Оглавление

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 ( Linux  в    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 alignof and and_eq asm auto bitand bitor bool break case catch char char16_t char32_t class compl const constexpr const_cast continue	decltype default delete do double dynamic_cast else enum explicit export extern false float for friend goto if inline int long mutable	namespace new noexcept not not_eq nullptr operator or or_eq private protected public register reinterpret_cast return short signed sizeof static static_assert static_cast	struct switch template this thread_local throw true try typedef typeid typename union unsigned using virtual void volatile wchar_t while xor 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	
3	`++,--`	Префиксный инкремент и декремент	Справа-налево
	`+,-`	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
15	`=`	Прямое присваивание (предоставляемое по умолчанию для C++ классов)	Справа-налево
	`+=, -=`	Присвоение с суммированием и разностью	
	`*=,/=,%=`	Присвоение с умножением, делением и остатком от деления	
	`<<=,>>=`	Присвоение с побитовым сдвигом влево и сдвигом вправо	
	`&=,^=,|=`	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", "\
	заключенная в кавычки	t3начение r = \0xF5\n"
Прочие	Нулевой указатель	*nullpt*
	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	2.225073858507201 e-308	1.797693134862316 e+308
long double, l	Расширенная, 19	10 байт	3.362103143112093506 3 e-4932	1.189731495357231765 e+4932

Таблица 9. Преобразование типов

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

Таблица 10. Некоторые функци cmath

Функция*	**Описание**
Округление	
round	Округляет число по правилам арифметики. Для возвращения целых значений используются префиксы `l` и `ll` (**lround, llround**)
floor	Округляет число вниз (“пол”), при этом `floor(1.5) == 1`, `floor(-1.5) == -2`
ceil	Округляет число вверх (“потолок”), при этом `ceil(1.5) == 2`, `ceil(-1.5) == -1`
trunc	Округление в сторону нуля (отбрасывание дробной части)
Функция*	**Описание**
abs	Модуль (абсолютная величина) целого числа (int). Или с префиксами **labs** и **llabs**
fabs	Модуль вещественного числа
modf	Разлагает на целую и дробную части (сохраняется по указателю: mod(x, &p))
Корни, степени, логарифмы	
sqrt	Квадратный корень. Использование: **sqrt(x)**
pow	Возведение в степень, возвращает a^b. Использование: **pow(a,b)** **
exp	Экспонента, возвращает e^x. Использование: **exp(x)**
log	Натуральный логарифм
log10	Десятичный логарифм
hypot	Гипотенуза **hypot(x,y)** **

Тригонометрия	
sin	Синус угла, задаваемого в радианах
cos	Косинус угла, задаваемого в радианах
tan	Тангенс угла, задаваемого в радианах
asin	Арксинус, возвращает значение в радианах
acos	Арккосинус, возвращает значение в радианах
atan	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
M_E	**e**	**M_1_PI**	**1/π**
M_LOG2E	**$\log_2(e)$**	**M_2_PI**	**2/π**
M_LOG10E	**$\log_{10}(e)$**	**M_2_SQRTPI**	**2/sqrt(π)**
M_LN2	**ln(2)**	**M_SQRT2**	**sqrt(2)**
M_LN10	**ln(10)**	**M_SQRT1_2**	**1/sqrt(2)**
M_PI	**π**	**M_PI_4**	**π/4**
M_PI_2	**π/2**	*необходимо подключить <cmath>	

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 страницы новая страница -	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 **\Unnnnnnnn**	Произвольное Юникод значение - . Результатом могут быть несколько символов .	кодовая позиция **U+nnnn** **U+nnnnn**

Таблица 13. Основные конструкции языка C++

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

Инструкция цикла со счетчиком	Шаблон

Цикл с параметром **for: for** (инициализация условие модификация; ;)
{
// тело цикла
}
Бесконечный цикл :
for (;;) {...}
Цикл **for** основанный на диапазоне
: for (auto &r : arr) {...}

template <typename T1, typename T2, ...>
// Определение функции

Функции

Объявление прототип **( ):**
void myFunc1(*типыпараметров***); type myFunc2(***типыпараметров***);**

Вызов:
myFunc1(*аргументы***); cout << myFunc2(***аргументы***);**

Описание: **void myFunc1(***параметры***) {**
// Тело функции
} type myFunc2(*параметры***) {**
// Тело функции
}

С-массивы

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];
}

Структуры

Определение структуры
: struct structName { type atrib;
// остальные элементы структуры
}; // точка с запятой обязательна **—**
// объявление переменных абстрактного // типа: structName structVar1, structVar2, ...;

Доступ к элементам структуры

Указатель на переменную структуры
structName: structName *structPtr = &structVar1;
доступ с помощью операции **«.»**:
structVar1.atrib доступ с помощью операции **«->»:**
structPtr -> atrib
что равносильно **(*structPtr).atrib**

Классы

Исключения

```
class Имякласса_  {
private:
    // Закрытые члены класса
public:
    // Открытые члены класса
// Конструктор
    _Имя класса (параметры) {
    // Инициализация полей    ;
    }
    // Или списком инициализации
    _Имя класса (параметры) :
var1 (инициализатор),          var2
(инициализатор)
        //…
        {}
};
```

```
try {
    // Блок контролируемого кода

}
catch (exception1) {
    // Будет выполняться этот блок кода
                          ,     // если
перехвачено исключение типа
       exception1
}
catch (exception2) {
    // Блок обработки исключения
       exception2
}
catch (exception3) {
    // Блок обработки исключения
       exception3     throw;
}
catch ( ... ) {
    // Обрабатывается исключение
    // любого другого типа
      , }
```

Функтор	Лямбда-выражение

```
struct Имяфунктора_ {    //
Поля структуры
    //    Конструктор    если    необходим,
void operator() () {
    // Инструкции;
    }
}; // Или
class   Имяфунктора_   {
//      Поля      класса
public:
    //    Конструктор    если    необходим,
void operator() () {
    //Инструкции;
    }
};
```

```
Полная версия     :
[списокзахвата   ] (списокпараметров
     ) mutable exception attribute ->
возвращаемыйтип  {телофункции     }
или сокращенная   :
[списокзахвата   ] (списокпараметров
     ) {телофункции   } 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
```

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 учолчанию)
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	**iostream**	Выравнивание по левой границе
right	**iostream**	Выравнивание по правой границе (по умолчанию)
boolalpha	**iostream**	Вывод логических величин в текстовом виде
Манипулятор*	**Заголовок**	**Описание**
noboolalpha	**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
nouppercase	**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)	**iomanip**	Установить символ с 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
`cout.fill(char)`	устанавливает символ заполнитель
`cout.width(int)`	задает ширину поля
`cout.precision(int)`	задает количество знаков после десятичной точки

Таблица 17. Функции заголовка cctype

Функция*	Описание
isalnum	проверяет, является ли символ буквенно-цифровым
isalpha	проверяет, является ли символ буквенным
isdigit	проверяет, является ли символ цифрой
isxdigit	проверяет, является ли символ шестнадцатеричной цифрой
iscntrl	проверяет, является ли символ управляющим символом
isspace	проверяет, является ли символ символом пробела
islower	проверяет, является ли символ символом в нижнем регистре
isupper	проверяет, является ли символ символом прописной буквы
ispunct	проверяет, является ли символ символом пунктуации
tolower	преобразует символ в нижний регистр
toupper	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	
at()	Получение указанного символа с проверкой выхода индекса за границы
front()	Получение первого символа (ссылка)
back()	Получение последнего символа (ссылка)
data()	Возвращает указатель на лежащий в основе строки С-массив, такой,

Метод	Описание
	что `data() + i == &operator[](i)` для каждого `i` в `[0, size()]`
Итераторы	
begin() **cbegin()**	Возвращает итератор на первый элемент. Возвращает константный итератор на первый элемент.
end() **cend()**	Возвращает итератор на элемент, следующий за последним. Возвращает константный итератор на элемент, следующий за последним.

	Использование:
	`// заголовок`
	`#include <iterator>`
	`// инициализация`
	`auto first = stroka.cbegin();`
	`auto last = stroka.cend();`
	`// в программе while`
	`(first != last) {`
	`cout << *first;`
	`first++;`
	`}`

rbegin() **rend()** **crbegin()** **crend()**	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. crbegin() и crend() возвращают константные реверсивные итераторы.
Вместимость	
empty()	Проверяет, является ли строка пустой
size() **length()**	Возвращает количество символов в строке

max_size()	Возвращает максимальное количество элементов, которое может содержать строка
reserve()	Задает емкость строки, минимум `size`. Новая память для хранения выделяется при необходимости.
capacity()	Возвращает количество символов, под которые у строки есть выделенное место.
shrink_to_fit()	Запрос для уменьшения емкости `capacity` до `size`
Операции	
assign()	Заменяет содержимое строки
	Использование: 1. `stroka.assign(count, ch); /* заменить на count символов */`

Метод	Описание
	2. `stroka.assign(s); /* stroka будет заменена на строку s; s — может быть как объектом класса string, так и С-строкой */` 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 */` 4. `stroka.assign(first, last); /* заменяет содержимое данной строки копией символов диапазона [first, last); first и last — InputIt (итераторы вставки) */` 5. `stroka.assign(ilist); /* заменяет содержимое данной строки содержимым списка инициализации ilist */`
clear()	Удаляет все символы строки. Выделенная память не будет освобождена, оставляя емкость `capacity` неизменной. Итераторы, указывающие за последний элемент строки остаются действительными.
insert()	Вставка символов в строку

	Использование: 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 С-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()	Удаляет указанные символы из строки
	Использование:

Метод	Описание
	1. **stroka.erase(index, count);** /* удаляет count символов, начиная с позиции index */ 2. **stroka.erase(it);** /* удаляет символ в позиции it. Возвращает итератор, следующий за последним удаленным символом */ 3. **stroka.erase(first, last);** /* удаляет символы в диапазоне [first; last). Возвращает итератор, следующий за последним удаленным символом */
push_back()	Добавляет переданный символ к концу строки
pop_back()	Удаляет последний символ строки
append()	Добавляет символы в конец строки

	Использование: 1. **stroka.append(count, ch);** /* добавляет count символов ch */ 2. **stroka.append(s);** /* добавляет строку s; s может быть как C-строка, так и объект класса string */ 3. **stroka.append(s, index, count);** /* добавляет подстроку [index, index + count) из s. Если запрошенная подстрока выходит за границы конца строки, или если count == npos, диапазоном добавляемой подстроки будет [index, size()). Если index >= s.size(), будет сгенерировано исключение out_of_range */ 4. **stroka.append(s, count);** /* добавляет первые count символов из символьной строки, на которую указывает s. s может содержать нулевые символы. */ 5. **stroka.append(first, last);** /* добавляет символы в диапазоне [first, last) first и last итераторы вставки*/ 6. **stroka.append(ilist);** /* добавляет символы из списка инициализации ilist */
replace()	Заменяет часть строки, указанную диапазоном [pos, pos + count) или [first, last), на новую строку
	Использование: 1. **stroka.replace(pos, count, s);** /* заменить count символов строки stroka начиная с позиции pos строкой s */ 2. **stroka.replace(first, last, s);** /* заменить [first, last), на новую строку s; итераторы const_iterator */ 3. **stroka.replace(pos, count, s, pos2, count2);** /* замена подстрокой [pos2, pos2 + count2) из s */ 4. **stroka.replace(first, last, first2, last2);** /* или символами из диапазона [first2, last2); итераторы first и last константные, first2 и last2 — итераторы вставки */ 5. **stroka.replace(pos, count, cstr, count2);** /* первые count2 символов строки, на которую указывает cstr */ 6. **stroka.replace(first, last, cstr, count2);** /* тоже, но диапазон для замены [first, last); итераторы first и last

Метод	Описание

	const_iterator */ 7. **stroka.replace(pos, count, cstr);** /* C-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 */
substr()	Возвращает подстроку [pos, pos + count). Если запрашиваемая подстрока выходит за границы конца строки или если count == npos, диапазон возвращаемой подстроки будет [pos, size())
	Использование: **stroka.substr(pos, count);**
copy()	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 == npos, диапазон копируемой подстроки будет [pos, size()). Результирующая строка не завершается нулевым символом. Если pos >= size(), будет сгенерировано исключение out_of_range
	Использование: **stroka.copy(dest, count, pos);**
resize()	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
	Использование: **stroka.resize(count);** или **stroka.resize(count, ch);**
swap()	Обменивает содержимое строки с содержимым other. Все итераторы и ссылки остаются действительными
	Использование: **stroka.swap(other);**
Поиск	
find()	Находит первую подстроку, равную переданной последовательности символов. Поиск начинается с позиции pos, т.е. найденная подстрока не может начинаться в позиции, предшествующей pos

	Использование: 1. **stroka.find(s, pos);** /* находит первую подстроку, равную s, начиная с позиции pos, где s — объект класса string */ 2. **stroka.find(s, pos, count);** /* находит первую подстроку, равную первым count символам строки, на которую указывает s. s может включать нулевые символы. */ 3. **stroka.find(s, pos);** /* находит первую подстроку, равную

Метод	Описание
	символьной строке, на которую указывает s (С-строка). Длина строки определяется по первому вхождению нулевого символа. */ 4. **stroka.find(ch, pos);** /* находит первое вхождение символа ch в строку stroka */
rfind()	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.
	Использование: **stroka.rfind(s), stroka.rfind(s, pos, count)** /* Находит последнюю подстроку, равную первым count символам строки, на которую указывает s. s может включать нулевые символы */, **stroka.rfind(ch)**
find_first_of()	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.
	stroka.find_first_of(basic_string& str, size_type pos = 0) **stroka.find_first_of(CharT* s, size_type pos, size_type count)** **stroka.find_first_of(CharT* s, size_type pos = 0)** **stroka.find_first_of(CharT ch, size_type pos = 0)**
find_last_of()	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.
	stroka.find_last_of(basic_string& str, size_type pos = npos) **stroka.find_last_of(CharT* s, size_type pos, size_type count)** **stroka.find_last_of(CharT* s, size_type pos = npos)** **stroka.find_last_of(CharT ch, size_type pos = npos)**
Константы	

npos	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

	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"`

Метод	Описание
	`string` `s2(s, 2,` **`string`**`::`***`npos`***`);` **`cout`** `<< s2 <<` **`endl`**`;` **`bitset`**`<5> b("aaabb",` **`string`**`::`**`npos`**`, 'a', 'b');` **`cout`** `<< b <<` **`endl`**`;`
Функции, не являющиеся членами класса	
`getline()`	Считывает неформатированные данные из потока в строку. Останавливается, как только найден символ, равный разделителю (delim), или исчерпан поток
	Использование: 1. **`getline(input, s, delim);`** 2. **`getline(input, s);`** `/* delim опускается, если символ разделитель — нулевой символ */` **`Примечание:`** `s — объект класса string`
`stoi()` **`stol()`** **`stoll()`**	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

	Использование: **`stoi(s, pos, base);`**
	`// Например:` **`string`** `s = "45";`
	`int` `myint =` **`stoi`**`(s);` **`cout`** `<<`
	`myint << '\n';`

`stoul()` **`stoull()`**	Извлекает беззнаковое целое число из строки

stof() **stod()** **stold()**	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 необязательном порядке, содержать десятичную часть, символ `e` или `E`, за которым следует необязательный 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.
to_string()	Преобразует целое число или число с плавающей точкой в объект класса `string`

Таблица 19. Методы класса vector

Метод	Описание
Доступ к элементам	
at()	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 `out_of_range`
[]	Возвращает ссылку на элемент по индексу. Проверка на выход за границы не выполняется
front()	Возвращает ссылку на первый элемент в контейнере
back()	Возвращает ссылку на последний элемент в контейнере
Итераторы	
begin() **cbegin()**	Возвращает итератор на первый элемент контейнера. (`cbegin()` возвращает `const_iterator`)
end() **cend()**	Возвращает итератор на элемент, следующий за последним элементом контейнера. Этот элемент выступает в качестве заполнителя; попытка доступа к нему приводит к неопределенному поведению. (`cend()` возвращает `const_iterator`)
rbegin() **crbegin()** **rend()** **crend()**	Возвращают реверсивные итераторы на последний элемент массива и на элемент массива, предшествующий первому. Если итератор получен с помощью инструкции `iter = v.rbegin()`, то он будет смещаться к началу массива, если будет выполняться операция: `iter++` (пока `iter != v.rend()`).
Вместимость	
empty()	Проверяет есть ли элементы в контейнере. Возвращает true, если контейнер пуст

size()	Возвращает количество элементов в контейнере (размер массива). Возвращаемый тип соответствует типу `size_type` (машиннозависимый беззнаковый целый тип) Примечание: используйте в циклах вместо unsigned тип size_t, например: **`for (size_t i = 0; i < v.size(); ++i) {…}`**
max_size()	Возвращает максимальное количество элементов, которое может содержать вектор. Это значение обычно равно `numeric_limits<size_type>::max()`, и отражает теоретический предел на размер контейнера.
reserve()	Задает емкость контейнера по крайней мере до размера `size`: **`reserve(size);`** Новая память выделяется по необходимости.
capacity()	Возвращает количество элементов контейнера, для которых в данный момент выделена память.
shrink_to_fit()	Освобождение неиспользуемой памяти контейнера (сократить

Метод	Описание
	`capacity` до `size`)
Модификаторы и функция-член assign	
assign()	`Заменяет содержимое контейнера .` Использование: 1. **`v.assign(count, value);`** `/* заменяет count копиями, имеющих значения value */` 2. **`v.assign(first, last);`** `/* заменяет копиями из диапазона [first, last), где first и last - InputIterator */`
clear()	Удаляет все элементы из контейнера. Делает недействительными все ссылки, указатели или итераторы указывающие на удалённые элементы. Оставляет `capacity()` вектора без изменений.
insert()	Вставляет элементы в указанную позицию в контейнере. Использование: 1. **`v.insert(pos, value);`** 2. **`v.insert(pos, count, value);`** 3. **`v.insert(pos, first, last);`** 4. **`v.insert(pos, ilist);`** `/* 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 */`

erase()	Удаляет указанные элементы из контейнера. Использование: 1. **v.erase(pos);** /* Удаляет элемент в позиции pos */ 2. **v.erase(first, last);** /* Удалить диапазон [first, last) */ /* Возвращает итератор. pos, first и last - const_iterator */
push_back()	Добавляет элемент value в конец массива v: **v.push_back(value);** Если новый size() больше, чем capacity(), то все итераторы и указатели становятся недействительными.
pop_back()	Удаляет последний элемент из контейнера. Итераторы и указатели остаются в рабочем состоянии.
resize()	Изменяет размер контейнера v, чтобы содержать count элементов. Если текущий размер меньше, чем count, дополнительные элементы добавляются и инициализируется value. Если текущий размер больше count, контейнер сводится к его первым count элементам. Использование: **1. v.resize(count);** **2. v.resize(count, value);**
swap()	Обмен элементами между двумя контейнерами v и othe. Использование: **v.swap(other); /* othe — другой вектор */**

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>