



**Росдистант**  
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

# ОСНОВЫ

ПРОГРАММИРОВАНИЯ 7 ЧАСТЬ

## МАССИВЫ СИМВОЛОВ. СТРОКИ

1. <code>char stroka[7];</code> <code>stroka="Москва";</code>	2. <code>char stroka[6]={ 'M', 'o', 'c', 'к',</code> <code>'в', 'а'};</code>
В памяти эта строка будет выглядеть: Москва\0	В памяти эта строка будет выглядеть: Москва



### Слайд 171

#### Тема 7. Массивы символов. Строки

«Символ – элементарная единица, некоторый набор, которых несет определенный смысл. В языке программирования **C++** предусмотрено использование символьных литерал. Символьный литерал - это целочисленное значение, типа **int** , представленное в виде символа, заключённого в одинарные кавычки. В таблице **ASCII** представлены символы и их целочисленные значения.»

На слайде приведены два примера, два способа описания и определения строки символов.

Работа со строками реализуется путем использования одномерных массивов символов типа **char** .

Рассмотрим первый пример. На языке **C++** допускаются строковые константы – это строка символов, заключенная в двойные кавычки.

Любая последовательность символов, заключенная в двойные кавычки «», рассматривается как строковая константа. Символьная строка – это одномерный массив, заканчивающийся нулевым байтом. Для нулевого байта определена специальная символьная константа – **\0** , целочисленное значение которого равно 0. Если строка должна содержать **n** символов, то при описании массива необходимо задать размер массива равный **n+1** . В конце строковой константы

автоматически ставится символьная константа `\0` . Так, последовательность символов, представляющая собой строковую константу, будет размещена в оперативной памяти компьютера, включая нулевой байт. Строковые константы размещаются в статической памяти. Начальный адрес последовательности символов в двойных кавычках трактуется как адрес строки. Строковые константы часто используются для осуществления диалога с пользователем.

Рассмотрим второй пример. Строка символов представляет собой одномерный массив символов типа `char` . Размер массива определяется количеством символов в массиве. При разной организации символьного массива и подходы обработки массивов будут разными.

## ФУНКЦИИ ОБРАБОТКИ СТРОК

<conio.h>	
getch()	ожидается ввод любого символа
<stdio.h>	
gets(s);	функция ввода строки символов
puts(s);	функция вывода строки символов до нулевого символа '\0'



### Слайд 172

На языке **C++** существуют функции, позволяющие обрабатывать строку символов. Рассмотрим эти функции и примеры их использования.

В таблице указаны заголовочные файлы, которые необходимо подключать к программе, для использования функций обработки строк символов. Название образовано от имени строчного типа данных **string**, от английского слова строка. В языке **C++** и его предшественнике, языке программирования **C**, нет встроенной поддержки строкового типа данных, вместо этого используется массив символов. Являясь частью стандартной библиотеки **C++**, эти объекты также являются частью стандартного пространства имён – **std**.

Функция **getch** - ожидается ввод любого символа. Пустые скобки рядом с именем этой функции необходимы для соблюдения синтаксиса языка. Эта функция используется, как правило, в конце программы, чтобы успеть проанализировать результаты перед возвратом к программному коду.

При использовании этой функции необходимо подключить к программе заголовочный файл **conio**.

Функция **gets** используется для ввода любой цепочки символов. Автоматически в конце строки ставится нулевой байт.

Функция **puts** используется для вывода строки символов до нулевого символа, то есть символа - **\0**.

В качестве параметра этих функций ставится имя массива символов.

При использовании этих функций необходимо подключить к программе заголовочный файл **stdio** .

## ФУНКЦИЯ КОПИРОВАНИЯ

<string.h>	
Структура функции: char *strcpy ( char *s, char *s1);	
strcpy( s,s1 );	Пример: char s[ ]= "kkkk"; cout<<"\n"<<strcpy(s, "abcd")<<"\t"<<s; Результат: abcd      abcd

### Слайд 173

«Есть много случаев, когда мы не знаем заранее, насколько длинной будет наша строка. Например, рассмотрим проблему написания программы, где мы просим пользователя ввести свое имя. Насколько длинным оно будет? Это неизвестно до тех пор, пока пользователь его не введет!

В таком случае мы можем объявить массив размером больше, чем нам нужно:

Тем не менее, использование функции **strcpy** может легко вызвать переполнение массива, если не быть осторожным!»

Рассмотрим функцию копирования одной строки в другую.

Для использования этой функции необходимо подключить к программе заголовочный файл **string** .

Функция копирует строку **s1** в строку **s**.

Во второй строке таблицы показана структура этой функции в общем виде, которая разъясняет работу этой функции.

Из описания структуры видно - результатом работы этой функции является указатель. В скобках указывается два параметра:

- строка **s** , в которую копируются символы;
- строка **s1** , которая копируется.

Строка **s** – должна быть достаточно большой, чтобы в нее поместилась строка **s1**. Если места мало, компилятор не выдаст ошибки, но может привести к порче других данных.

На слайде показан пример использования этой функции на конкретном примере. Обратите внимание, как происходит вызов этой функции – в структуре функции вывода на печать.

## ФУНКЦИЯ КОПИРОВАНИЯ

<string.h>	
Структура функции:	
char *strncpy ( char *s, char *s1, int n);	
strncpy( s,s1 );	char s[] = "abcdf"; 1. cout<< strncpy ( s, "kkkk ", 4); Результат: kkkk 2. cout<< strncpy ( s, "kkkk", 2); Результат: kkcdf



### Слайд 174

«Большинство операций языка **Си**, имеющих дело со строками, работает с указателями. Для размещения в оперативной памяти строки символов необходимо:

выделить блок оперативной памяти под массив;  
проинициализировать строку.

Для выделения памяти под хранение строки могут использоваться функции динамического выделения памяти. При этом необходимо учитывать требуемый размер строки. Под хранение строки выделяются последовательно идущие ячейки оперативной памяти. Таким образом, строка представляет собой массив символов. Для хранения кода каждого символа строки отводится один байт.»

Результатом работы многих функции обработки строк, массивов символов, является указатель. Следующая функция **strncpy** которая копирует, но только часть строки **s1** в строку **s**. Функция также возвращает указатель на начало строки **s**.

У данной функции три параметра:

- строка **s**, в которую копируются символы;
- строка **s1**, из которой копируется;



- **n** - количество копируемых символов.

В первом примере показано: если копируется вся строка **s1** , то хвост строки **s** отбрасывается, строка **s** как говорится затирается.

Во втором примере показано: если копируется только часть строки **s1** , то заменяются первые символы строки **s** .

В результате работы этой функции строка **s** изменяется, а строка **s1** - остается неизменной.

На слайде приведен пример, демонстрирующий работу этой функции. Массивы символов инициализированы при объявлении этих массивов.

## ФУНКЦИЯ КОНКОТЕНАЦИИ

<string.h>	
Структура функции:	
char *strcat(char *s, char *s1 );	
strcat( s, s1 );	char s[] = "abcd", s1[] = "kk"; cout<< strcat ( s, s1 )<< "\t"<<s; Результат: abcdkk    abcdkk

### Слайд 175

Функция конкатенации, или как говорят - присоединения, объединяет две строки, присоединяет к строке **s** строку **s1**.

Внимательно рассмотрим структуру этой функции. Перед именем функции **strcat** стоит символ звездочка. Это означает, что функция возвращает результат в виде указателя. Тип функции **char** свидетельствует о том, что результатом работы этой функции будет указатель на строку символов.

В списке параметров этой функции в скобках указывается два параметра:

строка **s**, к которой присоединяется;

строка **s1**, которая присоединяется.

В результате работы этой функции строка **s** изменяется, а строка **s1** - остается неизменной.

Рассмотрим пример на слайде. Так как при определении массива значения элементов массива определены, инициализированы, количество символов в массиве можно не указывать, оставив пустые квадратные скобки. Обращение к функции происходит в структуре функции вывода на печать. Если функция возвращает результат, то вызов функции допустим только в структуре какого-либо оператора. Функция возвращает указатель на начало строки **s**, Возврат из функции инициирует вывод на печать строки **s**,

Выведя на печать строку **s** , можно убедиться, что она изменилась.

Для использования этой функции необходимо подключить к программе заголовочный файл **string** .

## ФУНКЦИИ КОНКОТЕНАЦИИ

<string.h>	
Структура функции:	
char *strncat (char *s, char *s1, int n);	
strncat(s,s1,n);	char s[] = "abcd", s1[] = "kfrsv"; cout<< strncat (s, s1, 2)<< "\t"<<s; Результат: abcdkf   abcdkf

### Слайд 176

Рассмотрим еще одну функцию конкатенации. В отличие от функции, представленной на предыдущем слайде, функция **strncat**, присоединяет только часть строки **s1** к строке **s**.

Из описания структуры видно, что результатом работы этой функции является указатель. При обращении к этой функции указывается три параметра:

строка **s**, к которой присоединяется;

строка **s1**, которая присоединяется;

**n** - количество символов, которые необходимо присоединить.

Пример на слайде демонстрирует работу этой функции. Обращение к функции происходит в операторе вывода на печать. Возвращается результат в виде указателя на начало строки **s**. Выведя на печать результат работы этой функции, можно убедиться, что только часть строки **s1** присоединилась к строке **s**. Строка **s1** остается неизменной. Можно присоединить всю строку, но при этом лучше воспользоваться функцией **strcat**, описанной на предыдущем слайде.

Внимательно проанализируйте приведенный на слайде пример, чтобы разобраться в особенностях применения этой функции.

Для использования этой функции необходимо подключить к программе заголовочный файл **string**, которая указана в первой строке таблицы.

## ФУНКЦИЯ ПОИСКА В СТРОКЕ СИМВОЛОВ

<string.h>	
Структура функции:	
int strspn( char *s, char *s1);	
strspn( s, s1 );	cout<< strspn("abcdfkkk", "abk" ); Результат: 2

### Слайд 177

Представленная на слайде функция определяет длину первого сегмента в строке **s** , содержащего только символы из множества **s1** .

Из описания структуры понятно, что тип функции описан как **int** - целый. То есть результатом работы этой функции, возвращаемый функцией результат, является целое число. В скобках указывается два параметра, два символьных массива, две строки.

В приведенном примере видно, что функция возвращает целое число. Обращение к функции происходит в структуре функции вывода на печать. В нашем примере результат - число два, которое равно количеству подряд идущих символов строки **s1** , найденных в строке **s** . Несмотря на то, что некоторые символы далее в строке **s** встречаются, например, символ **k**, в рассмотрение не попадают. Посимвольно сравниваются две строки, и как только произошло не сравнение кодов строк, функция возвращает результат.

В результате работы этой функции строка **s** и строка **s1** - остаются неизменными.

Если символы строки **s1** в строке **s** не найдены, результатом работы этой функции будет число ноль.

Для использования этой функции необходимо подключить к программе заголовочный файл **string**.

## ФУНКЦИЯ ПОИСКА В СТРОКЕ СИМВОЛОВ

<string.h>	
Структура функции:	
int strcspn ( char *s, char *s1);	
strcspn( s, s1 );	int t= strcspn ("abcdf ", "dk"); cout<< t; Результат: 3

### Слайд 178

Функция, представленная на слайде, выполняет действия, противоположные действиям предыдущей функции. Функция определяет длину первого сегмента строки **s** , содержащего символы, не входящие в строку **s1** .

Из описания структуры видно - результатом работы этой функции является целое число, тип функции описан как **int** - целый. В скобках указывается два формальных параметра, два указателя типа **char** .

Обращение к функции происходит о операторе присваивания. Описана целая переменная **t** и инициализацией значения этой переменной и есть обращение к функции. В виде фактических параметров при обращении к функции указаны две строки.

В приведенном примере видно, что функция возвращает целое число, которое равно количеству символов строки **s** , не содержащих символов строки **s1** . Каждый символ строки **s1** ищется в строке **s** . Как только один из символов найден, функция возвращает результат. В приведенном на слайде примере это число три.

Обратите внимание, в результате работы этой функции строка **s** и строка **s1** - остаются неизменными.

Если символы строки **s1** в строке **s** не найдены, результатом работы этой функции будет длина строки **s** без учета признака конца строки, символа **\0** .

Для использования этой функции необходимо подключить к программе заголовочный файл **string**.

## ПРИМЕР ФУНКЦИИ ПОИСКА В СТРОКЕ СИМВОЛОВ

<string.h>	
Структура функции:	
char *strstr (const char *s, const char *s1);	
strstr( s, s1 );	cout<<strstr ("abcABcabc", "ABC");
Результат: ABcabc	

### Слайд 179

Функция **strstr** ищет в строке **s** подстроку **s1** и возвращает указатель на первое вхождение строки **s1** в строку **s**.

Из описания структуры видно - результатом работы этой функции является указатель на найденную строку. В скобках указывается два параметра, две строки.

- строка **s** , в которой ищется подстрока **s1**;
- строка **s1** , которая ищется.

В приведенном примере видно, что функция возвращает указатель на тот элемент в строке **s** , на тот символ, с которого начинается строка **s1**.

В результате работы этой функции строка **s** и строка **s1**- остаются неизменными. Используя эту функцию совместно с **cout**, на печать выводится часть строки **s** с положения указателя и до конца этой строки.

При решении различных задач, можно использовать эту функцию поиска, оперируя указателем. Если совпадений не обнаружено, возвращается NULL.

Если необходимо продолжить поиск символов в исходной строке, достаточно сдвинуть внутренний указатель на следующий символ в строке и продолжить поиск. Этот принцип поиска мы рассмотрим далее на примере задач.

Для использования этой функции необходимо подключить к программе



заголовочный файл **string** с помощью препроцессорной команды.

## ФУНКЦИЯ ПОИСКА В СТРОКЕ СИМВОЛОВ. ПРИМЕР И РЕЗУЛЬТАТ

<string.h>	
Структура функции: char *strchr ( char *s, int k );	
strchr ( s, k );	cout<<strchr ( "abcAabc", 65 ); Результат: Aabc 65 – код 'A'.

### Слайд 180

Функция **strchr** ищет в строке - символ с кодом **k** и возвращает указатель на элемент в строке **s** , с кодом **k** .

Существует международная таблицы **ASCII** кодов. Таблицы **ASCII** кодов это таблица кодировки, в которой некоторым распространённым печатным и непечатным символам сопоставлены числовые коды. Таблица была разработана и стандартизована в США, в 1963 году.

Таблица ASCII определяет коды для символов:

- десятичных цифр;
- символы латинского алфавита;
- символы национального алфавита;
- знаков препинания;
- управляющих символов.

Из описания структуры видно, что результатом работы этой функции является указатель. При вызове функции **strchr** указывают два параметра:

строка **s** , в которой ищется символ с кодом **k** ;

код искомого символа.

В приведенном примере видно, что функция возвращает указатель на первое

вхождение символа с кодом с указанным кодом в строке. Где, число 65 код заглавной буквы латинского алфавита А.

В результате работы этой функции исходная строка остается неизменной.

При решении различных задач, можно использовать эту функцию поиска, оперируя указателем.

Для использования этой функции необходимо подключить к программе заголовочный файл **string** .

## ЗАПОЛНЕНИЕ СТРОКИ СИМВОЛАМИ

<string.h>	
Структура функции: int strset ( char *s, int k);	
strset( s, k );	char s[] = "abcd"; cout<<strset ( s, 66 )<< "\t"<<s; Результат: BBBB    BBBB

### Слайд 181

Функция **strset** заполняет строку **s** - символами с кодом **k** . Функция возвращает указатель на начало строки **s** . Значение **k** указывается из таблицы **ASCII** кодов.

Из описания структуры видно - результатом работы этой функции является указатель на начало преобразованной строки символов.

Функции **strset** имеет два параметра:

- строка **s** , которая заполняется символами с кодом **k**;
- код символа.

В приведенном примере видно, что функция заполняет строку **s** заглавной буквой латинского алфавита **B**, и возвращает указатель на начало строки **s** . Где 66 - код заглавной буквы латинского алфавита **B** . При этом, длина строки **s** остается неизменной.

В приведенном примере строка символом инициализирована при объявлении массива. Размер массива можно не указывать при объявлении массива. Он определяется длиной цепочки символов при инициализации. Поэтому рядом с именем массива можно указаны пустые квадратные скобки.

Для использования этой функции необходимо подключить к программе заголовочный файл **string** .

## ЗАПОЛНЕНИЕ СТРОКИ СИМВОЛАМИ

<string.h>	
Структура функции:	
int strnset ( char *s, int k, int n);	
strnset( s, k, n );	cout<<strnset ("abcdf ", 65, 3 );
Результат: AAAdf	

### Слайд 182

Функция **strnset** заменяет первые **n** символов с строке **s** символами с кодом **k** . Функция возвращает указатель на начало строки **s** . Значение числа **k** указывается из международной таблицы **ASCII** кодов.

Из описания структуры видно - результатом работы этой функции является указатель.

Функции **strnset** имеет три параметра:

- строка **s** , которая заполняется символами с кодом **k** ;
- код символа;
- количество символов.

В описании структуры этой функции, первый формальный параметр определен как указатель символьного типа. А так как известно, что имя массива и есть указатель, при обращении к функции первым фактическим параметром является строка символов.

В приведенном примере видно, что функция заполняет три первых символа строки **s** заглавной буквой латинского алфавита **A**, и возвращает указатель на начало строки. Число 65 - код заглавной буквы **A** латинского алфавита. В результате работы этой функции длина строки **s** – остается неизменной.

Для использования этой функции необходимо подключить к программе

заголовочный файл **string** .

## ПРЕОБРАЗОВАНИЕ СТРОКИ СИМВОЛОВ

<string.h>	
Структура функции: char *strupr ( char *s );	
strupr ( s );	cout<<strupr ("abcd"); Результат: ABCD

### Слайд 183

Представленная на слайде функция преобразует буквы нижнего регистра в буквы верхнего регистра. Анализируются коды символов, согласно таблице **ASCII** кодов.

Из описания структуры функции видно, что результатом работы этой функции является преобразованная строка. Указатель устанавливается на начало преобразованной строки. В конечном итоге, функция возвращает указатель на начало преобразованной строки.

Функции имеет один параметр – строку символов, которую необходимо преобразовать.

В приведенном примере малые буквы латинского алфавита преобразованы в заглавные. Если в цепочке символов встретятся заглавные буквы, то они остаются неизменными.

Вызов этой функции реализован в структуре функции вывода на печать, соответственно преобразованная строка символов распечатается на экране.

Длина строки **s** – остается неизменной.

Для использования этой функции необходимо подключить к программе заголовочный файл **string**.

## ПРЕОБРАЗОВАНИЕ СТРОКИ СИМВОЛОВ

<string.h>	
Структура функции: char * strlwr ( char *s );	
strlwr (s);	cout<<strlwr ("ABC" ); Результат: abc

### Слайд 184

Представленная на слайде функция преобразует буквы верхнего регистра в буквы нижнего регистра.

Из описания структуры функции видно, что результатом работы этой функции является указатель на начало преобразованной строки.

Функции имеет один параметр – строку символов, которую необходимо преобразовать.

В приведенном примере заглавные буквы латинского алфавита преобразованы в малые.

Функция устанавливает внутренний указатель на начало преобразованной строки. Вызов этой функции реализован в структуре функции вывода, соответственно преобразованная строка символов распечатается на экране.

Длина строки **s** – остается неизменной.

Для использования этой функции необходимо подключить к программе заголовочный файл **string** .



## СРАВНЕНИЕ СТРОК СИМВОЛОВ

<string.h>	
Структура функции:	
int strcmp(char *s1, char *s2);	
strcmp(s1,s2);	s1=s2 – функция возвращает значение 0, s1 < s2 – положительное значение s1 > s2 – отрицательное значение

### Слайд 185

Представленную на слайде функцию мы уже применяли ранее при решении некоторых задач. Рассмотрим работу этой функции подробно.

Из описания структуры видно - результатом работы этой функции является целое число, тип функции описан как **int**. Функция посимвольно сравнивает коды двух строк, строки символов **s1** и строки **s2** и возвращает значение ноль, если коды совпали. В таблице **ASCII** кодов, коды заглавных и прописных букв отличаются. Если одно и тоже слово написано в одной строке заглавными, а в другой строке прописными, сравнение не пройдет.

Если сравнение строк не прошло, функция возвращает число, которое можно использовать в программе для дальнейшего анализа. На слайде приведены возможные значения, возвращаемые этой функцией.

Функция имеет два параметра, две строки, которые сравниваются. На слайде показано, какие значения может возвращать эта функция.

В результате работы этой функции строка **s1** и строка **s2** - остаются неизменными.

Ниже, мы рассмотрим примеры с использованием этой функции.

Для использования этой функции необходимо подключить к программе заголовочный файл **string**.

## ИНВЕРТИРОВАНИЕ СТРОКИ СИМВОЛОВ

<string.h>	
Структура функции: char *strrev ( char *s );	
strrev( s );	<pre>char s[100]; gets(s);           // ввод строки puts ( strrev ( s ) ); // вывод инвертированной строки  Если ввести: ABCD  Результат:  DCBA</pre>



### Слайд 186

На слайде представлена функция инвертирование строки символов.

Из описания структуры функции видно, что перед именем функции стоит символ звездочка. Следовательно, результатом работы этой функции является указатель, указатель на начало инверсированной строки символов.

Функция имеет один параметр – строку символов.

В приведенном примере описан массив символом на сто элементов, символов.

С помощью функции **gets** вводится строка символов, длиной не больше ста элементов. По завершению ввода символов, в конце строки автоматически добавится признак конца строки, символ **\0**.

На слайде приведен пример, где функция инвертирования указана как аргумент функции **puts**, вывода строки символов на экран. Вывод преобразованной строки **s** осуществляется до признака конца строки.

Для использования этой функции необходимо подключить к программе заголовочный файл **string**.

## ОПРЕДЕЛЕНИЕ ДЛИНЫ СТРОКИ СИМВОЛОВ

<string.h>	
Структура функции: int strlen(char *s);	
strlen( s );	char s []= "abcde"; cout<<strlen(s); Результат: 5

### Слайд 187

Функция **strlen** определяет длину строки символов. Без этой функции практически не обходится ни одна задача обработки строк символов. Можно использовать символьный массив, но намного удобнее работать со строкой символов, и обрабатывать всю строку целиком.

Из описания структуры видно - результатом работы этой функции является целое число, определяющее количество символов в строке **s**.

Функции имеет один параметр – строка символов. Возвращаемый функцией результат – целое число, длина строки символов.

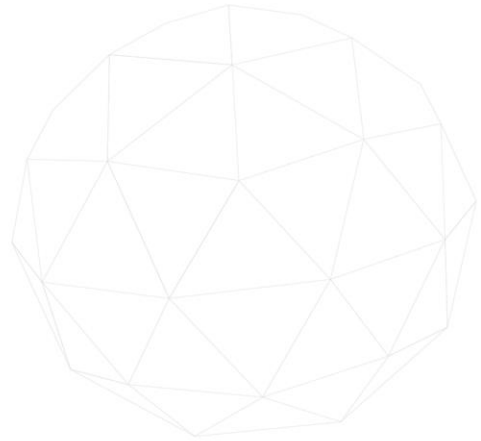
В приведенном примере определена строка символов, заданная в двойных кавычках. При такой инициализации массива в конце строки автоматически вставляется признак конца, символ **\0**. При подсчете длины строки этот символ не учитывается. Обращение к функции происходит в структуре функции вывода на печать.

Исходная строка символов остается неизменной.

Для использования этой функции необходимо подключить к программе заголовочный файл **string**. Имя файла указано в верхней строке таблицы.

## ПОИСК СИМВОЛА В ТЕКСТЕ

```
...  
char s[100];  
cout<<"\n Введите текст \n";  
gets( s );  
for (int i=0, k=0; i<strlen(s); i++)  
    if (s[ i ] == '?') k++;  
cout<<"\n k="<<k;  
...
```



### Слайд 188

Рассмотрим задачи, в которых рассмотрим практическое применение рассмотренных ранее функции обработки строк символов.

На данном слайде продемонстрирован фрагмент, позволяющий подсчитать, сколько раз в тексте встретился восклицательный знак.

Описан одномерный массив символов.

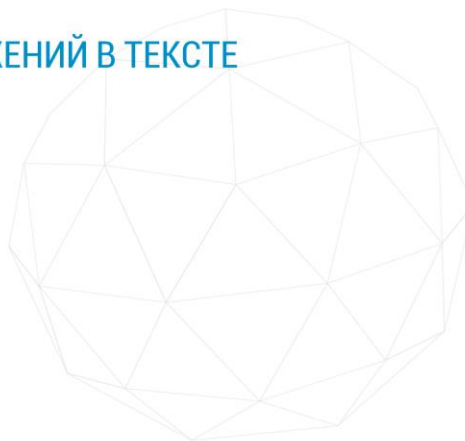
Для ввода строки символов используется функция ввода **gets**, ввод символов завершается нажатием клавиши **enter**. В конце строки автоматически вставляется символ **\0** – признак конца строки.

В цикле подсчитывается, сколько раз встретился искомый символ.

В структуре оператора цикла указано условие прекращения цикла – функция **strlen**. Эта функция подсчитывает длину строки без признака конца строки. В приведенном примере строка рассматривается как обычный одномерный массив и анализируется каждый элемент массива.

## ПОДСЧЕТ КОЛИЧЕСТВА ПРЕДЛОЖЕНИЙ В ТЕКСТЕ

```
#include<iostream>
#include<stdio.h>
#include<string.h>
using namespace std;
int main()
{ char s[100];
  cout<<"\n Введите текст \n"; gets( s);
  for (int i=0, k=0; i<strlen(s); i++)
    if (s[i]=='!' || s[i]=='.' || s[i]=='?') k++;
  cout<<"\n k="<<k;
  ...
}
```



### Слайд 189

Рассмотрим задачу: подсчитать количество предложений в тексте.

Для использования функций обработки строк символов необходимо подключить заголовочные файлы:

**iostream** – для вывода информации на экран;

**stdio** – для функции ввода строки символов **gets** ;

**string** – для функции определения длины строки символов **strlen** .

Предложение может заканчиваться одним из следующих знаков: точка, вопросительный знак, восклицательный знак. Исходя из этого, мы и будем решать задачу. Строка символов вводится с клавиатуры. Ввод завершается нажатием клавиши **Enter** . Введенный текст рассматривается как одномерный массив символов.

Подсчет предложений в тексте производится в цикле, количество циклом определяется длиной введенного текста. Длина текста вычисляется функцией **strlen** .

На слайде продемонстрирован фрагмент программы.

## ПОДСЧЕТ КОЛИЧЕСТВА СЛОВ В ТЕКСТЕ

Задача: подсчитать, сколько раз искомое слово встретилось в тексте

Обозначения	
word []	искомое слово
text []	исходный текст
s []	буфер слов



### Слайд 190

На слайде представлена формулировка задачи. Введены обозначения используемых переменных.

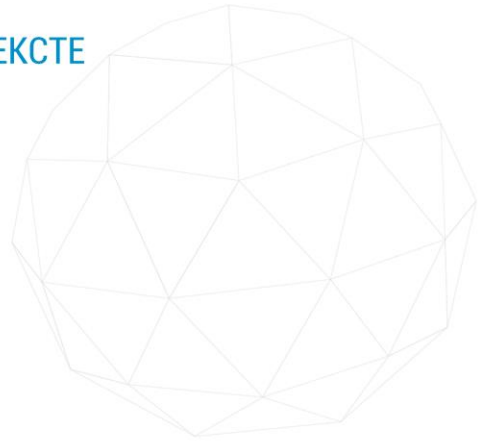
Из исходного текста будем выделять количество символов, равных длине искомого слова. Под искомым словом понимается любая цепочка символов. Выделенные из исходного текста символы попадают в переменную - буфер слов и сравниваются с искомым словом.

Таким образом, перемещаясь по тексту, сначала с первого символа, затем со второго символа и так далее, подсчитаем, сколько раз встретилось искомое слово. Для сравнения будем использовать функцию посимвольного сравнения выделенных из текста символов с искомым словом.

Последний раз необходимо выделить символы исходного текста так, чтобы точно уложиться в этот текст, не выходя за его границы.

## ПОДСЧЕТ КОЛИЧЕСТВА СЛОВ В ТЕКСТЕ

```
#include<iostream>
#include<stdio.h>
#include<conio.h>
#include<string.h>
using namespace std;
int main()
{ char text[100], word[20], s[20];
  gets(text);    // ввод текста
  gets(word);    // ввод слова
```



### Слайд 191

На слайде представлен первый фрагмент программы решения задачи.

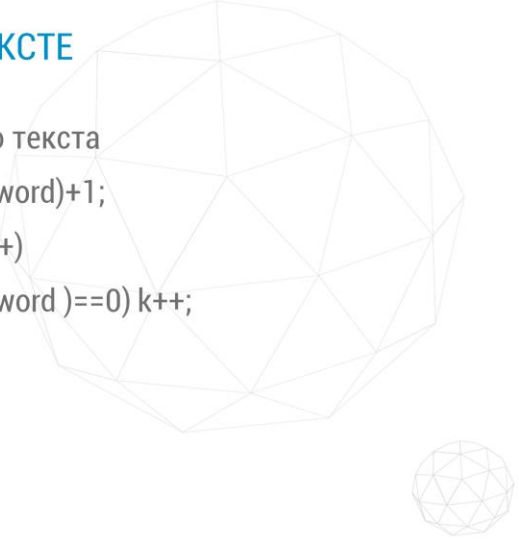
Подключены заголовочные файлы, необходимые для использования функций ввода-вывода строки символов, функции определения длины строки, функции сравнения строк.

Описаны массивы символов для исходного текста, искомого слова и буфера слов. Размер буфера равен длине искомого слова.

С клавиатуру вводится исходный текст и искомое слово. Используется функция ввода строки символов **gets**. В конце строки автоматически введется признак конца строки, константа **\0**. Это позволит в следующем фрагменте программы, использовать функции обработки строк.

## ПОДСЧЕТ КОЛИЧЕСТВА СЛОВ В ТЕКСТЕ

```
char *p=text; // указатель в начало текста
for (int i=0,k=0; i<strlen(text)-strlen(word)+1;
    i++, p++)
if (strcmp( strncpy(s,p,strlen(word) ),word )==0) k++;
    1
    2
cout<<"\n k="<<k;
getch(); }
```



### Слайд 192

Введенный с клавиатуры текст будем рассматривать как строку символов. Введена переменная-указатель **p**, которая устанавливается в начало исходного текста. Обратите внимание как описана и определена эта переменная. Указателю присваивается адрес начала строки символов.

Перемещаясь по тексту, от символа к символу, функция **strncpy** выделяет из искомого текста в буфер, переменную **s**, количество символов, равных длине искомого слова.

Функция **strcmp** сравнивает коды символов буфера с искомым словом. Если прошло сравнение, эта функция возвращает значение равное нулю. Переменная **k** подсчитывает сколько раз встретилось искомое слово в тексте.

На слайде введены обозначения:

1 – выделение символов из исходного текста в буфер **s**, начиная с позиции указателя **p**;

2 – сравнение выделенных символов с искомым словом.

Параметрами функции **strncpy** являются:

- **s** – буфер слов, строка в которую копируются выделенные символы;
- **p** – указатель, позиция, с которой из исходного текста выделяются символы;
- **strlen(word)** – количество выделяемых из исходного текста символов, равное



длине искомого слова.

Ранее мы рассматривали принцип работы этой функции.

Обратите внимание, после каждого цикла значение указателя **p** увеличивается на единицу. Переходим к следующему символу исходного текста и процесс поиска продолжается.

Нет необходимости очищать буфер слов, строку **s**, так как количество символов, которые копируются из исходного текста все время одинаковое, равное длине искомого слова. На каждом шаге новые символы как бы затирают старые символы.

## УДАЛЕНИЕ ЧАСТИ СТРОКИ

Задача: удалить из текста некоторое слово. Под словом понимают любую цепочку символов

Обозначения	
text []	Исходный текст
word []	Слово
newText []	Новый текст

### Слайд 193

Рассмотрим задачу удаления части текста.

Введены обозначения: исходный текст, часть текста, которую необходимо удалить, преобразованный текст. Часть текста, которую надо удалить будем называть просто словом.

Для решения этой задачи, поиска слова в исходном тексте будем использовать функцию поиска **strstr**. Ранее мы рассматривали функцию поиска. Результатом работы этой функции является указатель. Начиная с этого указателя и будем удалять часть исходного текста.

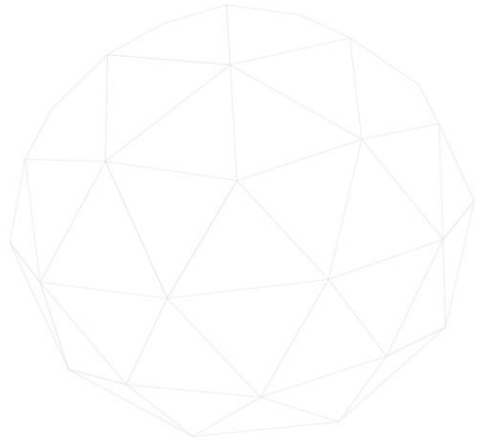
Функция конкатенации, присоединения, сформирует новый текст из исходного. К части текста до удаляемого слова добавляется часть текста после слова.

Управляя указателем, как результатом работы этих функций, сформируем новый текст.

На следующих слайдах представлена программа решение этой задачи.

## УДАЛЕНИЕ ЧАСТИ СТРОКИ

```
# include <iostream>
# include <stdio.h>
# include <string.h>
# include <conio.h>
using namespace std;
int main()
{ char text[100], word[25];
  char newText[ ]="\0"; // новый текст
  gets( text );          // исходный текста
  gets( word );          // удаляемое слова
```



### Слайд 194

Подключенные заголовочные файлы необходимы для применения перечисленных выше функций.

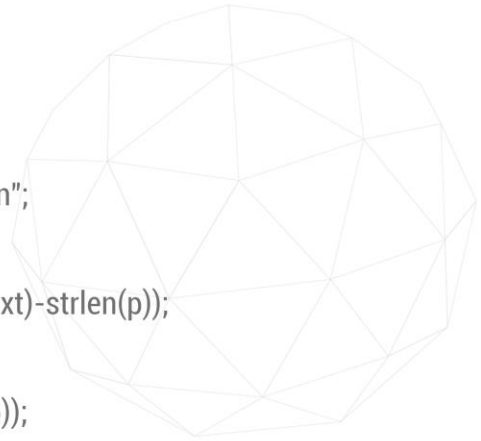
Исходный текст и удаляемое слово, или часть текста, описаны как массивы символьного типа. Конкретные размеры будут определены во время ввода с клавиатуры исходного текста и удаляемого слова. Ввод осуществляется с помощью функции **gets** .

Ввод строки символов завершается нажатием клавиши **Enter** . При этом, в конце строки, определяющей исходный текст и в конце строки, определяющей удаляемое слово, автоматически формируется признак конца строки.

Обратите внимание, текст, который будет сформирован из исходного текста, то есть **newText** , описан как символьный массив без указания размера. Пустые квадратные скобки рядом с именем массива вполне допустимы. Размер этого массива заранее не известен. Однако, в пустой текст введен признак конца строки, символ **\0** . Так как новый текст будет формироваться с помощью функции конкатенации, присутствие признака конца строки необходимо. При использовании функции конкатенации, признак конца строки автоматически не добавляется.

## УДАЛЕНИЕ ЧАСТИ СТРОКИ

```
char *p=strstr(text,word);  
if (!p) cout<<"\n Слово не найдено!\n";  
else  
{ strncat( newText, text, strlen(text)-strlen(p));  
  p=p+ strlen(word);  
  if (p) strncat(newText,p,strlen(p));  
  puts(newText);}  
getch(); }
```



### Слайд 195

Функция **strstr** ищет слово, которое необходимо удалить из текста. Результат работы этой функции – указатель на начало найденного слова. Под словом понимают любую цепочку символов.

Введена переменная **p**, указатель, для поиска слова в исходном тексте. Анализируется значение указателя, и если слово не найдено в тексте, появится соответствующее сообщение.

Если слово найдено, функция конкатенации **strncat** присоединит часть исходного текста до удаляемого слова к новому тексту. Признак конца слова, который внесен в новый массив в начале программы, автоматически отодвигается.

Указатель устанавливается в начало найденного слова в исходном тексте. Далее, указатель **p** отодвигается в тексте на длину удаляемого слова.

Еще раз, используя функцию конкатенации, присоединяем к новому тексту остаток исходного текста, после слова, если оно не последнее. Признак конца автоматически отодвигается.

Функция **puts** выведет на экран новый текст, до признака конца.

## ПОИСК СИММЕТРИЧНЫХ СЛОВ

Задача: распечатать симметричные слова из текста (например, асса)

Обозначения	
text []	исходный текст
word []	буфер слов
f	«флажок», признак симметричности



### Слайд 196

Рассмотрим определение симметричности слова. Симметричными считаются слова, которые одинаково читаются слева на право и справа на лево, так называемые палиндромы.

На слайде сформулирована задача и показаны введенные обозначения.

Это исходный текст и буфер слов, в который выделяется каждое слово текста и анализируется на симметричность. Они определены как массивы символов. Признаком конца слова подразумевается символ пробел.

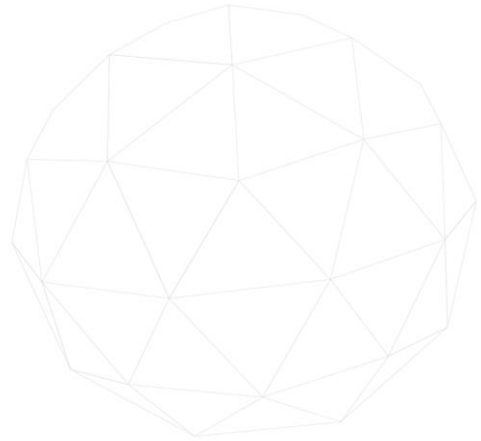
Конечно, на самом деле признаками конца слова являются и запятая, и двоеточие, и точка, если текст состоит из нескольких предложений, и так далее. Чтобы не загромождать программу в этом примере признаком конца слова считается символ пробел. Добавить дополнительные проверки на другие символы не составит сложности.

Каждое отдельное слово будет выделено в буфер - переменную **word** и проверяться на симметричность.

Переменная **f** - введена как признак симметричности.

## ПОИСК СИММЕТРИЧНЫХ СЛОВ

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
using namespace std;
int main()
{ char text[250],word[25];
  gets(text);
  int p=0;
```



### Слайд 197

На слайде представлен первый фрагмент программы. Подключены заголовочные файлы для использования функций обработки символьных массивов, строк.

Описаны исходный текст и буфер слов как одномерные массивы символов. Каждое слово текста, будет выделено в переменную **word** .

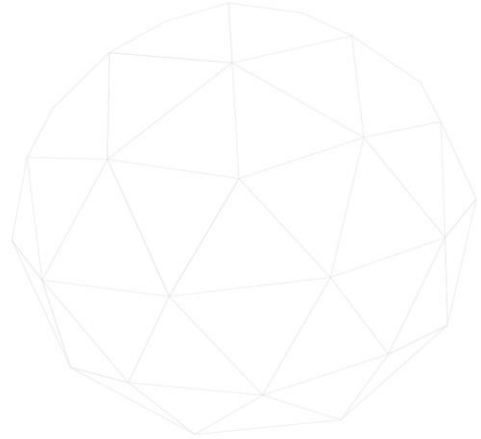
Для ввода исходного текста используется функция **gets** . Напомним, ввод цепочки символов завершается нажатием клавиши **Enter** . Если цепочка символом, то есть исходный текст, достаточно длинная, курсор сам перейдет на начало следующей строки и продолжается ввод.

Переменная **p** определена как начало каждого слова, то есть позиция начала каждого слова в исходном тексте, фактически это индекс одномерного массива символов. Переменная описана как целая, так как известно, что индексы массивов должны иметь целый тип.

Известно, что в любом массиве есть нулевой элемент. Поэтому этой переменной присвоено значение ноль, начало первого слова в исходном тексте.

## ПОИСК СИММЕТРИЧНЫХ СЛОВ

```
puts("\n Симметричные слова \n");
while( p<strlen(text) )
{ for( int j=0, i=p; text[i] != ' '; i++, j++)
    word[j]=text[i];
  word[j]='\0';
  p=i+1;
```



### Слайд 198

Для поиска симметричных слов используется структура вложенных циклов: внешний - цикла с предусловием; два внутренних – циклы с параметром.

Как говорилось выше, переменная **p** – позиция начала каждого слова, внешний цикл продолжается до тех пор, пока не достигли конца исходного текста.

Функция **strlen** - определяет длину исходного текста.

В первом внутреннем цикле, реализованном оператором **for** , каждый символ исходного текста переносится посимвольно в буфер слова от пробела до пробела.

Переменная **i** – используется как индекс исходного текста, значение которого на каждом шаге итерации увеличивается на единицу.

Переменная **j** – используется как индекс буфера слов, значение которого обнуляется перед каждым переносом очередного слова в буфер слов.

Первый внутренний цикл завершается, как только очередное слово до пробела пересыпано, перенесено в буфер.

Индекс **j** определен для буфера слов. Индекс **i** меняется в исходном тексте.

В переменную, определенную как буфер слов, добавляется признак конца, символ **\0** . Формируется признак конца строки. В операторе **for** значение переменной **j** , после завершения цикла на единицу больше, чем длина

очередного выделенного слова. В предыдущих темах мы подробно рассматривали принцип работы оператора **for** . Именно в эту позицию заносится признак конца строки.

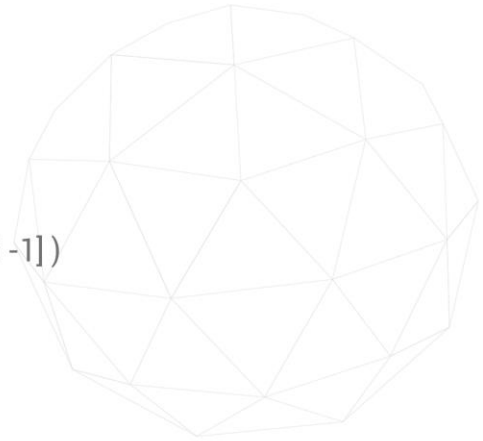
После каждого слова переменная **p** переносится на начало следующего слова в исходном тексте.



## СИММЕТРИЧНЫЕ СЛОВА

```
bool f=true;
for ( i=0; i<strlen(word)/2; i++)
    if ( word[ i ] != w[ strlen(word) - i -1] )
        f=false;

if ( f ) puts(word);
}
getch(); }
```



### Слайд 199

На слайде представлен следующий фрагмент программы. Тип переменной **f** определен как логический. Обратите внимание, перед каждым словом значению этой переменной присваивается значение **true**, истина. Предполагается, что слово симметрично.

Анализ каждого слова на симметричность выполняется во втором внутреннем цикле. Слово мысленно делится пополам и посимвольно сравниваются символы с левой и с правой сторон очередного слова. Функция **strlen** определяет длину очередного слова. В случае, если символы не совпали, выставляется признак несимметричности, то есть переменной **f** присваивается значение **false**, ложь. Количество проверок равно половине длины слова.

По окончании второго внутреннего цикла, в зависимости от значения переменной **f** выводятся на экран симметричные слова исходного цикла. Фигурная скобка закрывает внешний цикл.

Мы рассмотрели примеры, позволяющие осуществлять поиск в тексте, удалять часть текста, проверять текст на симметричность, чтобы продемонстрировать работу функций обработки массивов.

На этом мы завершаем изучение темы символьные массивы, строки.

----

Введение в языки программирования С и С++ | Уроки С++ - Ravesli /  
<https://ravesli.com/urok-2-vvedenie-v-yazyki-programmirovaniya-c-i-s/>

Т, А. Павловская С/С++ Программирование на языке высокого  
уровня/<http://cph.phys.spbu.ru/documents/First/books/7.pdf>

Введение в программирование | Уроки С++ - Ravesli/<https://ravesli.com/urok-1-vvedenie-v-programmirovanie/>

Технология программирования - Информатика, автоматизированные  
информационные технологии и системы/  
[https://studref.com/441961/informatika/tehnologiya\\_programmirovaniya](https://studref.com/441961/informatika/tehnologiya_programmirovaniya)

Бьерн Страуструп. Язык программирования С++ 11/ [https://vk.com/doc-145125017\\_450056359](https://vk.com/doc-145125017_450056359)

Язык СИ++ Учебное пособие / <http://5fan.ru/wievjob.php?id=4301>