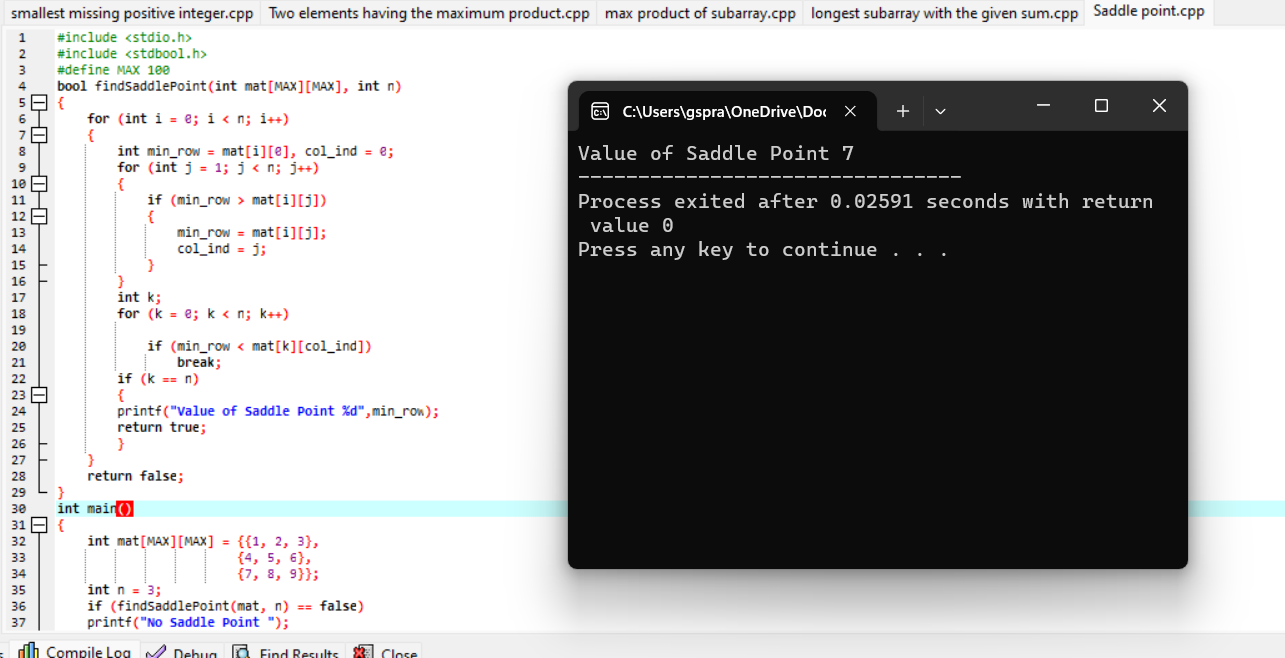
3.max product of subarray.



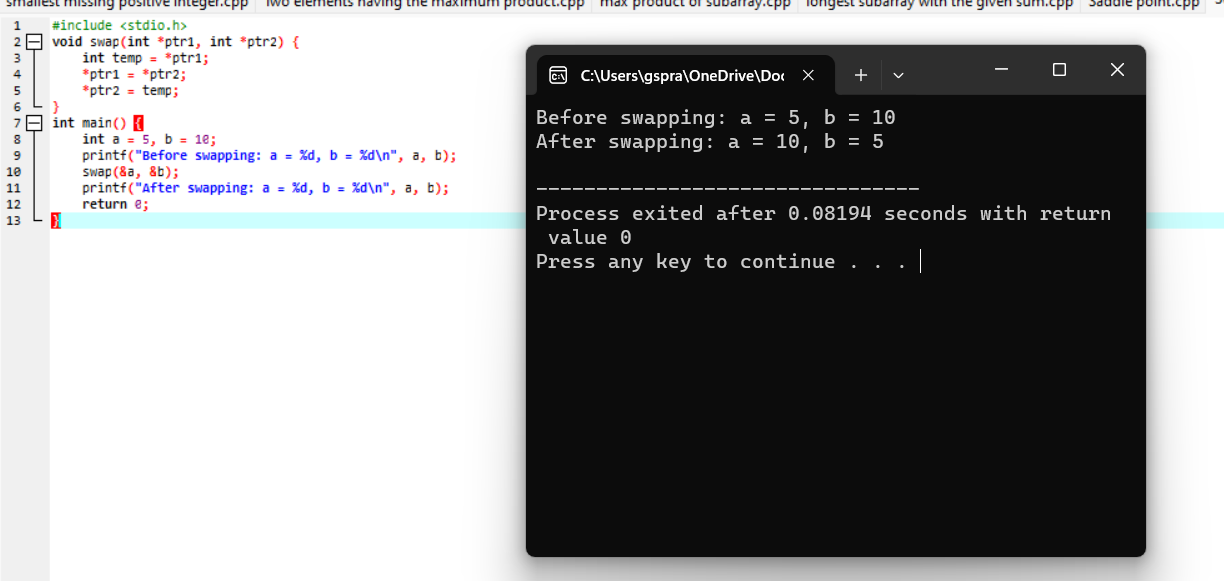
4. Longest subarray with the given sum.



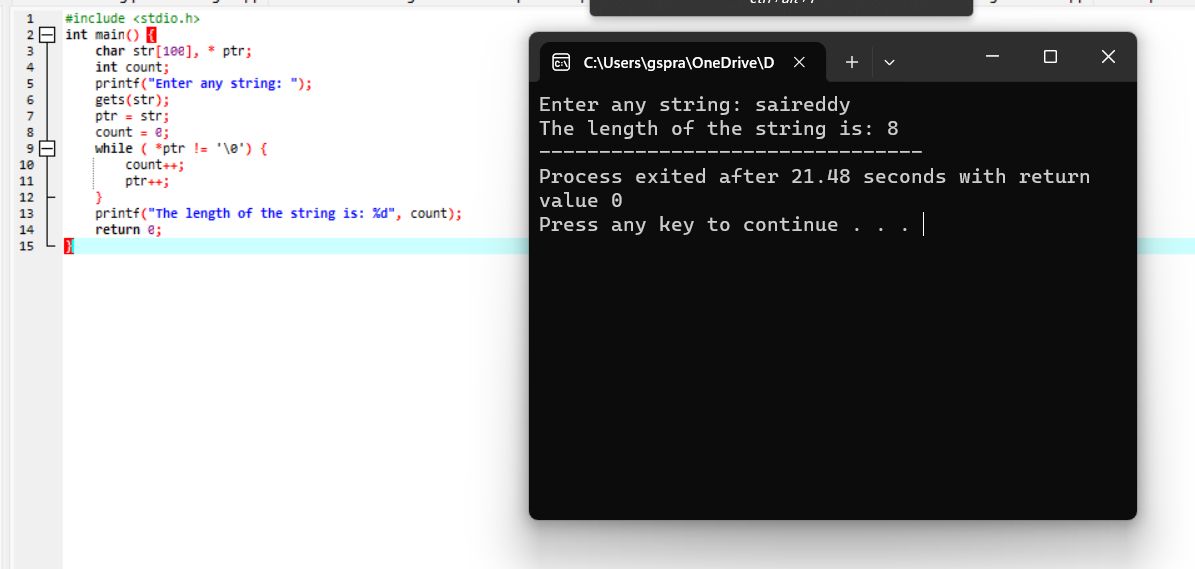
5. Saddle point.



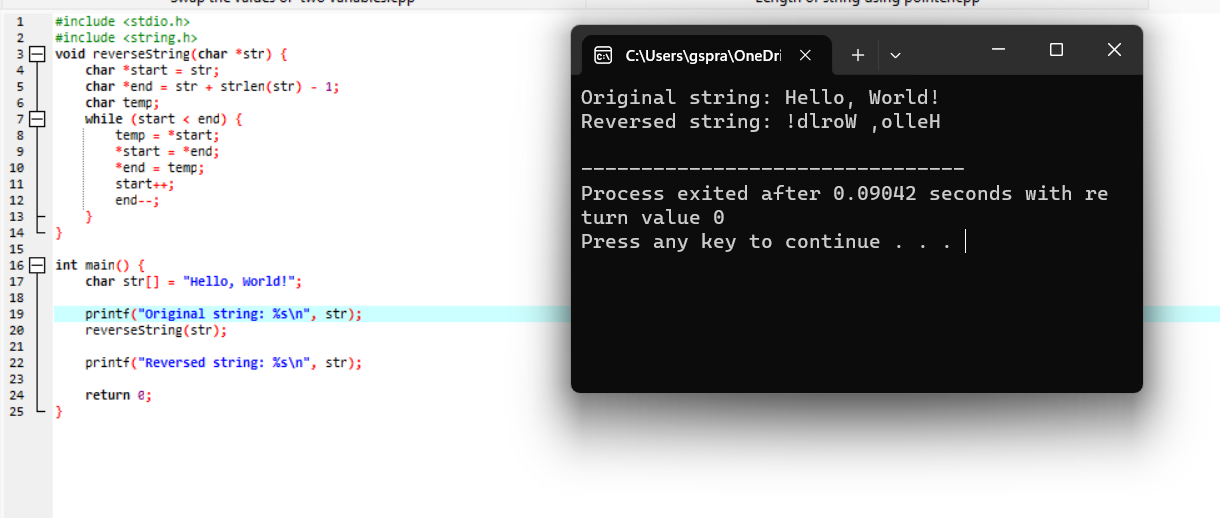
6. swap the values of two variables using pointers.



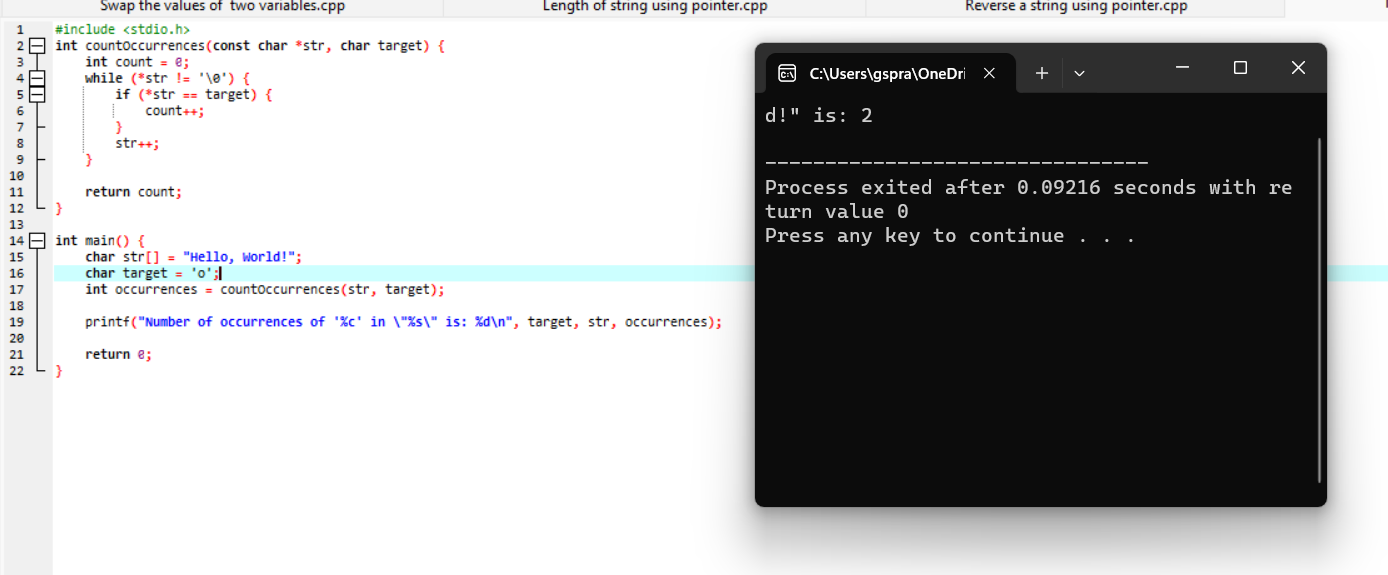
7.Length of string using pointer.



8.Reverse a string using pointer.



9. Number of occurrences of a character in a string using pointers.



Analytical programs:

**1.Longest substring without repeating characters.**

#include <stdio.h>

#include <string.h>

#define MAX\_CHARS 256

int max(int a, int b) {

return (a > b) ? a : b;

}

void longestUniqueSubsttr(char \*str) {

int n = strlen(str);

int visited[MAX\_CHARS];

int start = 0;

int currentLength = 1;

int maxLength = 1;

int previousIndex;

for (int i = 0; i < MAX\_CHARS; i++)

visited[i] = -1;

visited[str[0]] = 0;

for (int i = 1; i < n; i++) {

previousIndex = visited[str[i]];

if (previousIndex == -1 || i - currentLength > previousIndex)

currentLength++;

else {

if (currentLength > maxLength) {

maxLength = currentLength;

start = i - maxLength;

}

currentLength = i - previousIndex;

}

visited[str[i]] = i;

}

if (currentLength > maxLength) {

maxLength = currentLength;

start = n - maxLength;

}

printf("The longest substring without repeating characters is: ");

for (int i = start; i < start + maxLength; i++)

printf("%c", str[i]);

printf("\n");

}

int main() {

char str[] = "abcabcbb";

longestUniqueSubsttr(str);

return 0;

}

**Output:**

The length of the longest substring without repeating characters is: 3

**2.Maximum product subarray in a given array of integers.**

#include <stdio.h>

int max(int a, int b) {

return (a > b) ? a : b;

}

int min(int a, int b) {

return (a < b) ? a : b;

}

int maxProductSubarray(int arr[], int n) {

if (n == 0) {

return 0;

}

int maxEndingHere = arr[0];

int minEndingHere = arr[0];

int maxSoFar = arr[0];

for (int i = 1; i < n; i++) {

int temp = maxEndingHere;

maxEndingHere = max(arr[i], max(arr[i] \* maxEndingHere, arr[i] \* minEndingHere));

minEndingHere = min(arr[i], min(arr[i] \* temp, arr[i] \* minEndingHere));

maxSoFar = max(maxSoFar, maxEndingHere);

}

return maxSoFar;

}

int main() {

int arr[] = { 3, 4};

int n = sizeof(arr) / sizeof(arr[0]);

int maxProd = maxProductSubarray(arr, n);

printf("Maximum product subarray is: %d\n", maxProd);

return 0;

}

**Output:**

Maximum product subarray is: 12

**3.Rotates a given array of integers to the right by k steps.**

#include <stdio.h>

int main()

{

int arr[] = { 1, 3, 5, 7, 9, 11 };

int n = sizeof(arr) / sizeof(arr[0]);

int k = 3;

k = k % n;

int i, j;

for (i = n - k, j = n - 1; i < j; i++, j--) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

for (i = 0, j = n - k - 1; i < j; i++, j--) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

for (i = 0, j = n - 1; i < j; i++, j--) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

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

printf("%d ", arr[i]);

}

return 0;

}

**Output:**

7 9 11 1 3 5

--------------------------------

Process exited after 0.09181 seconds with return value 0

Press any key to continue . . .

**4.Longest common prefix string amongst an array of strings.**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int min(int a, int b) {

return (a < b) ? a : b;

}

char\* longestCommonPrefix(char\*\* strs, int strsSize) {

if (strsSize == 0)

return "";

int minLen = strlen(strs[0]);

for (int i = 1; i < strsSize; ++i)

minLen = min(minLen, strlen(strs[i]));

int i, j;

for (i = 0; i < minLen; ++i) {

for (j = 1; j < strsSize; ++j) {

if (strs[j][i] != strs[j - 1][i])

break;

}

if (j != strsSize)

break;

}

char\* commonPrefix = (char\*)malloc(sizeof(char) \* (i + 1));

strncpy(commonPrefix, strs[0], i);

commonPrefix[i] = '\0';

return commonPrefix;

}

int main() {

char \*strings[] = {"flower", "flow", "flight"};

int size = sizeof(strings) / sizeof(strings[0]);

char \*prefix = longestCommonPrefix(strings, size);

printf("Longest common prefix: %s\n", prefix);

free(prefix);

return 0;

}

**Output:**

Longest common prefix:fl

**5.combinations of a given length k.**

#include <stdio.h>

void printCombination(int arr[], int n, int r, int index, int data[], int i) {

if (index == r) {

for (int j = 0; j < r; j++)

printf("%d ", data[j]);

printf("\n");

return;

}

if (i >= n)

return;

data[index] = arr[i];

printCombination(arr, n, r, index + 1, data, i + 1);

while (i < n - 1 && arr[i] == arr[i + 1])

i++;

printCombination(arr, n, r, index, data, i + 1);

}

void combinations(int arr[], int n, int r) {

int data[r];

printCombination(arr, n, r, 0, data, 0);

}

int main() {

int arr[] = {1, 2, 3, 4};

int n = sizeof(arr) / sizeof(arr[0]);

int k = 2;

printf("Distinct combinations of length %d:\n", k);

combinations(arr, n, k);

return 0;

}

**Output**: Distinct combinations of length 2:

1 2

1 3

1 4

2 3

2 4

3 4

**6. removes all duplicates from a sorted array.**

#include <stdio.h>

int removeDuplicates(int arr[], int n) {

if (n == 0 || n == 1)

return n;

int j = 0;

for (int i = 0; i < n - 1; i++) {

if (arr[i] != arr[i + 1]) {

arr[j++] = arr[i];

}

}

arr[j++] = arr[n - 1];

return j;

}

int main() {

int arr[] = {1, 2, 2, 3, 4, 4, 4, 5};

int n = sizeof(arr) / sizeof(arr[0]);

printf("Original array: ");

for (int i = 0; i < n; i++)

printf("%d ", arr[i]);

printf("\n");

n = removeDuplicates(arr, n);

printf("Array after removing duplicates: ");

for (int i = 0; i < n; i++)

printf("%d ", arr[i]);

printf("\n");

return 0;

}

**Output:**

Original array: 1 2 2 3 4 4 4 5

Array after removing duplicates: 1 2 3 4 5

**7. Implement a function that determines if a given Sudoku board is valid.**

#include <stdio.h>

#include <stdbool.h>

#define SIZE 9

bool isValidRow(char board[SIZE][SIZE], int row) {

bool visited[SIZE + 1] = {false};

for (int col = 0; col < SIZE; col++) {

if (board[row][col] != '.') {

int num = board[row][col] - '0';

if (visited[num])

return false;

visited[num] = true;

}

}

return true;

}

bool isValidCol(char board[SIZE][SIZE], int col) {

bool visited[SIZE + 1] = {false};

for (int row = 0; row < SIZE; row++) {

if (board[row][col] != '.') {

int num = board[row][col] - '0';

if (visited[num])

return false;

visited[num] = true;

}

}

return true;

}

bool isValidSubgrid(char board[SIZE][SIZE], int startRow, int startCol) {

bool visited[SIZE + 1] = {false};

for (int row = startRow; row < startRow + 3; row++) {

for (int col = startCol; col < startCol + 3; col++) {

if (board[row][col] != '.') {

int num = board[row][col] - '0';

if (visited[num])

return false;

visited[num] = true;

}

}

}

return true;

}

bool isValidSudoku(char board[SIZE][SIZE]) {

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

if (!isValidRow(board, i))

return false;

}

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

if (!isValidCol(board, i))

return false;

}

for (int startRow = 0; startRow < SIZE; startRow += 3) {

for (int startCol = 0; startCol < SIZE; startCol += 3) {

if (!isValidSubgrid(board, startRow, startCol))

return false;

}

}

return true;

}

int main() {

char board[SIZE][SIZE] = {

{'5', '3', '.', '.', '7', '.', '.', '.', '.'},

{'6', '.', '.', '1', '9', '5', '.', '.', '.'},

{'.', '9', '8', '.', '.', '.', '.', '6', '.'},

{'8', '.', '.', '.', '6', '.', '.', '.', '3'},

{'4', '.', '.', '8', '.', '3', '.', '.', '1'},

{'7', '.', '.', '.', '2', '.', '.', '.', '6'},

{'.', '6', '.', '.', '.', '.', '2', '8', '.'},

{'.', '.', '.', '4', '1', '9', '.', '.', '5'},

{'.', '.', '.', '.', '8', '.', '.', '7', '9'}

};

if (isValidSudoku(board))

printf("The Sudoku board is valid.\n");

else

printf("The Sudoku board is not valid.\n");

return 0;

}

**Output:** The Sudoku board is valid.

**8.Implement a function that returns the index of the first occurrence of a target value in a sorted array**

#include <stdio.h>

int findFirstOccurrence(int arr[], int size, int target) {

int low = 0;

int high = size - 1;

int result = -1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == target) {

result = mid;

high = mid - 1;

} else if (arr[mid] < target) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return result;

}

int main() {

int arr[] = {1, 2, 3, 4, 4, 4, 5, 6};

int size = sizeof(arr) / sizeof(arr[0]);

int target = 3;

int index = findFirstOccurrence(arr, size, target);

if (index != -1) {

printf("First occurrence of %d is at index %d\n", target, index);

} else {

printf("%d not found in the array\n", target);

}

return 0;

}

**Output:**

First occurrence of 3 is at index 2