**- Задача проверки корректности IP адреса узла.**

#include <stdio.h>

// Функция копирования строки

void strcpy\_custom(char \*dest, const char \*src) {

while (\*src != '\0') {

\*dest = \*src;

dest++;

src++;

}

\*dest = '\0';

}

// Функция для поиска символа в строке

int strchr\_custom(const char \*str, char ch) {

while (\*str != '\0') {

if (\*str == ch) {

return 1;

}

str++;

}

return 0;

}

// Функция для проверки корректности IP-адреса

int isValidIpAddress(const char \*ipAddress) {

int num;

char \*token, \*rest;

char ipCopy[100]; // Предполагаем, что длина строки не превышает 100 символов

strcpy\_custom(ipCopy, ipAddress);

token = ipCopy;

int count = 0;

while (\*token != '\0') {

int numDigits = 0;

int currentNum = 0;

while (\*token != '.' && \*token != '\0') {

if (\*token < '0' || \*token > '9') {

return 0; // Недопустимый символ

}

numDigits++;

currentNum = currentNum \* 10 + (\*token - '0');

token++;

}

if (numDigits == 0 || numDigits > 3 || currentNum > 255) {

return 0; // Некорректный формат числа

}

count++;

if (\*token == '\0' && count != 4) {

return 0; // Недостаточное количество чисел

}

if (\*token == '.') {

token++;

}

}

return count == 4;

}

int main() {

const char \*ip1 = "192.168.0.1";

const char \*ip2 = "256.0.0.1";

const char \*ip3 = "192.168.0.256";

const char \*ip4 = "192.168.0";

if (isValidIpAddress(ip1)) {

printf("%s - корректный IP адрес\n", ip1);

} else {

printf("%s - некорректный IP адрес\n", ip1);

}

if (isValidIpAddress(ip2)) {

printf("%s - корректный IP адрес\n", ip2);

} else {

printf("%s - некорректный IP адрес\n", ip2);

}

if (isValidIpAddress(ip3)) {

printf("%s - корректный IP адрес\n", ip3);

} else {

printf("%s - некорректный IP адрес\n", ip3);

}

if (isValidIpAddress(ip4)) {

printf("%s - корректный IP адрес\n", ip4);

} else {

printf("%s - некорректный IP адрес\n", ip4);

}

return 0;

}

**- Определение префикса строки, состоящего только из допустимых символов.**

#include <stdio.h>

int findValidPrefix(const char \*str) {

int i = 0;

// Перебираем символы строки

while (str[i] != '\0') {

// Если текущий символ недопустимый, возвращаем позицию префикса

if (!((str[i] >= 'a' && str[i] <= 'z') || (str[i] >= 'A' && str[i] <= 'Z'))) {

return i;

}

i++;

}

// Если вся строка состоит из допустимых символов, возвращаем длину строки

return i;

}

int main() {

const char \*str1 = "hello123";

const char \*str2 = "123world";

const char \*str3 = "abcDEF";

const char \*str4 = "!@#$%^";

printf("Префикс строки \"%s\" с допустимыми символами: %d\n", str1, findValidPrefix(str1));

printf("Префикс строки \"%s\" с допустимыми символами: %d\n", str2, findValidPrefix(str2));

printf("Префикс строки \"%s\" с допустимыми символами: %d\n", str3, findValidPrefix(str3));

printf("Префикс строки \"%s\" с допустимыми символами: %d\n", str4, findValidPrefix(str4));

return 0;

}  
**- Определение префикса строки, не содержащего недопустимых символов.**

#include <stdio.h>

int findValidPrefix(const char \*str) {

int i = 0;

// Перебираем символы строки

while (str[i] != '\0') {

// Если текущий символ недопустимый, возвращаем позицию префикса

if (!((str[i] >= 'a' && str[i] <= 'z') || (str[i] >= 'A' && str[i] <= 'Z'))) {

return i;

}

i++;

}

// Если вся строка состоит из допустимых символов, возвращаем длину строки

return i;

}

int main() {

const char \*str1 = "hello123";

const char \*str2 = "123world";

const char \*str3 = "abcDEF";

const char \*str4 = "!@#$%^";

printf("Префикс строки \"%s\" с допустимыми символами: %d\n", str1, findValidPrefix(str1));

printf("Префикс строки \"%s\" с допустимыми символами: %d\n", str2, findValidPrefix(str2));

printf("Префикс строки \"%s\" с допустимыми символами: %d\n", str3, findValidPrefix(str3));

printf("Префикс строки \"%s\" с допустимыми символами: %d\n", str4, findValidPrefix(str4));

return 0;

}

**- Вставка/удаление фрагмента строки.**

#include <stdio.h>

#define MAX\_LENGTH 1000 // Максимальная длина строки

// Функция для вставки фрагмента строки

void insertSubstring(char \*str, int pos, const char \*substr) {

int len = 0, subLen = 0;

while (str[len] != '\0') len++;

while (substr[subLen] != '\0') subLen++;

if (pos < 0 || pos > len) {

printf("Ошибка: недопустимая позиция для вставки\n");

return;

}

if (len + subLen >= MAX\_LENGTH) {

printf("Ошибка: превышена максимальная длина строки\n");

return;

}

// Сдвигаем символы с позиции pos до конца строки на subLen позиций вправо,

// чтобы сделать место для вставки подстроки

for (int i = len + subLen; i >= pos + subLen; i--)

str[i] = str[i - subLen];

// Копируем подстроку в освободившееся место

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

str[pos + i] = substr[i];

}

// Функция для удаления фрагмента строки

void deleteSubstring(char \*str, int pos, int len) {

int strLen = 0;

while (str[strLen] != '\0') strLen++;

if (pos < 0 || pos >= strLen || len <= 0 || pos + len > strLen) {

printf("Ошибка: недопустимые параметры для удаления подстроки\n");

return;

}

// Сдвигаем символы после удаляемого фрагмента влево на len позиций

for (int i = pos; i <= strLen - len; i++)

str[i] = str[i + len];

str[strLen - len] = '\0'; // обрезаем строку после удаления

}

int main() {

char str[MAX\_LENGTH] = "Hello, world!";

printf("Исходная строка: %s\n", str);

insertSubstring(str, 7, "beautiful ");

printf("После вставки: %s\n", str);

deleteSubstring(str, 0, 6);

printf("После удаления: %s\n", str);

return 0;

}

**- Поиск подстроки в строке.**

#include <stdio.h>

// Функция для определения длины строки

int stringLength(const char \*str) {

int len = 0;

while (str[len] != '\0') len++;

return len;

}

// Функция для сравнения двух строк

int stringCompare(const char \*str1, const char \*str2) {

int i = 0;

while (str1[i] != '\0' && str2[i] != '\0' && str1[i] == str2[i]) i++;

return str1[i] - str2[i];

}

// Функция для поиска подстроки в строке

int findSubstring(const char \*str, const char \*substr) {

int len1 = stringLength(str);

int len2 = stringLength(substr);

// Перебираем все возможные начальные позиции подстроки в строке

for (int i = 0; i <= len1 - len2; i++) {

int j;

// Проверяем, совпадают ли символы начиная с текущей позиции

for (j = 0; j < len2; j++) {

if (str[i + j] != substr[j])

break;

}

// Если все символы совпали, возвращаем позицию начала подстроки в строке

if (j == len2)

return i;

}

// Если подстрока не найдена, возвращаем -1

return -1;

}

int main() {

const char \*str = "Hello, world! This is a test string.";

const char \*substr1 = "world";

const char \*substr2 = "test";

int pos1 = findSubstring(str, substr1);

int pos2 = findSubstring(str, substr2);

if (pos1 != -1) {

printf("Подстрока \"%s\" найдена в строке на позиции %d\n", substr1, pos1);

} else {

printf("Подстрока \"%s\" не найдена в строке\n", substr1);

}

if (pos2 != -1) {

printf("Подстрока \"%s\" найдена в строке на позиции %d\n", substr2, pos2);

} else {

printf("Подстрока \"%s\" не найдена в строке\n", substr2);

}

return 0;

}

**- Перевод числа в строковое представление**

#include <stdio.h>

// Функция для определения длины строки

int stringLength(const char \*str) {

int len = 0;

while (str[len] != '\0') len++;

return len;

}

// Функция для обратного перевода числа в строковое представление

void reverseString(char \*str) {

int len = stringLength(str);

for (int i = 0; i < len / 2; i++) {

char temp = str[i];

str[i] = str[len - i - 1];

str[len - i - 1] = temp;

}

}

// Функция для перевода числа в строковое представление

void intToString(int num, char \*str) {

int i = 0;

// Обрабатываем случай, когда число равно 0

if (num == 0) {

str[i++] = '0';

} else {

// Записываем каждую цифру числа в строку справа налево

while (num != 0) {

int digit = num % 10;

str[i++] = '0' + digit;

num /= 10;

}

}

str[i] = '\0'; // Добавляем завершающий нулевой символ

reverseString(str); // Обратный порядок цифр

}

int main() {

int number = 12345;

char str[20]; // Здесь предполагается, что строка не будет длиннее 20 символов

intToString(number, str);

printf("Число в строковом представлении: %s\n", str);

return 0;

} **- Перевод символов строки в число.**

#include <stdio.h>

int stringToInt(const char \*str) {

int num = 0;

int sign = 1; // Переменная для хранения знака числа

// Пропускаем начальные пробелы

while (\*str == ' ')

str++;

// Обрабатываем знак числа

if (\*str == '-' || \*str == '+') {

if (\*str == '-')

sign = -1;

str++;

}

// Преобразуем символы строки в число

while (\*str >= '0' && \*str <= '9') {

num = num \* 10 + (\*str - '0');

str++;

}

return num \* sign;

}

int main() {

const char \*str = "-12345";

int num = stringToInt(str);

printf("Строка \"%s\" преобразована в число: %d\n", str, num);

return 0;

}

**- Конкатенация строк.**

#include <stdio.h>

#define MAX\_LENGTH 1000 // Максимальная длина строки

// Функция для определения длины строки

int stringLength(const char \*str) {

int len = 0;

while (str[len] != '\0') len++;

return len;

}

// Функция для конкатенации строк

void concatenateStrings(const char \*str1, const char \*str2, char \*result) {

int len1 = stringLength(str1);

int len2 = stringLength(str2);

int i = 0;

// Копируем первую строку в результат

while (i < len1) {

result[i] = str1[i];

i++;

}

// Добавляем вторую строку в конец результата

for (int j = 0; j < len2; j++) {

result[i + j] = str2[j];

}

result[i + len2] = '\0'; // Добавляем завершающий нулевой символ

}

int main() {

const char \*str1 = "Hello, ";

const char \*str2 = "world!";

char result[MAX\_LENGTH];

concatenateStrings(str1, str2, result);

printf("Результат конкатенации: %s\n", result);

return 0;

}

**- Проверка строк на совпадение с учетом регистра**

#include <stdio.h>

// Функция для проверки совпадения строк с учетом регистра

int areEqual(const char \*str1, const char \*str2) {

int i = 0;

// Перебираем символы обеих строк

while (str1[i] != '\0' && str2[i] != '\0') {

// Если текущие символы не совпадают, возвращаем разницу

if (str1[i] != str2[i]) {

return str1[i] - str2[i];

}

i++;

}

// Если одна из строк закончилась, проверяем их длины

if (str1[i] != '\0') {

return 1;

} else if (str2[i] != '\0') {

return -1;

}

// Если все символы совпадают, возвращаем 0

return 0;

}

int main() {

const char \*str1 = "Hello, world!";

const char \*str2 = "Hello, World!";

const char \*str3 = "Hello, world!";

if (areEqual(str1, str2) == 0) {

printf("Строки \"%s\" и \"%s\" совпадают.\n", str1, str2);

} else {

printf("Строки \"%s\" и \"%s\" не совпадают.\n", str1, str2);

}

if (areEqual(str1, str3) == 0) {

printf("Строки \"%s\" и \"%s\" совпадают.\n", str1, str3);

} else {

printf("Строки \"%s\" и \"%s\" не совпадают.\n", str1, str3);

}

return 0;

}

**- Проверка строк на совпадение без учета регистра.**

#include <stdio.h>

// Функция для приведения символа к нижнему регистру

char toLower(char ch) {

if (ch >= 'A' && ch <= 'Z') {

return ch + ('a' - 'A');

} else {

return ch;

}

}

// Функция для проверки совпадения строк без учета регистра

int areEqualIgnoreCase(const char \*str1, const char \*str2) {

while (\*str1 && \*str2) {

// Приводим символы к нижнему регистру и сравниваем их

if (toLower(\*str1) != toLower(\*str2)) {

return toLower(\*str1) - toLower(\*str2);

}

str1++;

str2++;

}

// Проверяем, что обе строки окончились одновременно

if (\*str1 == '\0' && \*str2 == '\0') {

return 0; // Строки совпадают без учета регистра

} else if (\*str1 != '\0') {

return 1; // str1 длиннее str2

} else {

return -1; // str1 короче str2

}

}

int main() {

const char \*str1 = "Hello, world!";

const char \*str2 = "hello, World!";

const char \*str3 = "HELLO, WORLD!";

if (areEqualIgnoreCase(str1, str2) == 0) {

printf("Строки \"%s\" и \"%s\" совпадают без учета регистра.\n", str1, str2);

} else {

printf("Строки \"%s\" и \"%s\" не совпадают без учета регистра.\n", str1, str2);

}

if (areEqualIgnoreCase(str1, str3) == 0) {

printf("Строки \"%s\" и \"%s\" совпадают без учета регистра.\n", str1, str3);

} else {

printf("Строки \"%s\" и \"%s\" не совпадают без учета регистра.\n", str1, str3);

}

return 0;

}

**- Сравнение строк по алфавиту.**

#include <stdio.h>

// Функция для сравнения строк по алфавиту

int compareStrings(const char \*str1, const char \*str2) {

while (\*str1 && \*str2) {

// Если текущие символы не совпадают, возвращаем разницу

if (\*str1 != \*str2) {

return \*str1 - \*str2;

}

str1++;

str2++;

}

// Если одна из строк закончилась, проверяем их длины

if (\*str1 != '\0') {

return 1;

} else if (\*str2 != '\0') {

return -1;

}

// Если все символы совпадают, возвращаем 0

return 0;

}

int main() {

const char \*str1 = "apple";

const char \*str2 = "banana";

const char \*str3 = "banana";

int result1 = compareStrings(str1, str2);

if (result1 < 0) {

printf("Строка \"%s\" идет раньше строки \"%s\" по алфавиту.\n", str1, str2);

} else if (result1 > 0) {

printf("Строка \"%s\" идет после строки \"%s\" по алфавиту.\n", str1, str2);

} else {

printf("Строки \"%s\" и \"%s\" одинаковы по алфавиту.\n", str1, str2);

}

int result2 = compareStrings(str2, str3);

if (result2 < 0) {

printf("Строка \"%s\" идет раньше строки \"%s\" по алфавиту.\n", str2, str3);

} else if (result2 > 0) {

printf("Строка \"%s\" идет после строки \"%s\" по алфавиту.\n", str2, str3);

} else {

printf("Строки \"%s\" и \"%s\" одинаковы по алфавиту.\n", str2, str3);

}

return 0;

}

**- Копирование строки.**

#include <stdio.h>

// Функция для копирования строки

void copyString(const char \*source, char \*destination) {

while (\*source) {

\*destination = \*source;

source++;

destination++;

}

\*destination = '\0'; // Добавляем завершающий нуль-символ

}

int main() {

const char \*source = "Hello, world!";

char destination[100]; // Предполагаем, что длина строки не превышает 100 символов

copyString(source, destination);

printf("Скопированная строка: %s\n", destination);

return 0;

}

**- Преобразование CamelCase в snake\_case**

#include <stdio.h>

// Функция для преобразования CamelCase в snake\_case

void camelToSnake(const char \*camel, char \*snake) {

int snakeIndex = 0;

for (int i = 0; camel[i] != '\0'; i++) {

if (camel[i] >= 'A' && camel[i] <= 'Z') {

// Если символ верхнего регистра, добавляем перед ним символ '\_'

if (i > 0 && camel[i - 1] != '\_') {

snake[snakeIndex++] = '\_';

}

snake[snakeIndex++] = camel[i] + ('a' - 'A'); // Приводим к нижнему регистру

} else {

snake[snakeIndex++] = camel[i];

}

}

// Добавляем завершающий нулевой символ

snake[snakeIndex] = '\0';

}

int main() {

const char \*camelCase = "helloWorldExample";

char snakeCase[100]; // Предполагаем, что строка не будет превышать 100 символов

camelToSnake(camelCase, snakeCase);

printf("Преобразование CamelCase \"%s\" в snake\_case: %s\n", camelCase, snakeCase);

return 0;

}

**- Преобразование snake\_case в CamelCase**

#include <stdio.h>

// Функция для преобразования snake\_case в CamelCase

void snakeToCamel(const char \*snake, char \*camel) {

int camelIndex = 0;

int i = 0;

while (snake[i] != '\0') {

if (snake[i] == '\_') {

i++;

if (snake[i] != '\0') {

camel[camelIndex++] = snake[i] - 32;

i++;

}

} else {

camel[camelIndex++] = snake[i];

i++;

}

}

// Добавляем завершающий нулевой символ

camel[camelIndex] = '\0';

}

int main() {

const char \*snakeCase = "hello\_world\_example";

char camelCase[100]; // Предполагаем, что строка не будет превышать 100 символов

snakeToCamel(snakeCase, camelCase);

printf("Преобразование snake\_case \"%s\" в CamelCase: %s\n", snakeCase, camelCase);

return 0;

}

**- Подсчет суммы четных/нечетных чисел**

#include <stdio.h>

// Функция для подсчета суммы четных или нечетных чисел в заданном диапазоне

int sumEvenOrOdd(int start, int end, int isEven) {

int sum = 0;

// Перебираем числа в заданном диапазоне

for (int i = start; i <= end; i++) {

// Проверяем, является ли число четным или нечетным в зависимости от значения isEven

if ((i % 2 == 0 && isEven) || (i % 2 != 0 && !isEven)) {

sum += i; // Добавляем число к сумме

}

}

return sum;

}

int main() {

int start = 1;

int end = 10;

// Подсчет суммы четных чисел от start до end

int sumEven = sumEvenOrOdd(start, end, 1);

printf("Сумма четных чисел от %d до %d: %d\n", start, end, sumEven);

// Подсчет суммы нечетных чисел от start до end

int sumOdd = sumEvenOrOdd(start, end, 0);

printf("Сумма нечетных чисел от %d до %d: %d\n", start, end, sumOdd);

return 0;

}

**- Подсчет количества вхождений в строку символов из набора**

#include <stdio.h>

// Функция для подсчета количества вхождений символов из набора в строку

int countOccurrences(const char \*str, const char \*chars) {

int count = 0;

// Проходим по каждому символу в строке

for (int i = 0; str[i] != '\0'; i++) {

// Проходим по каждому символу в наборе

for (int j = 0; chars[j] != '\0'; j++) {

// Если текущий символ в строке совпадает с текущим символом в наборе, увеличиваем счетчик

if (str[i] == chars[j]) {

count++;

break; // Переходим к следующему символу в строке

}

}

}

return count;

}

int main() {

const char \*str = "Hello, world!";

const char \*chars = "aeiou";

int occurrences = countOccurrences(str, chars);

printf("Количество вхождений символов из набора \"%s\" в строку \"%s\": %d\n", chars, str, occurrences);

return 0;

}