* [Файловый ввод-вывод в языке C](https://foxford.ru/lessons/32263/conspects/1)
* [Файловый ввод-вывод в C++](https://foxford.ru/lessons/32263/conspects/2)
* [Простое использование файлового ввода-вывода при помощи функции freopen](https://foxford.ru/lessons/32263/conspects/3)
* [Работа со строками в С](https://foxford.ru/lessons/32263/conspects/4)
* [Работа со строками в C++ (класс string)](https://foxford.ru/lessons/32263/conspects/5)

**Файловый ввод-вывод в языке C**

**Файловый ввод-вывод в языке C**

Файловый ввод-вывод в языке C реализован при помощи функций

**fscanf**

 и

**fprintf**

. Они полностью аналогичны функциям

**scanf**

 и

**printf**

, только работают с файлами, а не с клавиатурой и экраном. Также у этих функций есть первый параметр - указатель на некоторый объект специального файлового типа, который сначала нужно открыть при помощи функции

**fopen**

. Функции fopen передается два параметра. Первый параметр - С-строка с именем открываемого файла, второй параметр - строка с указанием режима открытия файла. Режим открытия файла может быть "r", когда файл открывается на чтение (read) или "w" для открытия файла на запись. После окончания работы с файлами, файлы нужно "закрыть" при помощи функции

**fclose**

. Приведем пример программы, которая считывает два числа из файла с именем

**input.txt**

 и выводит результат в файл

**output.txt**

.

**#include<stdio.h>**

**int main()**

**{**

**FILE \* fin = fopen("input.txt", "r");**

**FILE \* fout = fopen("output.txt", "w");**

**int a, b;**

**fscanf(fin, "%d%d", &a, &b);**

**fprintf(fout, "%d\n", a + b);**

**fclose(fin);**

**fclose(fout);**

**}**

 Аналогично для считывания строки до символа конца строки из файла можно использовать функцию

**fgets**

 с тремя параметрами - строка, куда записывается результат, размер строки и указатель на файл. Например:

**char s[101]; fgets(s, 101, fin);**

Для проверки состояния файла можно использовать функцию feof с параметром - указателем на файл. Она возвращает значение 0 или 1 в зависимости от того, был ли обнаружен конец файла при чтении:

**if (feof(fin))**

Но при использовании этой функции могут возникнуть затруднения, например, с тем, что если после последнего числа в файле стоит символ конца строки, то состояние "достигнут конец файла" произойдет не после чтения последнего числа, а после следующего чтения. Более надежный способ считать последовательность чисел из файла до конца файла — использование значения, возвращаемого функцией

**fscanf**

:

**while (fscanf(fin, "%d", &a) == 1)**

**Файловый ввод-вывод в C++**

**Файловый ввод-вывод в языке C++**

В языке C++ ввод-вывод осуществляется через объекты классов

**ifstream**

 (для чтения данных) и

**ofstream**

 (для вывода данных). Эти классы объявлены в заголовочном файле

**fstream**

.

При создании этих объектов нужно передать в конструктор один параметр - имя файла.

После этого с этими объектами можно работать точно так же, как с объектами

**cin**

 и

**сout**

.

После окончания работы с файлами, файлы нужно "закрыть" эти объекты, вызвав для них метод

**close()**

.

Приведем пример программы, которая считывает два числа из файла с именем

**input.txt**

 и выводит результат в файл

**output.txt**

.

**#include<fstream>**

**using namespace std;**

**int main()**

**{**

**ifstream fin("input.txt");**

**ofstream fout("output.txt");**

**int a, b;**

**fin >> a >> b;**

**fout << a + b << endl;**

**fin.close();**

**fout.close();**

**}**

 Аналогично для считывания строки до символа конца строки из файла можно использовать функцию

**getline**

 с двумя параметрами - файловый объект, из которого читаются данные и строка, куда записывается результат. Например:

**string s;**

**getline(fin, s);**

Для проверки состояния файла можно использовать метод файла

**eof()**

. Он возвращает значение

**true**

 или

**false**

 в зависимости от того, был ли обнаружен конец файла при чтении:

**if (fin.eof())**

Но при использовании этого метода могут возникнуть затруднения, например, с тем, что если после последнего числа в файле стоит символ конца строки, то состояние "достигнут конец файла" произойдет не после чтения последнего числа, а после следующего чтения.

Более надежный (и простой!) способ считать последовательность чисел из файла до конца файла — использование значения, возвращаемого при считывании:

**while (fin >> d)**

#### Простое использование файлового ввода-вывода при помощи функции freopen

На олимпиадах часто для файлового ввода-вывода используют функцию freopen, определенную в языке C в заголовочном файле stdio.h. Эта функция позволяет "переоткрыть" файл, то есть связать уже открытый файловый дескриптор с существующим файлом на диске. На олимпиадах это устроено так - файловый дескриптор stdin, связанный со стандартным потоком ввода, переоткрывают на файл, который указан в условии задачи, как файл для хранения входных данных, а файловый дескриптор stdout переоткрывают на файл для выходных данных.

Пример использования:

**freopen("input.txt", "r", stdin);**

**freopen("output.txt", "w", stdout);**

После этого весь стандартный ввод-вывод будет перенаправляться в указанные файлы.

Функцию freopen можно использовать как вместе с выводом в языке C (функции printf и scanf), так и вместе с вводом-выводом в языке C++ (через потоки cin и cout).

#### Работа со строками в С

Строка — это последовательность ASCII или UNICODE символов.

**Строки в C**, как и в большинстве языков программирования высокого уровня рассматриваются как отдельный тип, входящий в систему базовых типов языка. Так как язык C по своему происхождению является языком системного программирования, то **строковый тип данных в C как таковой отсутствует, а в качестве строк в С используются обычные массивы символов**.

Исторически сложилось два представления формата строк:

1. формат ANSI;
2. cтроки с завершающим нулем (используется в C).

Формат ANSI устанавливает, что значением первой позиции в строке является ее длина, а затем следуют сами символы строки. Например, представление строки "Моя строка!" будет следующим:

**11 'М' 'о' 'я' ' ' 'с' 'т' 'р' 'о' 'к' 'а' '!'**

В строках с завершающим нулем, значащие символы строки указываются с первой позиции, а признаком завершения строки является значение ноль. Представление рассмотренной ранее строки в этом формате имеет вид:

**'М' 'о' 'я' ' ' 'с' 'т' 'р' 'о' 'к' 'а' '!' 0**

### Объявление строк в C

Строки реализуются посредством массивов символов. Поэтому объявление ASCII строки имеет следующий синтаксис:

**char имя[длина];**

Объявление **строки в С** имеет тот же синтаксис, что и объявление одномерного символьного массива. Длина строки должна представлять собой целочисленное значение (в стандарте C89 – константа, в стандарте C99 может быть выражением). Длина строки указывается с учетом одного символа на хранение завершающего нуля, поэтому максимальное количество значащих символов в строке на единицу меньше ее длины. Например, строка может содержать максимально двадцать символов, если объявлена следующим образом:  
char str[21]; Инициализация строки в С осуществляется при ее объявлении, используя следующий синтаксис:  
char str[длина] = строковый литерал;

Строковый литерал – строка ASCII символов заключенных в двойные кавычки. Примеры объявления строк с инициализацией:

**char str1[20] = "Введите значение: ", str2[20] = "";**

Пример:

**const char message[] = "Сообщение об ошибке!";**

### **Работа со строками в С**

Так как строки на языке С являются массивами символов, то к любому символу строки можно обратиться по его индексу. Для этого используется синтаксис обращения к элементу массива, поэтому первый символ в строке имеет индекс ноль. Например, в следующем фрагменте программы в строке str осуществляется замена всех символов 'a' на символы 'A' и наоборот.

**for(int i = 0; str[i] != 0; i++)**

**{**

**if (str[i] == 'a') str[i] = 'A';**

**else if (str[i] == 'A') str[i] = 'a';**

**}**

### Массивы строк в С

Объявление массивов строк в языке С также возможно. Для этого используются двумерные массивы символов, что имеет следующий синтаксис:  
char имя[количество][длина];

Первым размером матрицы указывается количество строк в массиве, а вторым – максимальная (с учетом завершающего нуля) длина каждой строки. Например, объявление массива из пяти строк максимальной длиной 30 значащих символов будет иметь вид:

**char strs[5][31];**

При объявлении массивов строк можно производить инициализацию:

**char имя[количество][длина] = {строковый литерал №1, … строковый литерал №N};**

Число строковых литералов должно быть меньше или равно количеству строк в массиве. Если число строковых литералов меньше размера массива, то все остальные элементы инициализируются пустыми строками. Длина каждого строкового литерала должна быть строго меньше значения длины строки (для записи завершающего нуля).

Например:

**char days[12][10] = {**

**"Январь", "Февраль", "Март", ”Апрель", "Май",**

**"Июнь", "Июль", "Август", "Сентябрь","Октябрь",**

**"Ноябрь", "Декабрь"**

**};**

При объявлении массивов строк с инициализацией допускается не указывать количество строк в квадратных скобках. В таком случае, количество строк в массиве будет определено автоматически по числу инициализирующих строковых литералов.  
Например, массив из семи строк:

**char days[][12] = {**

**"Понедельник", "Вторник", "Среда", "Четверг",**

**"Пятница", "Суббота", "Воскресенье"**

**};**

### Функции для работы со строками в С

Все библиотечные функции, предназначенные для работы со строками, можно разделить на три группы:

1. ввод и вывод строк;
2. преобразование строк;
3. обработка строк.

### Ввод и вывод строк в С

Для ввода и вывода строковой информации можно использовать функции форматированного ввода и вывода (printf и scanf). Для этого в строке формата при вводе или выводе строковой переменной необходимо указать спецификатор типа %s. Например, ввод и последующий вывод строковой переменной будет иметь вид:

**char str[31] = "";**

**printf("Введите строку: ");**

**scanf("%30s”,str);**

**printf("Вы ввели: %s”,str);**

Недостатком функции scanf при вводе строковых данных является то, что символами разделителями данной функции являются:

1. перевод строки,
2. табуляция;
3. пробел.

Поэтому, используя данную функцию невозможно ввести строку, содержащую несколько слов, разделенных пробелами или табуляциями. Например, если в предыдущей программе пользователь введет строку: "Сообщение из нескольких слов", то на экране будет выведено только "Сообщение".  
Для ввода и вывода строк в библиотеке stdio.h содержатся специализированные функции gets и puts.

Функция gets предназначена для ввода строк и имеет следующий заголовок:

**char \* gets(char \*buffer);**

Между тем использовать функцию

**gets**

 категорически не рекомендуется, ввиду того, что она не контролирует выход за границу строки, что может произвести к ошибкам. Вместо нее используется функция fgets с тремя параметрами:

**char \* fgets(char \* buffer, int size, FILE \* stream);**

где buffer - строка для записи результата, size - максимальное количество байт, которое запишет функция fgets, stream - файловый объект для чтения данных, для чтения с клавиатуры нужно указать stdin. Эта функция читает символы со стандартного ввода, пока не считает n - 1 символ или символ конца строки, потом запишет считанные символы в строку и добавит нулевой символ. При этом функция fgets записывает в том символ конца строки в данную строку, что нужно учитывать.

Функция puts предназначена для вывода строк и имеет следующий заголовок:

**int puts(const char \*string);**

Простейшая программа: ввод и вывод строки с использованием функций fgets и puts будет иметь вид:

**char str[102] = "";**

**printf("Введите строку: ");**

**fgets(str, 102, stdin);**

**printf("Вы ввели: ");**

**puts(str);**

Для считывания одного символа можно использовать функцию

**fgetc(FILE \* stream)**

. Она считывает один символ и возвращает значение этого символа, преобразованное к типу int, если же считывание не удалось, то возвращается специальная константа EOF, равная -1. Функция возвращает значение -1 для того, чтобы можно было обрабатывать ситуацию конца файла, посимвольное чтение до конца файла можно реализовать следующим образом:

**int c;**

**while ((c = fgetc(stdin)) != EOF) {**

**// Обработка символа**

**}**

Для вывода одного символа можно использовать функцию

**int fputc(int c, FILE \*stream);**

.

Помимо функций ввода и вывода в потоки в библиотеке stdio.h присутствуют функции форматированного ввода и вывода в строки. Функция форматированного ввода из строки имеет следующий заголовок:

**int sscanf(const char \* restrict buffer, const char \* restrict string, [address] …);**

Функции форматированного вывода в строку имеют следующие заголовки:

**int sprintf(char \* restrict buffer,**

**const char \* restrict format, [argument] …);**

**int snprintf(char \* restrict buffer, size\_t maxsize,**

**const char \* restrict format, [argument] …);**

### Преобразование строк

В С для преобразования строк, содержащих числа, в численные значения в библиотеке stdlib.h  
предусмотрен следующий набор функций:

**double atof(const char \*string);**

 // преобразование строки в число типа double

**int atoi(const char \*string);**

 // преобразование строки в число типа int

**long int atol(const char \*string);**

 // преобразование строки в число типа long int

**long long int atoll(const char \*string);**

 // преобразование строки в число типа long long int

Корректное представление вещественного числа в текстовой строке должно удовлетворять формату:

[ ][{+|-}][цифры][.цифры][{e|E}[{+|-}]цифры]

После символов E, e указывается порядок числа. Корректное представление целого числа в текстовой строке должно удовлетворять формату:

[ ] [{+|-}] цифры

Помимо приведенных выше функций в библиотеке stdlib.h доступны также следующие функции преобразования строк в вещественные числа:

**float strtof(const char \* restrict string, char \*\* restrict endptr);**

**double strtod(const char \* restrict string, char \*\* restrict endptr);**

**long double strtold(const char \* restrict string,char \*\* restrict endptr);**

Аналогичные функции присутствуют и для преобразования строк в целочисленные значения:

**long int strtol(const char \* restrict string, char \*\* restrict endptr, int base);**

**unsigned long strtoul(const char \* restrict string,**

**char \*\* restrict endptr, int base);**

**long long int strtoll(const char \* restrict string,**

**char \*\* restrict endptr, int base);**

**unsigned long long strtoull(const char \* restrict string,char \*\* restrict endptr, int base);**

Функции обратного преобразования (численные значения в строки) в библиотеке stdlib.h присутствуют, но они не регламентированы стандартом, и рассматриваться не будут. Для преобразования численных значений в строковые наиболее удобно использовать функции sprintf и snprintf.

### Обработка строк

В библиотеке string.h содержаться функции для различных действий над строками.  
Функция вычисления длины строки:

**size\_t strlen(const char \*string);**

Пример:

**char str[] = "1234";**

**int n = strlen(str); //n == 4**

Функции копирования строк:

**char \* strcpy(char \* restrict dst, const char \* restrict src);**

**char \* strncpy(char \* restrict dst, const char \* restrict src, size\_t num);**

Функции сравнения строк:

**int strcmp(const char \*string1, const char \*string2);**

**int strncmp(const char \*string1, const char \*string2,size\_t num);**

Функции осуществляют сравнение строк по алфавиту и возвращают:

положительное значение – если string1 больше string2;  
отрицательное значение – если string1 меньше string2;  
нулевое значение – если string1 совпадает с string2;

Функции объединения (конкатенации) строк:

**char \* strcat(char \* restrict dst, const char \* restrict src);**

**char \* strncat(char \* restrict dst, const char \* restrict src, size\_t num);**

Функции поиска символа в строке:

**char \* strchr(const char \*string, int c);**

**char \* strrchr(const char \*string, int c);**

Функция поиска строки в строке:

**char \* strstr(const char \*str, const char \*substr);**

Пример:

**char str[] = "Строка для поиска";**

**char \*str1 = strstr(str,"для"); //str1 == "для поиска"**

Функция поиска первого символа в строке из заданного набора символов:

**size\_t strcspn(const char \*str, const char \*charset);**

Функции поиска первого символа в строке не принадлежащему заданному набору символов:

**size\_t strspn(const char \*str, const char \*charset);**

Функции поиска первого символа в строке из заданного набора символов:

**char \* strpbrk(const char \*str, const char \*charset);**

Функция поиска следующего литерала в строке:

**char \* strtok(char \* restrict string, const char \* restrict charset);**

#### Работа со строками в C++ (класс string)

## Основы работы со строками в C++

В языке C++ для удобной работы со строками есть класс string, для использования которого необходимо подключить заголовочный файл string.

Строки можно объявлять и одновременно присваивать им значения:

**string S1, S2 = "Hello";**

Строка S1 будет пустой, строка S2 будет состоять из 5 символов.

К отдельным символам строки можно обращаться по индексу, как к элементам массива или C-строк. Например

**S[0]**

 - это первый символ строки.

Для того, чтобы узнать длину строки можно использовать метод size() строки. Например, последний символ строки S это

**S[S.size() - 1**

].

### Конструкторы строк

Строки можно создавать с использованием следующих конструкторов:  
string() - конструктор по умолчанию (без параметров) создает пустую строку.  
string(string & S) - копия строки S  
string(size\_t n, char c) - повторение символа c заданное число n раз.  
string(size\_t c) - строка из одного символа c.  
string(string & S, size\_t start, size\_t len) - строка, содержащая не более, чем len символов данной строки S, начиная с символа номер start.

Конструкторы можно вызывать явно, например, так:

**S += string(10, 'z');**

В этом примере явно вызывается конструктор string для создания строки, состоящей из 10 символов 'z'.

Неявно конструктор вызывается при объявлении строки с указанием дополнительных параметров. Например, так:

**string S(10, 'z');**

Подробней о конструкторах для строк читайте [здесь](http://www.cppreference.com/wiki/ru/string/string_constructors).

### 

### Ввод-вывод строк

Строка выводится точно так же, как и числовые значения:

**cout << S;**

Для считывания строки можно использовать операцию ">>" для объекта cin:

**cin >> S;**

В этом случае считывается строка из непробельных символов, пропуская пробелы и концы строк. Это удобно для того, чтобы разбивать текст на слова, или чтобы читать данные до конца файла при помощи

**while (cin >> S)**

.

Можно считывать строки до появления символа конца строки при помощи функции getline. Сам символ конца строки считывается из входного потока, но к строке не добавляется:

**getline(cin S);**

### Арифметические операторы

Со строками можно выполнять следующие арифметические операции:  
= - присваивание значения.  
+= - добавление в конец строки другой строки или символа.  
+ - конкатенация двух строк, конкатенация строки и символа.  
==, != - посимвольное сравнение.  
<, >, <=, >= - лексикографическое сравнение.

То есть можно скопировать содержимое одной строки в другую при помощи операции S1 = S2, сравнить две строки на равенство при помощи S1 == S2, сравнить строки в лексикографическом порядке при помощи S1 < S2, или сделать сложение (конкатенацию) двух строк в виде S = S1 + S2.

Подробней об операторах для строк читайте [здесь](http://www.cppreference.com/wiki/ru/string/string_operators).

### Методы строк

У строк есть разные методы, многие из них можно использовать несколькими разными способами (с разным набором параметров).

Рассмотрим эти методы подробней.

#### size

Метод size() возращает длину длину строки. Возвращаемое значение является беззнаковым типом (как и во всех случаях, когда функция возращает значение, равное длине строке или индексу элемента - эти значения беззнаковые). Поэтому нужно аккуратно выполнять операцию вычитания из значения, которое возвращает size(). Например, ошибочным будет запись цикла, перебирающего все символы строки, кроме последнего, в виде for (int i = 0; i < S.size() - 1; ++i).

Кроме того, у строк есть метод length(), который также возвращает длину строки.

Подробней о методе [size](http://www.cppreference.com/wiki/ru/string/size).

#### resize

**S.resize(n)**

 - Изменяет длину строки, новая длина строки становится равна n. При этом строка может как уменьшится, так и увеличиться. Если вызвать в виде

**S.resize(n, c)**

, где c - символ, то при увеличении длины строки добавляемые символы будут равны c.

Подробней о методе [resize](http://www.cppreference.com/wiki/ru/string/resize).

#### clear

**S.clear()**

 - очищает строчку, строка становится пустой.

Подробней о методе [clear](http://www.cppreference.com/wiki/ru/string/clear).

#### empty

**S.empty()**

 - возвращает true, если строка пуста, false - если непуста.

Подробней о методе [empty](http://www.cppreference.com/wiki/ru/string/empty).

#### push\_back

**S.push\_back(c)**

 - добавляет в конец строки символ c, вызывается с одним параметром типа char.

Подробней о методе [push\_back](http://www.cppreference.com/wiki/ru/string/push_back).

#### append

Добавляет в конец строки несколько символов, другую строку или фрагмент другой строки. Имеет много способов вызова.

**S.append(n, c)**

 - добавляет в конец строки n одинаковых символов, равных с. n имеет целочисленный тип, c - char.

**S.append(T)**

 - добавляет в конец строки S содержимое строки T. T может быть объектом класса string или C-строкой.

**S.append(T, pos, count)**

 - добавляет в конец строки S символы строки T начиная с символа с индексом pos количеством count.

Подробней о методе [append](http://www.cppreference.com/wiki/ru/string/append).

#### erase

**S.erase(pos)**

 - удаляет из строки S  с символа с индексом pos и до конца строки.

**S.erase(pos, count)**

 - удаляет из строки S  с символа с индексом pos количеством count или до конца строки, если pos + count > S.size().

Подробней о методе [erase](http://www.cppreference.com/wiki/ru/string/erase).

#### insert

Вставляет в середину строки несколько символов, другую строку или фрагмент другой строки. Способы вызова аналогичны способам вызова метода append, только первым параметром является значение i - позиция, в которую вставляются символы. Первый вставленный символ будет иметь индекс i, а все символы, которые ранее имели индекс i и более сдвигаются вправо.

**S.insert(i, n, c)**

 - вставить n одинаковых символов, равных с. n имеет целочисленный тип, c - char.

**S.insert(i, T)**

 - вставить содержимое строки T. T может быть объектом класса string или C-строкой.

**S.insert(i, T, pos, count)**

 - вставить символы строки T начиная с символа с индексом pos количеством count.

Подробней о методе [insert](http://www.cppreference.com/wiki/ru/string/insert).

#### substr

**S.substr(pos)**

 - возвращает подстроку данной строки начиная с символа с индексом pos и до конца строки.

**S.substr(pos, count)**

 - возвращает подстроку данной строки начиная с символа с индексом pos количеством count или до конца строки, если pos + count > S.size().

Подробней о методе [substr](http://www.cppreference.com/wiki/ru/string/substr).

#### replace

Заменяет фрагмент строки на несколько равных символов, другую строку или фрагмент другой строки. Способы вызова аналогичны способам вызова метода append, только первыми двумя параметрами являются два числа: pos и count. Из данной строки удаляется count символов, начиная с символа pos, и на их место вставляются новые символы.

**S.replace(pos, count, n, c)**

 - вставить n одинаковых символов, равных с. n имеет целочисленный тип, c - char.

**S.replace(pos, count, T)**

 - вставить содержимое строки T. T может быть объектом класса string или C-строкой.

**S.replace(pos, count, T, pos2, count2)**

 - вставить символы строки T начиная с символа с индексом pos количеством count.

Подробней о методе [replace](http://www.cppreference.com/wiki/ru/string/replace).

#### find

Ищет в данной строке первое вхождение другой строки str. Возвращается номер первого символа, начиная с которого далее идет подстрока, равная строке str. Если эта строка не найдена, то возвращается константа string::npos (которая равна -1, но при этом является беззнаковой, то есть на самом деле является большим безннаковым положительным числом).

Если задано значение pos, то поиск начинается с позиции pos, то есть возращаемое значение будет не меньше, чем pos. Если значение pos не указано, то считается, что оно равно 0 - поиск осуществляется с начала строки.

**S.find(str, pos = 0)**

 - искать первое входение строки str начиная с позиции pos. Если pos не задано - то начиная с начала строки S.

**S.find(str, pos, n)**

 - искать в данной строке подстроку, равную первым n символам строки str. Значение pos должно быть задано.

Подробней о методе [find](http://www.cppreference.com/wiki/ru/string/find).

#### rfind

Ищет последнее вхождение подстроки ("правый" поиск). Способы вызова аналогичны способам вызова метода find.

Подробней о методе [rfind](http://www.cppreference.com/wiki/ru/string/rfind).

#### find\_first\_of

Ищет в данной строке первое появление любого из символов данной строки str. Возвращается номер этого символа или значение string::npos.

Если задано значение pos, то поиск начинается с позиции pos, то есть возращаемое значение будет не меньше, чем pos. Если значение pos не указано, то считается, что оно равно 0 - поиск осуществляется с начала строки.

**S.find\_first\_of(str, pos = 0)**

 - искать первое входение любого символа строки str начиная с позиции pos. Если pos не задано - то начиная с начала строки S.

Подробней о методе [find\_first\_of](http://www.cppreference.com/wiki/ru/string/find_first_of).

#### find\_last\_of

Ищет в данной строке последнее появление любого из символов данной строки str. Способы вызова и возвращаемое значение аналогичны методу find\_first\_of.

Подробней о методе [find\_last\_of](http://www.cppreference.com/wiki/ru/string/find_last_of).

#### find\_first\_not\_of

Ищет в данной строке первое появление символа, отличного от символов строки str. Способы вызова и возвращаемое значение аналогичны методу find\_first\_of.

Подробней о методе [find\_first\_not\_of](http://www.cppreference.com/wiki/ru/string/find_first_not_of).

#### find\_last\_not\_of

Ищет в данной строке последнее появление символа, отличного от символов строки str. Способы вызова и возвращаемое значение аналогичны методу find\_first\_of.

Подробней о методе [find\_last\_not\_of](http://www.cppreference.com/wiki/ru/string/find_last_not_of).

#### c\_str

Возвращает указать на область памяти, в которой хранятся символы строки, возвращает значение типа char\*. Возвращаемое значение можно рассматривать как C-строку и использовать в функциях, которые должны получать на вход C-строку.

Подробней о методе [c\_str](http://www.cppreference.com/wiki/ru/string/c_str).