1. #include <stdio.h>

#include <stdlib.h>

// Function to find the two indices

int\* twoSum(int\* nums, int numsSize, int target, int\* returnSize) {

// Allocate memory for the return array

int\* result = (int\*)malloc(2 \* sizeof(int));

// Iterate through the array

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

for (int j = i + 1; j < numsSize; j++) {

if (nums[i] + nums[j] == target) {

result[0] = i;

result[1] = j;

\*returnSize = 2; // The size of the returned array is 2

return result;

}

}

}

\*returnSize = 0; // If no solution is found, return size is 0

return NULL; // Return NULL if no solution is found

}

int main() {

int nums1[] = {2, 7, 11, 15};

int target1 = 9;

int returnSize1;

int\* result1 = twoSum(nums1, 4, target1, &returnSize1);

if (result1 != NULL) {

printf("Example 1: [%d, %d]\n", result1[0], result1[1]);

free(result1); // Free the allocated memory

}

int nums2[] = {3, 2, 4};

int target2 = 6;

int returnSize2;

int\* result2 = twoSum(nums2, 3, target2, &returnSize2);

if (result2 != NULL) {

printf("Example 2: [%d, %d]\n", result2[0], result2[1]);

free(result2); // Free the allocated memory

}

int nums3[] = {3, 3};

int target3 = 6;

int returnSize3;

int\* result3 = twoSum(nums3, 2, target3, &returnSize3);

if (result3 != NULL) {

printf("Example 3: [%d, %d]\n", result3[0], result3[1]);

free(result3); // Free the allocated memory

}

return 0;

}

2. #include <stdio.h>

#include <stdlib.h>

// Definition for singly-linked list.

struct ListNode {

int val;

struct ListNode \*next;

};

// Function to create a new node with a given value

struct ListNode\* createNode(int val) {

struct ListNode\* newNode = (struct ListNode\*)malloc(sizeof(struct ListNode));

newNode->val = val;

newNode->next = NULL;

return newNode;

}

// Function to add two numbers represented by linked lists

struct ListNode\* addTwoNumbers(struct ListNode\* l1, struct ListNode\* l2) {

struct ListNode dummy;

struct ListNode\* current = &dummy;

dummy.next = NULL;

int carry = 0;

while (l1 != NULL || l2 != NULL || carry) {

int sum = carry;

if (l1 != NULL) {

sum += l1->val;

l1 = l1->next;

}

if (l2 != NULL) {

sum += l2->val;

l2 = l2->next;

}

carry = sum / 10;

struct ListNode\* newNode = createNode(sum % 10);

current->next = newNode;

current = current->next;

}

return dummy.next;

}

// Helper function to print the linked list

void printList(struct ListNode\* node) {

while (node != NULL) {

printf("%d", node->val);

if (node->next != NULL) {

printf(" -> ");

}

node = node->next;

}

printf("\n");

}

// Helper function to free the linked list

void freeList(struct ListNode\* node) {

while (node != NULL) {

struct ListNode\* temp = node;

node = node->next;

free(temp);

}

}

int main() {

// Example 1: l1 = [2, 4, 3], l2 = [5, 6, 4]

struct ListNode\* l1 = createNode(2);

l1->next = createNode(4);

l1->next->next = createNode(3);

struct ListNode\* l2 = createNode(5);

l2->next = createNode(6);

l2->next->next = createNode(4);

struct ListNode\* result = addTwoNumbers(l1, l2);

printList(result);

// Free the allocated memory

freeList(l1);

freeList(l2);

freeList(result);

return 0;

}

3. #include <stdio.h>

#include <string.h>

// Function to find the length of the longest substring without repeating characters

int lengthOfLongestSubstring(char \* s) {

int n = strlen(s);

if (n == 0) return 0;

int maxLen = 0;

int start = 0;

int charIndex[128]; // Array to store the last index of each character (ASCII range)

// Initialize the array with -1

for (int i = 0; i < 128; i++) {

charIndex[i] = -1;

}

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

char currentChar = s[end];

if (charIndex[currentChar] >= start) {

start = charIndex[currentChar] + 1;

}

charIndex[currentChar] = end;

int currentLen = end - start + 1;

if (currentLen > maxLen) {

maxLen = currentLen;

}

}

return maxLen;

}

int main() {

char s1[] = "abcabcbb";

char s2[] = "bbbbb";

char s3[] = "pwwkew";

printf("Input: \"%s\", Output: %d\n", s1, lengthOfLongestSubstring(s1)); // Output: 3

printf("Input: \"%s\", Output: %d\n", s2, lengthOfLongestSubstring(s2)); // Output: 1

printf("Input: \"%s\", Output: %d\n", s3, lengthOfLongestSubstring(s3)); // Output: 3

return 0;

}

4. #include <stdio.h>

#include <stdlib.h>

// Function to find the median of two sorted arrays

double findMedianSortedArrays(int\* nums1, int nums1Size, int\* nums2, int nums2Size) {

int\* mergedArray = (int\*)malloc((nums1Size + nums2Size) \* sizeof(int));

int i = 0, j = 0, k = 0;

// Merge the two sorted arrays

while (i < nums1Size && j < nums2Size) {

if (nums1[i] < nums2[j]) {

mergedArray[k++] = nums1[i++];

} else {

mergedArray[k++] = nums2[j++];

}

}

// Copy remaining elements of nums1, if any

while (i < nums1Size) {

mergedArray[k++] = nums1[i++];

}

// Copy remaining elements of nums2, if any

while (j < nums2Size) {

mergedArray[k++] = nums2[j++];

}

// Find the median

int totalSize = nums1Size + nums2Size;

double median;

if (totalSize % 2 == 0) {

median = (mergedArray[totalSize / 2 - 1] + mergedArray[totalSize / 2]) / 2.0;

} else {

median = mergedArray[totalSize / 2];

}

// Free the allocated memory for the merged array

free(mergedArray);

return median;

}

int main() {

int nums1[] = {1, 3};

int nums2[] = {2};

int nums1Size = sizeof(nums1) / sizeof(nums1[0]);

int nums2Size = sizeof(nums2) / sizeof(nums2[0]);

double median = findMedianSortedArrays(nums1, nums1Size, nums2, nums2Size);

printf("Median is %.5f\n", median); // Output: 2.00000

int nums3[] = {1, 2};

int nums4[] = {3, 4};

nums1Size = sizeof(nums3) / sizeof(nums3[0]);

nums2Size = sizeof(nums4) / sizeof(nums4[0]);

median = findMedianSortedArrays(nums3, nums1Size, nums4, nums2Size);

printf("Median is %.5f\n", median); // Output: 2.50000

return 0;

}

5. #include <stdio.h>

#include <string.h>

#include <stdlib.h>

// Function to find the longest palindromic substring

char\* longestPalindrome(char\* s) {

int n = strlen(s);

if (n == 0) return "";

int start = 0, maxLength = 1;

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

// Check for odd length palindromes centered at i

int low = i, high = i;

while (low >= 0 && high < n && s[low] == s[high]) {

if (high - low + 1 > maxLength) {

start = low;

maxLength = high - low + 1;

}

low--;

high++;

}

// Check for even length palindromes centered between i and i+1

low = i, high = i + 1;

while (low >= 0 && high < n && s[low] == s[high]) {

if (high - low + 1 > maxLength) {

start = low;

maxLength = high - low + 1;

}

low--;

high++;

}

}

char\* result = (char\*)malloc((maxLength + 1) \* sizeof(char));

strncpy(result, s + start, maxLength);

result[maxLength] = '\0';

return result;

}

int main() {

char s1[] = "babad";

char s2[] = "cbbd";

char\* result1 = longestPalindrome(s1);

printf("Longest palindromic substring of \"%s\" is \"%s\"\n", s1, result1);

free(result1);

char\* result2 = longestPalindrome(s2);

printf("Longest palindromic substring of \"%s\" is \"%s\"\n", s2, result2);

free(result2);

return 0;

}

6. #include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Function to perform Zigzag conversion

char\* convert(char\* s, int numRows) {

if (numRows == 1) return s; // If there's only one row, return the string as it is

int len = strlen(s);

char\*\* rows = (char\*\*)malloc(numRows \* sizeof(char\*));

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

rows[i] = (char\*)malloc((len + 1) \* sizeof(char));

rows[i][0] = '\0';

}

int currRow = 0;

int goingDown = 0;

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

strncat(rows[currRow], &s[i], 1); // Add character to the current row

if (currRow == 0 || currRow == numRows - 1) goingDown = !goingDown;

currRow += goingDown ? 1 : -1;

}

char\* result = (char\*)malloc((len + 1) \* sizeof(char));

result[0] = '\0';

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

strcat(result, rows[i]);

free(rows[i]); // Free memory allocated for each row

}

free(rows); // Free memory allocated for rows array

return result;

}

int main() {

char s[] = "PAYPALISHIRING";

int numRows = 3;

char\* result = convert(s, numRows);

printf("Converted string: %s\n", result);

free(result); // Free memory allocated for the result string

return 0;

}

7. #include <stdio.h>

#include <limits.h>

// Function to reverse an integer

int reverse(int x) {

int reversed = 0;

while (x != 0) {

int pop = x % 10;

x /= 10;

// Check for overflow before it happens

if (reversed > INT\_MAX / 10 || (reversed == INT\_MAX / 10 && pop > 7)) return 0;

if (reversed < INT\_MIN / 10 || (reversed == INT\_MIN / 10 && pop < -8)) return 0;

reversed = reversed \* 10 + pop;

}

return reversed;

}

int main() {

int x1 = 123;

int x2 = -123;

int x3 = 1534236469;

printf("Reversed %d is %d\n", x1, reverse(x1)); // Output: 321

printf("Reversed %d is %d\n", x2, reverse(x2)); // Output: -321

printf("Reversed %d is %d\n", x3, reverse(x3)); // Output: 0 (due to overflow)

return 0;

}

8. #include <stdio.h>

#include <limits.h>

#include <ctype.h>

// Function to convert a string to a 32-bit signed integer

int myAtoi(char\* s) {

int i = 0;

int sign = 1;

int result = 0;

// Step 1: Ignore leading whitespace

while (s[i] == ' ') {

i++;

}

// Step 2: Check for the sign

if (s[i] == '-' || s[i] == '+') {

sign = (s[i] == '-') ? -1 : 1;

i++;

}

// Step 3: Convert digits to integer

while (isdigit(s[i])) {

int digit = s[i] - '0';

// Step 5: Check for overflow and underflow

if (result > (INT\_MAX - digit) / 10) {

return (sign == 1) ? INT\_MAX : INT\_MIN;

}

result = result \* 10 + digit;

i++;

}

// Step 4: Apply the sign

return result \* sign;

}

int main() {

char s1[] = "42";

char s2[] = " -42";

char s3[] = "4193 with words";

char s4[] = "words and 987";

char s5[] = "-91283472332";

printf("Input: \"%s\" -> Output: %d\n", s1, myAtoi(s1)); // Output: 42

printf("Input: \"%s\" -> Output: %d\n", s2, myAtoi(s2)); // Output: -42

printf("Input: \"%s\" -> Output: %d\n", s3, myAtoi(s3)); // Output: 4193

printf("Input: \"%s\" -> Output: %d\n", s4, myAtoi(s4)); // Output: 0

printf("Input: \"%s\" -> Output: %d\n", s5, myAtoi(s5)); // Output: -2147483648 (clamped)

return 0;

}

9. #include <stdio.h>

#include <stdbool.h>

// Function to check if an integer is a palindrome

bool isPalindrome(int x) {

// Special cases: negative numbers and numbers ending with 0 (except 0 itself)

if (x < 0 || (x % 10 == 0 && x != 0)) {

return false;

}

int reversed = 0;

while (x > reversed) {

reversed = reversed \* 10 + x % 10;

x /= 10;

}

// For odd-length palindromes, we need to eliminate the middle digit

return x == reversed || x == reversed / 10;

}

int main() {

int x1 = 121;

int x2 = -121;

int x3 = 10;

printf("Input: %d -> Output: %s\n", x1, isPalindrome(x1) ? "true" : "false"); // Output: true

printf("Input: %d -> Output: %s\n", x2, isPalindrome(x2) ? "true" : "false"); // Output: false

printf("Input: %d -> Output: %s\n", x3, isPalindrome(x3) ? "true" : "false"); // Output: false

return 0;

}

10. #include <stdio.h>

#include <stdbool.h>

#include <string.h>

bool isMatch(char \*s, char \*p) {

int m = strlen(s);

int n = strlen(p);

bool dp[m + 1][n + 1];

memset(dp, false, sizeof(dp));

dp[0][0] = true;

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

if (p[j - 1] == '\*') {

dp[0][j] = dp[0][j - 2];

}

}

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

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

if (p[j - 1] == '.' || p[j - 1] == s[i - 1]) {

dp[i][j] = dp[i - 1][j - 1];

} else if (p[j - 1] == '\*') {

dp[i][j] = dp[i][j - 2] || (dp[i - 1][j] && (s[i - 1] == p[j - 2] || p[j - 2] == '.'));

}

}

}

return dp[m][n];

}

int main() {

char s1[] = "aa";

char p1[] = "a";

printf("Input: s = \"%s\", p = \"%s\" -> Output: %s\n", s1, p1, isMatch(s1, p1) ? "true" : "false"); // Output: false

char s2[] = "aa";

char p2[] = "a\*";

printf("Input: s = \"%s\", p = \"%s\" -> Output: %s\n", s2, p2, isMatch(s2, p2) ? "true" : "false"); // Output: true

return 0;

}

11. class Solution {

public:

    int maxArea(vector<int>& height) {

        int left = 0;

        int right = height.size() - 1;

        int maxArea = 0;

        while (left < right) {

            int currentArea = min(height[left], height[right]) \* (right - left);

            maxArea = max(maxArea, currentArea);

            if (height[left] < height[right]) {

                left++;

            } else {

                right--;

            }

        }

        return maxArea;

    }

};

112. class Solution {

public:

    string intToRoman(int num) {

        string ones[] = {"", "I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX"};

        string tens[] = {"", "X", "XX", "XXX", "XL", "L", "LX", "LXX", "LXXX", "XC"};

        string hundreds[] = {"", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC", "CM"};

        string thousands[]= {"", "M", "MM", "MMM"};

        string Roman =  thousands[num / 1000] + hundreds[(num % 1000) / 100] + tens[(num % 100) / 10] + ones[num % 10];

        return Roman;

    }

};

13. class Solution {

public:

    int romanToInt(string s) {

        unordered\_map<char, int> m;

        m['I'] = 1;

        m['V'] = 5;

        m['X'] = 10;

        m['L'] = 50;

        m['C'] = 100;

        m['D'] = 500;

        m['M'] = 1000;

        int ans = 0;

        for(int i = 0; i < s.length(); i++){

            if(m[s[i]] < m[s[i+1]]){

                ans -= m[s[i]];

            }

            else{

                ans += m[s[i]];

            }

        }

        return ans;

    }

};

14. class Solution {

public:

    string longestCommonPrefix(vector<string>& v) {

        string ans="";

        sort(v.begin(),v.end());

        int n=v.size();

        string first=v[0],last=v[n-1];

        for(int i=0;i<min(first.size(),last.size());i++){

            if(first[i]!=last[i]){

                return ans;

            }

            ans+=first[i];

        }

        return ans;

    }

};

**15.** class Solution {

public:

    vector<vector<int>> threeSum(vector<int>& nums) {

        sort(nums.begin(), nums.end());

        vector<vector<int>> result;

        auto first = nums.begin();

        do

        {

            if (first != nums.begin())

            {

                first = upper\_bound(first, nums.end(), \*prev(first, 1));

            }

            auto second = next(first, 1);

            auto third = prev(nums.end(), 1);

            while (second < third)

            {

                const int sum = \*first + \*second + \*third;

                if (sum == 0)

                {

                    result.push\_back({\*first, \*second, \*third});

                    second = prev(upper\_bound(second, prev(third, 1), \*second), 1);

                    third = lower\_bound(next(second, 1), third, \*third);

                    advance(second, 1);

                    advance(third, -1);

                }

                else if (sum < 0)

                {

                    advance(second, 1);

                }

                else

                {

                    advance(third, -1);

                }

            }

            advance(first, 1);

        }

        while (first < prev(nums.end(), 2));

        return result;

    }

};

**16.** class Solution {

public:

    int threeSumClosest(vector<int>& nums, int target) {

        int n=nums.size();

        sort(nums.begin(),nums.end());

        int diff=INT\_MAX;

        int ans;

        for(int i=0;i<n;i++)

        {

            int s=i+1;

            int e=n-1;

            while(s<e)

            {

                if(nums[i]+nums[s]+nums[e]==target)

                    return target;

                 else if(abs(nums[i]+nums[s]+nums[e]-target)<diff)

                 {

                    diff=abs(nums[i]+nums[s]+nums[e]-target);

                    ans=nums[i]+nums[s]+nums[e];

                 }

                 if(nums[i]+nums[s]+nums[e]<target)

                    s++;

                  else if(nums[i]+nums[s]+nums[e]>target)

                        e--;

            }

        }

        return ans;

    }

};

17. class Solution {

public:

    vector<string> letterCombinations(string digits) {

        if (digits.empty()) {

            return {};

        }

        // Mapping of digits to corresponding characters

        unordered\_map<char, string> mp = {

            {'2', "abc"},

            {'3', "def"},

            {'4', "ghi"},

            {'5', "jkl"},

            {'6', "mno"},

            {'7', "pqrs"},

            {'8', "tuv"},

            {'9', "wxyz"}

        };

        string cur = ""; // Current combination

        vector<string> result; // Resulting combinations

        // DFS function to generate combinations

        dfs(digits, 0, mp, cur, result);

        return result;

    }

private:

    // DFS function

    void dfs(string digits, int index, unordered\_map<char, string>& mp, string& cur, vector<string>& result) {

        // Base case: if we've reached the end of the digits string

        if (index == digits.size()) {

            // Add the current combination to the result

            result.push\_back(cur);

            return;

        }

        // Get the characters corresponding to the current digit

        string str = mp[digits[index]];

        // Iterate through each character and explore all possible combinations

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

            // Append the current character to the combination

            cur.push\_back(str[i]);

            // Explore the next digit by recursively calling DFS

            dfs(digits, index + 1, mp, cur, result);

            // Backtrack: remove the last character to explore other combinations

            cur.pop\_back();

        }

    }

};

18. class Solution {

public:

    vector<vector<int>> fourSum(vector<int>& nums, int target) {

        int n=nums.size();

        vector<vector<int>> ans;

        sort(nums.begin(),nums.end());

        for(int i=0;i<n;i++){

            if(i!=0  && nums[i]==nums[i-1]) {

                continue;

            }

            for(int j=i+1;j<n;j++){

            if(j!=i+1 && nums[j]==nums[j-1]) {

                continue;

            }

                int k=j+1;

                int l=n-1;

                while(k<l){

                    long long sum=nums[i];

                    sum+=nums[j];

                    sum+=nums[k];

                    sum+=nums[l];

                    if(sum==target){

                        vector<int>temp={nums[i],nums[j],nums[k],nums[l]};

                        ans.push\_back(temp);

                        k++;

                        l--;

                        while(k<l&& nums[k]==nums[k-1]){

                            k++;

                        }

                        while(k<l&& nums[l]==nums[l+1]){

                            l--;

                        }

                    }

                    else if(sum<target){

                        k++;

                    }else{

                        l--;

                    }

                }

            }

        }

        return ans;

    }

};

19. /\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode() : val(0), next(nullptr) {}

\* ListNode(int x) : val(x), next(nullptr) {}

\* ListNode(int x, ListNode \*next) : val(x), next(next) {}

\* };

\*/

class Solution {

public:

int getLength(ListNode\* &head){

ListNode\* temp = head;

int len = 0;

while(temp != NULL){

len++;

temp = temp->next;

}

return len;

}

ListNode\* removeNthFromEnd(ListNode\* head, int n) {

int len = getLength(head);

int pos = len - n;

int count = 0;

ListNode\* curr = head;

ListNode\* prev = head;

if(pos == 0){

head = head->next;

return head;

}

while(curr != NULL){

if(count == pos){

prev->next = curr->next;

curr->next = NULL;

break;

}

else{

prev = curr;

curr = curr->next;

count++;

}

}

return head;

}

};

20.

class Solution {

public:

bool isValid(string s) {

stack<char> st; // create an empty stack to store opening brackets

for (char c : s) { // loop through each character in the string

if (c == '(' || c == '{' || c == '[') { // if the character is an opening bracket

st.push(c); // push it onto the stack

} else { // if the character is a closing bracket

if (st.empty() || // if the stack is empty or

(c == ')' && st.top() != '(') || // the closing bracket doesn't match the corresponding opening bracket at the top of the stack

(c == '}' && st.top() != '{') ||

(c == ']' && st.top() != '[')) {

return false; // the string is not valid, so return false

}

st.pop(); // otherwise, pop the opening bracket from the stack

}

}

return st.empty(); // if the stack is empty, all opening brackets have been matched with their corresponding closing brackets,

// so the string is valid, otherwise, there are unmatched opening brackets, so return false

}

};