

C++

Краткий справочник для 8 класса



Последнее изменение 08.03.19 г.

Оглавление

Таблица 1. Структура программы.....	3
Таблица 2. Локализация в стиле C++.....	4
Таблица 3. Ключевые слова.....	4
Таблица 4. Операции.....	5
Таблица 5. Константы.....	6
Таблица 6. Целочисленные типы.....	7
Таблица 7. Таблица значений, которые могут принимать целые типы.....	7
Таблица 8. Действительные типы.....	8
Таблица 9. Преобразование типов.....	8
Таблица 10. Некоторые функции <code>cmath</code>	8
Таблица 11. Математические константы.....	9
Таблица 12. Escape-последовательности.....	10
Таблица 13. Основные конструкции языка C++.....	10
Таблица 14. Флаги форматирования.....	12
Таблица 15. Манипуляторы.....	13
Таблица 16. Форматирующие функции-члены.....	15
Таблица 17. Функции заголовка <code>cctype</code>	15
Таблица 18. Методы класса <code>string</code>	15
Таблица 19. Методы класса <code>vector</code>	22

Таблица 1. Структура программы.

```
// Это однострочный комментарий
/*
    Это многострочный комментарий
*/
// Директивы с указанием заголовочных файлов
// Библиотека ввода/вывода
#include <iostream>
// Динамические строки C++
#include <string>

// Объявление использования имен стандартной библиотеки (STD)
using namespace std;

// Главная функция программы
int main() {
    // Русский текст в консоли Windows (в Linux не нужен)
    setlocale(LC_ALL, "");
    // Определение переменных
    int t;    // Целый тип
    string s; // Тип string
    // Консольный диалог
    // Вывод на экран
    cout << "Как вас зовут?\n";
    // Ввод строки
    getline(cin, s);
    cout << "Сколько вам лет?\n";
    // Ввод переменной
    cin >> t;
    // Добавление объектов в поток
    cout << "Вас зовут "
         << s
         << ",\nВам " // \n символ перевода на новую строку
         << t
         << " лет."
         << endl;    // манипулятор endl освобождает буфер вывода
                    // и завершает строку с переводом на новую
    // Возврат функции main успешного выполнения
    return 0;
}
```

Таблица 2. Локализация в стиле C++

Строки широких символов

```
#include <iostream>
#include <locale>
using namespace std;

int main () {
    // Глобальный объект локали
    locale::global(locale("ru_RU.UTF-8"));
    wstring G1;
    wcout << L"Введите строку G1:" ;
    getline(wcin, G1);
    wcout << L"Введите символ G2:" << endl;
    wchar_t G2;
    // Дальнейший текст программы
    ...
    return 0;
}
```

Таблица 3. Ключевые слова

alignas	decltype	namespace	struct
alignof	default	new	switch
and	delete	noexcept	template
and_eq	do	not	this
asm	double	not_eq	thread_local
auto	dynamic_cast	nullptr	throw
bitand	else	operator	true
bitor	enum	or	try
bool	explicit	or_eq	typedef
break	export	private	typeid
case	extern	protected	typename
catch	false	public	union
char	float	register	unsigned
char16_t	for	reinterpret_cast	using
char32_t	friend	return	virtual
class	goto	short	void
compl	if	signed	volatile
const	inline	sizeof	wchar_t
constexpr	int	static	while
const_cast	long	static_assert	xor
continue	mutable	static_cast	xor_eq

Таблица 4. Операции

Приоритет	Операция	Описание	Ассоциативность
1	::	Область видимости	Слева-направо
2	++, --	Суффиксальный/постфиксный инкремент и декремент	
	()	Вызов функции	
	[]	Обращение к массиву по индексу	
	.	Выбор элемента по ссылке	
	->	Выбор элемента по указателю	Справа-налево
3	++, --	Префиксный инкремент и декремент	
	+, -	Унарный плюс и минус	
	!, ~	Логическое НЕ и побитовое НЕ	
	(type)	Приведение к типу type	
	*	Косвенная адресация (разыменование)	
	&	Адрес	
	sizeof	Размер	
	new, new[]	Динамическое выделение памяти	
delete, delete[]	Динамическое освобождение памяти		
4	.*, ->*	Указатель на член	Слева-направо
5	*, /, %	Умножение, деление и остаток	
6	+, -	Сложение и вычитание	
7	<<, >>	Побитовый сдвиг влево и сдвиг вправо	
8	<, <=	Операторы сравнения < и ≤ соответственно	
	>, >=	Операторы сравнения > и ≥ соответственно	
9	==, !=	Операторы сравнения = и ≠ соответственно	

Приоритет	Операция	Описание	Ассоциативность
10	&	Побитовое И	
11	^	Побитовый XOR (исключающее или)	
12		Побитовое ИЛИ (inclusive or)	
13	&&	Логическое И	
14		Логическое ИЛИ	
15	?:	Тернарное условие	Справа-налево Справа-налево
	=	Прямое присваивание (предоставляемое по умолчанию для C++ классов)	
	+=, -=	Присвоение с суммированием и разностью	
	*=, /=, %=	Присвоение с умножением, делением и остатком от деления	
	<<=, >>=	Присвоение с побитовым сдвигом влево и сдвигом вправо	
15	&=, ^=, =	Присвоение с побитовыми AND, XOR, и OR	
16	throw	Генерация исключительной ситуации	
17	,	Запятая	Слева-направо

Таблица 5. Константы

Литералы	Формат	Примеры
Целые	Двоичный	0b1010, 0B111100111
	Восьмеричный	01, 020, 07155
	Десятичный	0, 1992211, 33
	Шестнадцатеричный	0xA, 0x1B8, 0X00FF
Логические		true, false
Вещественные	С фиксированной точкой	5.7, .001, 35.
	С плавающей точкой (экспоненциальный)	0.2E6, .11e-3, 5E10, 2.0e-10
Символьные	Один или два символа, заключенных в апострофы	'F', 'ю', 'db', '\n', '\0', '\012', '\x07\x07'
Строковые	Последовательность символов,	"Здесь был Vasia", "\

	заклученная в кавычки	tЗначение r = \0xF5\n"
Прочие	Нулевой указатель	<i>nullpt</i>
	User-defined (определяемые пользователем)	90.0_deg, "SDD"_prt
	Параметр пакета	...

Таблица 6. Целочисленные типы

Название	Суффикс	Размер	Знаковый	Синонимы
short		2 байта	Знаковый	short int, signed short, signed short int
unsigned short		2 байта	Беззнаковый	unsigned short int
int		4 байта	Знаковый	signed int
unsigned	u	4 байта	Беззнаковый	unsigned int
long	l	4 байта	Знаковый	long int, signed long, signed long int
unsigned long	ul	4 байта	Беззнаковый	unsigned long int
long long	ll	8 байт	Знаковый	long long int, signed long long, signed long long int
unsigned long long	ull	8 байт	Беззнаковый	unsigned long long int

Таблица 7. Таблица значений, которые могут принимать целые типы

Название	Размер*	MIN	MAX
short	16 бит	$-2^{15} = -32768$	$2^{15}-1 = 32767$
unsigned short	16 бит	0	$2^{16}-1 = 65535$
int, long**	32 бита	$-2^{31} = -2147483648$	$2^{31}-1 = 2147483647$
unsigned, unsigned long	32 бита	0	$2^{32}-1 = 4294967295$
long long	64 бита	$-2^{63} = -9223372036854775808$	$2^{63}-1 = 9223372036854775807$

Название	Размер*	MIN	MAX
unsigned long long	64 бита	0	$2^{64}-1 = 18446744073709551615$

*Зависит от реализации.

**GCC 64bit соответствует ширине long long, а unsigned long – unsigned long long

Таблица 8. Действительные типы

Тип, суффикс	Точность, мантисса	Размер	MIN	MAX
float, f	Одинарная, 7	4 байта	$1.1754944e-38$	$3.4028235e+38$
double	Двойная, 15	8 байт	$2.225073858507201e-308$	$1.797693134862316e+308$
long double, l	Расширенная, 19	10 байт	$3.3621031431120935063e-4932$	$1.189731495357231765e+4932$

Таблица 9. Преобразование типов

Рекомендуемый способ:
<code>static_cast<type>(expression)</code> type – тип к которому приводится выражение expression.
Преобразование в стиле вызова функции (в стиле C++) и в стиле C:
<code>type(expression)</code> <code>(type)expression</code>

Таблица 10. Некоторые функции smath

Функция*	Описание
Округление	
<code>round</code>	Округляет число по правилам арифметики. Для возвращения целых значений используются префиксы <code>l</code> и <code>ll</code> (<code>lround</code> , <code>llround</code>)
<code>floor</code>	Округляет число вниз ("пол"), при этом <code>floor(1.5) == 1</code> , <code>floor(-1.5) == -2</code>
<code>ceil</code>	Округляет число вверх ("потолок"), при этом <code>ceil(1.5) == 2</code> , <code>ceil(-1.5) == -1</code>
<code>trunc</code>	Округление в сторону нуля (отбрасывание дробной части)

Функция*	Описание
<code>abs</code>	Модуль (абсолютная величина) целого числа (int). Или с префиксами <code>labs</code> и <code>llabs</code>
<code>fabs</code>	Модуль вещественного числа
<code>modf</code>	Разлагает на целую и дробную части (сохраняется по указателю: <code>mod(x, &p)</code>)
Корни, степени, логарифмы	
<code>sqrt</code>	Квадратный корень. Использование: <code>sqrt(x)</code>
<code>pow</code>	Возведение в степень, возвращает a^b . Использование: <code>pow(a, b)**</code>
<code>exp</code>	Экспонента, возвращает e^x . Использование: <code>exp(x)</code>
<code>log</code>	Натуральный логарифм
<code>log10</code>	Десятичный логарифм
<code>hypot</code>	Гипотенуза <code>hypot(x, y)**</code>
Тригонометрия	
<code>sin</code>	Синус угла, задаваемого в радианах
<code>cos</code>	Косинус угла, задаваемого в радианах
<code>tan</code>	Тангенс угла, задаваемого в радианах
<code>asin</code>	Арксинус, возвращает значение в радианах
<code>acos</code>	Арккосинус, возвращает значение в радианах
<code>atan</code>	Арктангенс, возвращает значение в радианах

*Обычно возвращаемый тип соответствует типу аргумента/-ов .

** Если какой-то аргумент имеет целый тип, то функция вернет `double`

Таблица 11. Математические константы

Литерал*	Значение	Литерал	Значение
<code>M_E</code>	e	<code>M_1_PI</code>	$1/\pi$
<code>M_LOG2E</code>	$\log_2(e)$	<code>M_2_PI</code>	$2/\pi$
<code>M_LOG10E</code>	$\log_{10}(e)$	<code>M_2_SQRTPI</code>	$2/\sqrt{\pi}$
<code>M_LN2</code>	$\ln(2)$	<code>M_SQRT2</code>	$\sqrt{2}$
<code>M_LN10</code>	$\ln(10)$	<code>M_SQRT1_2</code>	$1/\sqrt{2}$
<code>M_PI</code>	π	<code>M_PI_4</code>	$\pi/4$
<code>M_PI_2</code>	$\pi/2$	*необходимо подключить <code><cmath></code>	

Таблица 12. Escape-последовательности

Символ	Описание	Представление
\'	одинарная кавычка	байт 0x27
\"	двойная кавычка	байт 0x22
\?	вопросительный знак	байт 0x3f
\\	обратный слеш	байт 0x5c
\0	нулевой символ	байт 0x00
\a	звуковой сигнал	байт 0x07
\b	забой	байт 0x08
\f	перевод строки – новая строка	байт 0x0c
\n	перевод строки – новая строка	байт 0x0a
\r	возврат каретки	байт 0x0d
\t	горизонтальная табуляция	байт 0x09
\v	вертикальная табуляция	байт 0x0b
\nnn	произвольное восьмеричное значение	байт nnn
\xnn	произвольное шестнадцатеричное значение	байт nn
\unnnn \Uxxxxxxxx	Произвольное Юникод-значение. Результатом могут быть несколько символов.	кодировка позиция U+nnnn U+nnnnn

Таблица 13. Основные конструкции языка C++

Ввод/вывод	Условная инструкция if
Вывод на строки дисплей: <code>cout << "Здравствуй, мир!" << endl;</code> Ввод данных с клавиатуры: <code>cin >> var1 >> var2 >> varN;</code>	<pre> if (условие) { // Инструкции, если условие true } else { // необязательная часть // Инструкции, если условие false } </pre>
Инструкции циклов с условием	Инструкция выбора switch
Цикл с предусловием: <code>while (условие) {</code> <code> // тело цикла;</code> <code>}</code> Цикл с постусловием: <code>do {</code> <code> // тело цикла;</code> <code>} while (условие);</code> Бесконечный цикл: <code>while (true) {...}</code>	<pre> switch (управляющее выражение) { case const_1: инструкции_1; break; case const_2: инструкции_2; break; ... default: инструкции_по_умолчанию; } </pre>

Инструкция цикла со счетчиком	Шаблон
<p>Цикл с параметром for: for (инициализация; условие; модификация) { // тело цикла }</p> <p>Бесконечный цикл: for (;;) {...}</p> <p>Цикл for основанный на диапазоне: for (auto &r : arr) {...}</p>	<pre>template <typename T1, typename T2, ...> // Определение функции</pre>
Функции	С-массивы
<p>Объявление (прототип): void myFunc1(<i>типы параметров</i>); type myFunc2(<i>типы параметров</i>);</p> <p>Вызов: myFunc1(<i>аргументы</i>); cout << myFunc2(<i>аргументы</i>);</p> <p>Описание: void myFunc1(<i>параметры</i>) { // Тело функции } type myFunc2(<i>параметры</i>) { // Тело функции }</p>	<pre>int mass1[9] {1, 2, 3, 4, 5, 6, 7, 8, 9} компилятор подсчитает сам float mass2[] {5.2, 6.5, 2.1, 0.7, 1.92} int mass3[20] {} // Обнулить элементы Вывод (N – элементов. Первый элемент – mass[0]) for(int i = 0; i < N; i++){ cout << mass[i] << endl; } Ввод for(int i = 0; i < N; i++){ cin >> mass[i]; }</pre>
Структуры	Доступ к элементам структуры
<p>Определение структуры: struct structName { type atrib; // остальные элементы структуры }; // точка с запятой – обязательна // объявление переменных абстрактного // типа: structName structVar1, structVar2, ...;</p>	<p>Указатель на переменную структуры structName: structName *structPtr = &structVar1; доступ с помощью операции «.» : structVar1.atrib доступ с помощью операции «->»: structPtr -> atrib что равносильно (*structPtr).atrib</p>
Классы	Исключения
<pre>class Имя_класса { private: // Закрытые члены класса public: // Открытые члены класса // Конструктор Имя_класса (<i>параметры</i>) {</pre>	<pre>try { // Блок контролируемого кода } catch (exception1) { // Будет выполняться этот блок кода, // если перехвачено исключение типа exception1 }</pre>

<pre>// Инициализация полей; } // Или списком инициализации Имя_класса (параметры) : var1 (инициализатор), var2 (инициализатор) //... {} };</pre>	<pre>catch (exception2) { // Блок обработки исключения exception2 } catch (exception3) { // Блок обработки исключения exception3 throw; } catch (...) { // Обрабатывается исключение // любого другого типа, }</pre>
Функтор	Лямбда-выражение
<pre>struct Имя_функтора { // Поля структуры // Конструктор, если необходим void operator() () { // Инструкции; } }; // Или class Имя_функтора { // Поля класса public: // Конструктор, если необходим void operator() () { //Инструкции; } };</pre>	<p>Полная версия: [список захвата] (список параметров) mutable exception attribute -> возвращаемый тип {тело функций} или сокращенная: [список захвата] (список параметров) {тело функций}</p> <p>Список захвата: [a,&b] a захвачена по значению, a b захвачена по ссылке. [this] захватывает указатель this по значению. [&] захват всех объектов по ссылке [=] захват всех объектов по значению [] ничего не захватывать</p>

Таблица 14. Флаги форматирования

Флаг*	Назначение
hex	Значения целого типа преобразуются к основанию 16 (как шестнадцатеричные)
dec	Значения целого типа преобразуются к основанию 10 (по умолчанию)
oct	Значения целого типа преобразуются к основанию 8 (как восьмеричные)
boolalpha	Вывод логических величин в текстовом виде
fixed	Числа с плавающей точкой выводятся в формате с фиксированной точкой (то есть nnn.ddd)

Флаг*	Назначение
scientific	Числа с плавающей точкой выводятся в так называемой научной записи (то есть n.xxxEyy)
showbase	Выводится основание системы счисления в виде префикса к целому числовому значению (например, число 1FE выводится как 0x1FE)
showpoint	Всегда показывать десятичную точку
showpos	При выводе положительных числовых значений выводится знак плюс
uppercase	Заменяет определенные символы нижнего регистра на символы верхнего регистра (символ "e" при выводе чисел в научной нотации на "E" и символ "x" при выводе 16-ричных чисел на "X")
left	Данные при выводе выравниваются по левому краю поля
right	Данные при выводе выравниваются по правому краю поля (по умолчанию)
internal	Добавляются символы-заполнители между всеми цифрами и знаками числа для заполнения поля вывода
skipws	Ведущие символы-заполнители (знаки пробела, табуляции и перевода на новую строку) отбрасываются
unitbuf	Очищаются все выходные потоки после каждой операции вставки в поток

*Для включения параметра используется следующий синтаксис: `cout.setf(ios::flag)`. Для выключения: `cout.unsetf(ios::flag)`

Таблица 15. Манипуляторы

Манипулятор*	Заголовок	Описание
setw(n)	iomanip	Определяет ширину поля вывода в n символов
setprecision(n)	iomanip	Определяет количество цифр (n-1) в дробной части числа
left	ostream	Выравнивание по левой границе
right	ostream	Выравнивание по правой границе (по умолчанию)
boolalpha	ostream	Вывод логических величин в текстовом виде

Манипулятор*	Заголовок	Описание
nboolalpha	iostream	Вывод логических величин в числовом виде
dec	iostream	Вывод величин в десятичной системе счисления (по умолчанию)
oct	iostream	Вывод величин в восьмеричной системе счисления
hex	iostream	Вывод величин в шестнадцатеричной системе счисления
ws	iostream	Удаляет ведущие пробелы из входного потока
showbase	iostream	Выводить индикатор основания системы счисления
noshowbase	iostream	Не выводить индикатор основания системы счисления
uppercase	iostream	Использовать в числах прописные буквы
lowercase	iostream	Использовать в числах строчные буквы
showpos	iostream	Выводить знак + для положительных чисел
noshowpos	iostream	Не выводить знак + для положительных чисел
scientific	iostream	Экспоненциальная форма вывода вещественных чисел
fixed	iostream	Фиксированная форма вывода вещественных чисел (по умолчанию)
setfill(c)	omanip	Установить символ c как заполнитель
endl	iostream	Вставляет символ '\n' и очищает входной поток
ends	iostream	Вставляет символ '\0' во входной поток
flush	iostream	Очищает входной поток

**Манипуляторы добавляются в поток ввода/вывода перегруженными операциями «>>>» и «<<<»*

Таблица 16. Форматирующие функции-члены

Метод	Описание
<code>cout.fill(char)</code>	устанавливает символ заполнитель
<code>cout.width(int)</code>	задает ширину поля
<code>cout.precision(int)</code>	задает количество знаков после десятичной точки

Таблица 17. Функции заголовка `cctype`

Функция*	Описание
<code>isalnum</code>	проверяет, является ли символ буквенно-цифровым
<code>isalpha</code>	проверяет, является ли символ буквенным
<code>isdigit</code>	проверяет, является ли символ цифрой
<code>isxdigit</code>	проверяет, является ли символ шестнадцатеричной цифрой
<code>iscntrl</code>	проверяет, является ли символ управляющим символом
<code>isspace</code>	проверяет, является ли символ символом пробела
<code>islower</code>	проверяет, является ли символ символом в нижнем регистре
<code>isupper</code>	проверяет, является ли символ символом прописной буквы
<code>ispunct</code>	проверяет, является ли символ символом пунктуации
<code>tolower</code>	преобразует символ в нижний регистр
<code>toupper</code>	преобразует символ в верхний регистр

* Для широких символов доступны в заголовке `cwctype`. Функции для широких символов имеют имена с добавлением «w»: `iswalnum`, `towlower`

Таблица 18. Методы класса `string`

Метод	Описание
Доступ к элементам	
<code>at()</code>	Получение указанного символа с проверкой выхода индекса за границы
<code>front()</code>	Получение первого символа (ссылка)
<code>back()</code>	Получение последнего символа (ссылка)
<code>data()</code>	Возвращает указатель на лежащий в основе строки C-массив, такой,

Метод	Описание
	что <code>data() + i == &operator[](i)</code> для каждого <code>i</code> в <code>[0, size())</code>
Итераторы	
<code>begin()</code> <code>cbegin()</code>	Возвращает итератор на первый элемент. Возвращает константный итератор на первый элемент.
<code>end()</code> <code>cend()</code>	Возвращает итератор на элемент, следующий за последним. Возвращает константный итератор на элемент, следующий за последним.
	Использование: // заголовок <code>#include <iterator></code> // инициализация <code>auto first = stroka.cbegin();</code> <code>auto last = stroka.cend();</code> // в программе <code>while (first != last) {</code> <code>cout << *first;</code> <code>first++;</code> <code>}</code>
<code>rbegin()</code> <code>rend()</code> <code>crbegin()</code> <code>crend()</code>	Реверсивные итераторы. Первый возвращает реверсивный итератор на первый элемент развернутой в обратном направлении строки. Он указывает на последний символ в неразвернутой строке. Второй возвращает реверсивный итератор на символ, следующий за последним символом развернутой в обратном направлении строки. Он указывает на символ, предшествующий первому символу в неразвернутой строке. <code>crbegin()</code> и <code>crend()</code> возвращают константные реверсивные итераторы.
Вместимость	
<code>empty()</code>	Проверяет, является ли строка пустой
<code>size()</code> <code>length()</code>	Возвращает количество символов в строке
<code>max_size()</code>	Возвращает максимальное количество элементов, которое может содержать строка
<code>reserve()</code>	Задаёт ёмкость строки, минимум <code>size</code> . Новая память для хранения выделяется при необходимости.
<code>capacity()</code>	Возвращает количество символов, под которые у строки есть выделенное место.
<code>shrink_to_fit()</code>	Запрос для уменьшения ёмкости <code>capacity</code> до <code>size</code>
Операции	
<code>assign()</code>	Заменяет содержимое строки
	Использование: 1. <code>stroka.assign(count, ch);</code> /* заменить на <code>count</code> символов */

Метод	Описание
	<p>2. stroka.assign(s); /* stroka будет заменена на строку s; s – может быть как объектом класса string, так и C-строкой */</p> <p>3. stroka.assign(s, pos, count); /* заменяет содержимое подстрокой диапазона [pos, pos + count) строки s. Если запрашиваемая подстрока выходит за границы конца строки или если count == npos, диапазон возвращаемой подстроки будет [pos, size()). Если pos >= str.size(), будет сгенерировано исключение out_of_range */</p> <p>4. stroka.assign(first, last); /* заменяет содержимое данной строки копией символов диапазона [first, last); first и last – InputIt (итераторы вставки) */</p> <p>5. stroka.assign(ilst); /* заменяет содержимое данной строки содержимым списка инициализации ilist */</p>
clear()	Удаляет все символы строки. Выделенная память не будет освобождена, оставляя емкость capacity неизменной. Итераторы, указывающие за последний элемент строки остаются действительными.
insert()	<p>Вставка символов в строку</p> <p>Использование:</p> <ol style="list-style-type: none"> 1. stroka.insert(index, count, ch); /* вставка count символов ch в позицию index строки stroka */ 2. stroka.insert(index, s); /* вставляет строку s в позицию index строки stroka; s может быть как C-строка, так и объект класса string */ 3. stroka.insert(index, s, count); /* вставляет первые count символов из строки, на которую указывает s, в позицию index строки stroka. Строка s может содержать нулевые символы. */ 4. stroka.insert(index, s, index_s, count); /* вставляет подстроку s, полученную с помощью s.substr(index_s, count), в позицию index строки stroka */ 5. stroka.insert(it, ch); /* вставляет символ ch перед символом в позиции it итератора. Возвращает итератор вставки */ 6. stroka.insert(it, count, ch); /* тоже, но только count символов ch. Возвращает итератор вставки */ 7. stroka.insert(it, first, last); /* вставляет символы из диапазона [first, last) в позиции константного итератора it, first и last – итераторы вставки. Возвращает итератор вставки */ 8. stroka.insert(it, ilist); /* вставляет элементы из списка инициализации ilist, где it – константный итератор. Возвращает итератор вставки */
erase()	<p>Удаляет указанные символы из строки</p> <p>Использование:</p>

Метод	Описание
	<ol style="list-style-type: none"> 1. <code>stroka.erase(index, count);</code> /* удаляет count символов, начиная с позиции index */ 2. <code>stroka.erase(it);</code> /* удаляет символ в позиции it. Возвращает итератор, следующий за последним удаленным символом */ 3. <code>stroka.erase(first, last);</code> /* удаляет символы в диапазоне [first; last). Возвращает итератор, следующий за последним удаленным символом */
<code>push_back()</code>	Добавляет переданный символ к концу строки
<code>pop_back()</code>	Удаляет последний символ строки
<code>append()</code>	<p>Добавляет символы в конец строки</p> <p>Использование:</p> <ol style="list-style-type: none"> 1. <code>stroka.append(count, ch);</code> /* добавляет count символов ch */ 2. <code>stroka.append(s);</code> /* добавляет строку s; s может быть как C-строка, так и объект класса string */ 3. <code>stroka.append(s, index, count);</code> /* добавляет подстроку [index, index + count) из s. Если запрошенная подстрока выходит за границы конца строки, или если count == pos, диапазоном добавляемой подстроки будет [index, size()). Если index >= s.size(), будет сгенерировано исключение out_of_range */ 4. <code>stroka.append(s, count);</code> /* добавляет первые count символов из символьной строки, на которую указывает s. s может содержать нулевые символы. */ 5. <code>stroka.append(first, last);</code> /* добавляет символы в диапазоне [first, last) first и last итераторы вставки*/ 6. <code>stroka.append(ilst);</code> /* добавляет символы из списка инициализации ilist */
<code>replace()</code>	<p>Заменяет часть строки, указанную диапазоном [pos, pos + count) или [first, last), на новую строку</p> <p>Использование:</p> <ol style="list-style-type: none"> 1. <code>stroka.replace(pos, count, s);</code> /* заменить count символов строки stroka начиная с позиции pos строкой s */ 2. <code>stroka.replace(first, last, s);</code> /* заменить [first, last), на новую строку s; итераторы const_iterator */ 3. <code>stroka.replace(pos, count, s, pos2, count2);</code> /* замена подстрокой [pos2, pos2 + count2) из s */ 4. <code>stroka.replace(first, last, first2, last2);</code> /* или символами из диапазона [first2, last2); итераторы first и last константные, first2 и last2 – итераторы вставки */ 5. <code>stroka.replace(pos, count, cstr, count2);</code> /* первые count2 символов строки, на которую указывает cstr */ 6. <code>stroka.replace(first, last, cstr, count2);</code> /* тоже, но диапазон для замены [first, last); итераторы first и last

Метод	Описание
	<pre>const_iterator */ 7. stroka.replace(pos, count, cstr); /* С-строкой, на которую указывает cstr */ 8. stroka.replace(first, last, cstr); /* тоже, аналогично, итераторы const_iterator */ 9. stroka.replace(pos, count, count2, ch); /* заменить на count2 символов ch */ 10. stroka.replace(first, last, count2, ch); /* тоже, аналогично, итераторы const_iterator */ 11. stroka.replace(first, last, ilist); /* заменить на символы в списке инициализации ilist, итераторы const_iterator */</pre>
substr()	<p>Возвращает подстроку [pos, pos + count). Если запрашиваемая подстрока выходит за границы конца строки или если count == pos, диапазон возвращаемой подстроки будет [pos, size())</p> <p>Использование: stroka.substr(pos, count);</p>
copy()	<p>Копирует подстроку, заданную диапазоном [pos, pos + count), в строку символов, на которую указывает dest. Если запрашиваемая подстрока выходит за границы конца строки или если count == pos, диапазон копируемой подстроки будет [pos, size()). Результирующая строка не завершается нулевым символом. Если pos >= size(), будет сгенерировано исключение out_of_range</p> <p>Использование: stroka.copy(dest, count, pos);</p>
resize()	<p>Изменяет размер строки, чтобы она могла содержать count символов. Если текущий размер меньше, чем count, будут добавлены дополнительные символы. Если текущий размер больше, чем count, строка сокращается до первых count символов</p> <p>Использование: stroka.resize(count); или stroka.resize(count, ch);</p>
swap()	<p>Обменивает содержимое строки с содержимым other. Все итераторы и ссылки остаются действительными</p> <p>Использование: stroka.swap(other);</p>
Поиск	
find()	<p>Находит первую подстроку, равную переданной последовательности символов. Поиск начинается с позиции pos, т.е. найденная подстрока не может начинаться в позиции, предшествующей pos</p> <p>Использование:</p> <ol style="list-style-type: none"> stroka.find(s, pos); /* находит первую подстроку, равную s, начиная с позиции pos, где s – объект класса string */ stroka.find(s, pos, count); /* находит первую подстроку, равную первым count символам строки, на которую указывает s. s может включать нулевые символы. */ stroka.find(s, pos); /* находит первую подстроку, равную

Метод	Описание
	<p>символьной строке, на которую указывает s (C-строка). Длина строки определяется по первому вхождению нулевого символа. */</p> <p>4. <code>stroka.find(ch, pos);</code> /* находит первое вхождение символа ch в строку stroka */</p>
<code>rfind()</code>	<p>Находит последнюю подстроку, равную переданной символьной последовательности. Поиск начинается с позиции pos, т.е. в поиске участвует только подстрока в диапазоне [pos, size). Если в качестве позиции pos передано npos, поиск будет произведен по всей строке.</p> <p>Использование: <code>stroka.rfind(s), stroka.rfind(s, pos, count)</code> /* Находит последнюю подстроку, равную первым count символам строки, на которую указывает s. s может включать нулевые символы */ <code>stroka.rfind(ch)</code></p>
<code>find_first_of()</code>	<p>Находит первый символ строки, равный одному из символов в переданной последовательности символов. Поиск начинается с позиции pos, т.е. найденный символ не может находиться в позиции, предшествующей pos.</p> <p><code>stroka.find_first_of(basic_string& str, size_type pos = 0)</code> <code>stroka.find_first_of(CharT* s, size_type pos, size_type count)</code> <code>stroka.find_first_of(CharT* s, size_type pos = 0)</code> <code>stroka.find_first_of(CharT ch, size_type pos = 0)</code></p>
<code>find_last_of()</code>	<p>Находит последний символ, равный одному из символов в переданной последовательности символов. Поиск начинается с позиции pos, т.е. в поиске участвует только подстрока в диапазоне [0, pos]. Если в качестве позиции pos передано npos, поиск будет произведен по всей строке.</p> <p><code>stroka.find_last_of(basic_string& str, size_type pos = npos)</code> <code>stroka.find_last_of(CharT* s, size_type pos, size_type count)</code> <code>stroka.find_last_of(CharT* s, size_type pos = npos)</code> <code>stroka.find_last_of(CharT ch, size_type pos = npos)</code></p>
Константы	
<code>npos</code>	<p>Это специальное значение, равное максимальному значению, которое может предоставить тип. Точный смысл данного значения зависит от контекста, но, как правило, оно используется либо как индикатор конца строки в функциях, которые ожидают позицию символа, либо как индикатор ошибки в функциях, которые возвращают позицию в строке</p> <p>Использование: // функции поиска в строке возвращают npos, если требуемое значение не найдено string s = "test"; if(s.find('a') == string::npos) cout << "Символ 'a' отсутствует в строке 'test'\n"; // функции, которые принимают диапазон в качестве аргументов // используют npos в качестве индикатора "все до конца строки"</p>

Метод	Описание
	<pre>string s2(s, 2, string::npos); cout << s2 << endl; bitset<5> b("aaabb", string::npos, 'a', 'b'); cout << b << endl;</pre>
Функции, не являющиеся членами класса	
getline()	<p>Считывает неформатированные данные из потока в строку. Останавливается, как только найден символ, равный разделителю (delim), или исчерпан поток</p>
	<p>Использование:</p> <ol style="list-style-type: none"> 1. <code>getline(input, s, delim);</code> 2. <code>getline(input, s);</code> /* delim опускается, если символ разделитель – нулевой символ */ <p>Примечание: s – объект класса string</p>
stoi() stol() stoll()	<p>Извлекает знаковое целое число из строки s. Функция отбрасывает пробельные символы. Затем из строки извлекаются символы, необходимые для формирования корректного представления целого числа в системе счисления с основанием base и преобразуются в целочисленное значение. Индекс первого непреобразованного символа сохраняется в pos. Если в качестве pos передан NULL, параметр игнорируется</p> <p>Использование:</p> <pre>stoi(s, pos, base); // Например: string s = "45"; int myint = stoi(s); cout << myint << '\n';</pre>
stoul() stoull()	<p>Извлекает беззнаковое целое число из строки</p>
stof() stod() stold()	<p>Извлекает число с плавающей точкой из строки. Допустимое значение числа с плавающей точкой. Десятичное выражение числа с плавающей точкой состоит из следующих частей: знак плюс или минус, непустая последовательность десятичных цифр, которая может, в обязательном порядке, содержать десятичную часть, символ e или E, за которым следует обязательный знак минус или плюс и непустая последовательность десятичных цифр — экспоненту; выражение бесконечности: знак плюс или минус, INF или INFINITY без учета регистра; выражение NaN (Not-a-Number): знак плюс или минус, NAN без учета регистра символов.</p>
to_string()	<p>Преобразует целое число или число с плавающей точкой в объект класса string</p>

Таблица 19. Методы класса vector

Метод	Описание
Доступ к элементам	
<code>at()</code>	Операция доступа к элементу вектора по индексу. Возвращает ссылку на элемент. Выполняется проверка выхода индекса за границы диапазона вектора. В случае выхода за пределы диапазона генерирует исключение <code>out_of_range</code>
<code>[]</code>	Возвращает ссылку на элемент по индексу. Проверка на выход за границы не выполняется
<code>front()</code>	Возвращает ссылку на первый элемент в контейнере
<code>back()</code>	Возвращает ссылку на последний элемент в контейнере
Итераторы	
<code>begin()</code> <code>cbegin()</code>	Возвращает итератор на первый элемент контейнера. (<code>cbegin()</code> возвращает <code>const_iterator</code>)
<code>end()</code> <code>cend()</code>	Возвращает итератор на элемент, следующий за последним элементом контейнера. Этот элемент выступает в качестве заполнителя; попытка доступа к нему приводит к неопределенному поведению. (<code>cend()</code> возвращает <code>const_iterator</code>)
<code>rbegin()</code> <code>crbegin()</code> <code>rend()</code> <code>crend()</code>	Возвращают реверсивные итераторы на последний элемент массива и на элемент массива, предшествующий первому. Если итератор получен с помощью инструкции <code>iter = v.rbegin()</code> , то он будет смещаться к началу массива, если будет выполняться операция: <code>iter++</code> (пока <code>iter != v.rend()</code>).
Вместимость	
<code>empty()</code>	Проверяет есть ли элементы в контейнере. Возвращает <code>true</code> , если контейнер пуст
<code>size()</code>	Возвращает количество элементов в контейнере (размер массива). Возвращаемый тип соответствует типу <code>size_type</code> (машиннозависимый беззнаковый целый тип) Примечание: используйте в циклах вместо <code>unsigned</code> тип <code>size_t</code> , например: <code>for (size_t i = 0; i < v.size(); ++i) {...}</code>
<code>max_size()</code>	Возвращает максимальное количество элементов, которое может содержать вектор. Это значение обычно равно <code>numeric_limits<size_type>::max()</code> , и отражает теоретический предел на размер контейнера.
<code>reserve()</code>	Задаёт ёмкость контейнера по крайней мере до размера <code>size</code> : <code>reserve(size);</code> Новая память выделяется по необходимости.
<code>capacity()</code>	Возвращает количество элементов контейнера, для которых в данный момент выделена память.
<code>shrink_to_fit()</code>	Освобождение неиспользуемой памяти контейнера (сократить

Метод	Описание
	capacity до size)
Модификаторы и функция-член assign	
assign()	<p>Заменяет содержимое контейнера.</p> <p>Использование:</p> <ol style="list-style-type: none"> <code>v.assign(count, value);</code> /* заменяет count копиями, имеющих значения value */ <code>v.assign(first, last);</code> /* заменяет копиями из диапазона [first, last), где first и last - InputIterator */
clear()	<p>Удаляет все элементы из контейнера. Делает недействительными все ссылки, указатели или итераторы указывающие на удалённые элементы. Оставляет capacity() вектора без изменений.</p>
insert()	<p>Вставляет элементы в указанную позицию в контейнере.</p> <p>Использование:</p> <ol style="list-style-type: none"> <code>v.insert(pos, value);</code> <code>v.insert(pos, count, value);</code> <code>v.insert(pos, first, last);</code> <code>v.insert(pos, ilist);</code> <p>/* Возвращает итератор вставки. pos – const_iterator, элемент перед которым будет осуществляться вставка, count – количество копий, value – значение вставляемого элемента, first и last (InputIt) - диапазон элементов для вставки (не должны быть итераторами целевого контейнера), ilist - список инициализации для вставки */</p>
erase()	<p>Удаляет указанные элементы из контейнера.</p> <p>Использование:</p> <ol style="list-style-type: none"> <code>v.erase(pos);</code> /* Удаляет элемент в позиции pos */ <code>v.erase(first, last);</code> /* Удалить диапазон [first, last) */ <p>/* Возвращает итератор. pos, first и last - const_iterator */</p>
push_back()	<p>Добавляет элемент value в конец массива v:</p> <p><code>v.push_back(value);</code></p> <p>Если новый size() больше, чем capacity(), то все итераторы и указатели становятся недействительными.</p>
pop_back()	<p>Удаляет последний элемент из контейнера. Итераторы и указатели остаются в рабочем состоянии.</p>
resize()	<p>Изменяет размер контейнера v, чтобы содержать count элементов. Если текущий размер меньше, чем count, дополнительные элементы добавляются и инициализируются value. Если текущий размер больше count, контейнер сводится к его первым count элементам.</p> <p>Использование:</p> <ol style="list-style-type: none"> <code>v.resize(count);</code> <code>v.resize(count, value);</code>
swap()	<p>Обмен элементами между двумя контейнерами v и othe.</p> <p>Использование:</p> <p><code>v.swap(other);</code> /* othe – другой вектор */</p>

Метод	Описание
vector<bool>	
flip()	Инверсия битов. Используется в std::vector<bool> — компактной «версии» специализированного вектора типа bool, который во всем подобен обычному вектору. Если размер такого вектора известен в процессе компиляции, то ему доступны методы шаблонного класса <bitset>