Муниципальное бюджетное общеобразовательное учреждение
гимназия №14 имени первого летчика-космонавта Юрия Алексеевича Гагарина города Ейска
муниципального образования Ейский район
Краснодарского края

КРАТКИЙ СПРАВОЧНИК ПО ЯЗЫКУ

для кружка

C++ Programmer

и учащихся старших классов

Автор составитель: учитель информатики и ИКТ Фомин А. Т.
Дата последнего изменения: 18.02.17
Источники: http://inf-w.ru/?page_id=996

г. Ейск
2017 г.

Структура программы. Первая программа

```
//============================================================
// Description : Hello World in C++, Ansi-style
//============================================================
// Директивы с указанием заголовочных файлов
#include <iostream>
#include <string>
// Директива использования имен стандартной библиотеки
using namespace std;
// Главная функция программы
int main() {
	// Описание переменных
	int t;
	string s;
	//====================//
	// Консольный диалог  //
	//====================//
	cout << "Как вас зовут?\n";
	getline(cin, s);
	cout << "Сколько вам лет?\n";
	cin >> t;
	// Добавление объектов в поток
	cout << "Вас зовут "
		<< s
		<< ",\nВам "
		<< t
		<< " лет."
		<< endl;
	// Признак успешного завершения работы программы
	return 0;
}
```

Локализация в стиле C++. Строки широких символов

```
#include <iostream>
#include <locale>

using namespace std;

int main ()
{
	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 1. 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 2. 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, заключенная в кавычки	`"Здесь был Vasia", "\t3начение r = \ 0xF5\n"`
Прочие	Нулевой указатель	`nullpt`
	User-defined (определяемые пользователем)	`90.0_deg, "SDD"_prt`
	Параметр пакета	`...`

Таблица 3. 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	
10	`&`	Побитовое И	
11	`^`	Побитовый XOR (исключающее или)	
12	`|`	Побитовое ИЛИ (inclusive or)	
13	`&&`	Логическое И	
14	`||`	Логическое ИЛИ	
15	`?:`	Тернарное условие	Справа-налево
	`=`	Прямое присваивание (предоставляемое по	

Приоритет	Операция	Описание	Ассоциативность
15		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 4. Целочисленные типы

Название	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`	4 байта	Беззнаковый	`unsigned int`
`long`	4 байта	Знаковый	`long int, signed long, signed long int`
`unsigned long`	4 байта	Беззнаковый	`unsigned long int`
`long long`	8 байт	Знаковый	`long long int, signed long long, signed long long int`
`unsigned long long`	8 байт	Беззнаковый	`unsigned long long int`

Таблица 5. Таблица значений, которые могут принимать различные типы

Название	Размер	Минимальное значение	Максимальное значение
`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
`unsigned long long`	64 бита	0	2^{64}-1= 18446744073709551615

Таблица 6. Действительные типы

Тип	Точность	Размер	Количество знаков мантиссы	Минимальное положительное значение	Максимальное значение
float	Одинарная	4 байта	7	1.4e-45	3.4e38
double	Двойная	8 байт	15	5.0e-324	1.7e308
long double	Расширенная	10 байт	19	1.9e-4951	1.1e4932

Таблица 7. 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 8. 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)
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)**
cbrt	Кубический корень. Использование: **cbrt(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	

Функция	Описание
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 9. 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 10. 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 11. Основные конструкции языка C++

Ввод/вывод

```
// Вывод на строки дисплей:
cout << "Здравствуй, мир!" << endl;
// Ввод данных с клавиатуры:
cin >> var1 >> var2 >> varN;
```

Условная инструкция if

```
if (условие) {
      // Инструкции, если условие true
}
else { // необязательная часть
       // Инструкции, если условие false
}
```

Инструкции циклов с условием

```
// Цикл с предусловием:
while (условие) {
      // тело цикла;
}
// Цикл с постусловием:
do {  // тело цикла;
} while (условие);
// Бесконечный цикл: while (true) {…}
```

Инструкция выбора switch

```
switch (управляющее выражение) {
      case const_1: инструкции_1;
      break;
      case const_2: инструкции_2;
      break;
      ...
      default: инструкции_по_умолчанию;
}
```

Инструкция цикла со счетчиком

```
// Цикл с параметром for:
for (инициализация; условие; модификация){
      // тело цикла
} // Бесконечный цикл: for (;;) {…}
// range-based for
for (auto &r : arr) {...}
```

Шаблон

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

Функции

```
// Объявление (прототип):
void myFunc1(типы параметров);
type myFunc2(типы параметров);

// 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 посчитает 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
```

Функтор

```
struct Имя_функтора {
    // Поля структуры
    // Конструктор, если необходим
    void operator() () {
    // Инструкции;
```

Лямбда-выражение

```
Полная версия:
[список захвата] (список параметров)
mutable exception attribute -> возвращаемый
тип {тело функции}
или сокращенная:
```

`    }`
`};`
`// Или`
`class *Имя_функтора* {`
`    // Поля класса`
`    public:`
`    // Конструктор, если необходим`
`    void operator() () {`
`    //Инструкции;`
`    }`
`};`
	`[`*список захвата*`] (`*список параметров*`)`
	`{`*тело функции*`}`
	`Список захвата:`
	`[a,&b] *a* захвачена по значению, а *b* захвачена по ссылке.`
	`[this] захватывает указатель this по значению.`
	`[&] захват всех объектов по ссылке`
	`[=] захват всех объектов по значению`
	`[] ничего не захватывать`

Таблица 12. 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)
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 13. 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
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**	Вставлеяет символ '\n' и очищает входной поток
ends	**iostream**	Вставлеяет символ '\0' во входной поток
flush	**iostream**	Очищает входной поток

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

Таблица 14. Форматирующие функции-члены

Еще одним способом форматирования вывода являются три форматирующие функции-члены:

cout.fill(char); — устанавливает символ заполнитель

```
cout.width(int);       — задает ширину поля
cout.precision(int);   — задает количество знаков после десятичной точки
```

Таблица 15: Сокращенная форма логического выражения

Использование операций сравнения	Без использования операций сравнения	Описание
`a == 0`	`!a`	Равенство 0 (истинно, если a == 0)
`a != 0`	`a`	Проверка на неравенство 0
`a % 2 == 0`	`!(a % 2)`	Проверка на четность Истинно, если a - четное
`a & 1 == 0`	`!(a & 1)`	
`a % 2 == 1` (или `a % 2 != 0`)	`a % 2`	Проверка на нечетность Истинно, если a - нечетное
`a & 1 == 1`	`a & 1`	

Таблица 16. 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 17. 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)

Метод	Описание
data()	Возвращает указатель на лежащий в основе строки C-массив, такой, 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 */ 2. **stroka.assign(s)**; /* stroka будет заменена на строку s; s – может быть как объектом класса string, так и C-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

Метод	Описание
	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: 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 С-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

Метод	Описание
	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 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 */
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. */

Метод	Описание
	3. **stroka.find(s, pos)**; /* находит первую подстроку, равную символьной строке, на которую указывает s (C-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)
	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)**
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"`
	`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

Метод	Описание
	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 18. 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.

Метод	Описание
shrink_to_fit()	Освобождение неиспользуемой памяти контейнера (сократить 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
emplace()	Вставляет новый элемент перед позицией pos (const_iterator). Элемент построен на месте, без использования операций копирования или перемещения. **emplace(pos, Args&&... args);** Если новый size() больше, чем capacity(), все итераторы и указатели становятся нерабочими. В противном случае, нерабочими становятся только итераторы и указатели на элементы, идущие после вставленных.
emplace_back()	Добавляет новый элемент в конец контейнера. Элемент построен на месте, без использования операций копирования или перемещения. Конструктор элемента вызывается с точно такими же аргументами, что и передающиеся в функцию. **emplace_back(Args&&... args);** Если новый size() больше, чем capacity(), все итераторы и указатели становятся нерабочими. В противном случае, все они остаются в рабочем состоянии.
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.

Метод	Описание

	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 */`
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>

Таблица 19. Функции библиотеки Algorithm

Немодифицирующие операции	

for_each()	Унарная функция. По порядку применяет заданный функциональный объект f к результату разыменования каждого итератора (InputIterator) в диапазоне [first, last):
	`for_each(first, last, f);`
	Пример:
	`struct Sum {`
	`  Sum() { sum = 0; }`
	`  void operator()(int n) { sum += n; }`
	`  int sum;`
	`};`
	`// В главной программе:`
	`  vector<int> nums{3, 4, 2, 9, 15, 267};`
	`  Sum s = for_each(nums.begin(), nums.end(), Sum());`

count()
count_if()
	Возвращает количество элементов в диапазоне [first, last) удовлетворяющих определенному условию, где first и last - InputIterator. Первый вариант подсчитывает элементы, равные value, второй вариант подсчитывает элементы, для которых предикат p возвращает значение true:
	`1. count(first, last, value);`
	`2. count_if(first, last, p);`
	Пример:
	`int data[] = { 1, 2, 3, 4, 4, 3, 7, 8, 9, 10 };`
	`std::vector<int> v(data, data+10);`
	`int target1 = 3;`
	`int target2 = 5;`
	`int num_items1 = std::count(v.begin(), v.end(), target1);`
	`int num_items2 = std::count(v.begin(), v.end(), target2);`

equal()	Возвращает true, если элементы одинаковы в двух диапазонах: одном, определяемом [first1, last1), и другом, начинающемся с first2, где first1, first2, last1 и last2 - InputIterator. Первый вариант функции использует operator == для сравнения элементов, второй вариант использует заданный бинарный предикат p:
	`1. equal(first1, last1, first2);`
	`2. equal(first1, last1, first2, p);`
	Пример:
	`void test(const string& s)`
	`{`
	`  if(equal(s.begin(), s.begin() + s.size() / 2, s.rbegin())) {`
	`  cout << "\"" << s << "\" - палиндром\n";`

	} **else** **{** **cout** **<<** **"\""** **<<** **s** **<<** **"\"** **не палиндром\n";** **}** **}**
find() **find_if()** **find_if_not()**	Эти функции находят в диапазоне [first, last) (где first и last - InputIterator) первый элемент, удовлетворяющий определённым условиям: find ищет элемент, равный value find_if ищет элемент, для которого предикат p возвращает значение true find_if_not ищет элемент, для которого предикат q возвращает значение false: 1. **find(first, last, value);** 2. **find_if(first, last, p);** 3. **find_if_not(first, last, q);** Пример: **int n = 3;** **vector<int> v{0, 1, 2, 3, 4};** **auto result = find(v.begin(), v.end(), n);** **if (result != v.end()) {** **cout << "v содержит: " << n << '\n';** **} else {** **cout << "v не содержит: " << n << '\n';** **}**
find_end()	Ищет последнее вхождение подпоследовательности элементов [s_first, s_last) в диапазон [first, last). Итераторы — ForwardIterator. Первый вариант использует operator == для сравнения элементов, второй вариант использует заданный бинарный предикат p: 1. **find_end(first, last, s_first, s_last);** 2. **find_end(first, last, s_first, s_last, p);** Пример: **vector<int> v{1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4};** **vector<int>::iterator result;** **vector<int> t{1, 2, 3};** **result = std::find_end(v.begin(), v.end(), t.begin(), t.end());** **if (result == v.end()) {** **cout << "подпоследовательность не найдена\n";** **} else {** **cout << "последнее вхождение в позиции: "** **<< distance(v.begin(), result) << "\n";** **}**
search()	Ищет первое вхождение последовательности элементов [s_first, s_last) в диапазон [first, last - (s_last - s_first)). Итераторы — ForwardIterator. Первый вариант использует operator == для сравнения элементов, второй вариант использует заданный бинарный предикат p: 1. **search(first, last, s_first, s_last);** 2. **search(first, last, s_first, s_last, p);** Пример: **template<typename Container>** **bool in_quote(const Container& cont, const string& s)** **{** **return search(cont.begin(), cont.end(), s.begin(), s.end()) != cont.end();** **}** **int main()** **{** **string str = "Зачем тратить время на обучение,** **если невежество приходит мгновенно?";** **cout << boolalpha << in_quote(str, "обучение") << '\n'** **<< in_quote(str, "обручение") << '\n';** **vector<char> vec(str.begin(), str.end());**

	`cout << std::boolalpha << in_quote(vec, "обучение") << '\n'`
	`<< in_quote(vec, "обручение") << '\n';`
	`}`

Модифицирующие операции	
copy() **copy_if()**	Копирует элементы диапазона [first, last) в диапазон, начинающийся с d_first. Второй вариант копирует только те элементы, для которых предикат pred возвращает true. Возвращает OutputIterator, first и last - InputIterator, a d_first – OutputIterator: 1. **copy(first, last, d_first);** 2. **copy_if(first, last, d_first, pred);** Пример: **vector<int> from_vector;** **for (int i = 0; i < 10; i++) {** **from_vector.push_back(i);** **}** **vector<int> to_vector(10);** **copy(from_vector.begin(), from_vector.end(), to_vector.begin());** **cout << "to_vector содержит: ";** **copy(to_vector.begin(), to_vector.end(),** **ostream_iterator<int>(cout, " "));** **cout << endl;**
copy_n()	Копирует ровно count значений из диапазона, начинающегося с first (InputIt) в диапазон, начинающийся с result (OutputIt). Возвращает OutputIt – итератор в целевом диапазоне, указывающий на элемент, следующий за последним скопированным если count > 0, иначе – first: **copy_n(first, count, result);**
fill()	Присваивает заданное значение value всем элементам диапазона [first, last), итераторы - ForwardIterator: **fill(first, last, value);**
fill_n()	Присваивает заданное значение value первым count элементам в диапазоне, начинающемся с first (OutputIterator), возвращает OutputIterator: **fill_n(first, count, value);**
remove() **remove_if()**	Удаляет из диапазона [first, last) все элементы, удовлетворяющие определенному условию. Первый вариант удаляет все элементы, равные value, второй вариант удаляет все элементы, для которых предикат p возвращает true: **remove(first, last, value);** **remove_if(first, last, p);** Удаление осуществляется путём сдвига элементов внутри диапазона таким образом, что удаляемые элементы перезаписываются. Элементы между старым и новым концами диапазона имеют неопределённое значение. Возвращается итератор (ForwardIterator) на новый конец диапазона. Относительный порядок оставшихся элементов сохраняется.
replace() **replace_if()**	Заменяет все элементы в диапазоне [first, last), удовлетворяющие определенному условию, на new_value. Итераторы ForwardIterator. Первый вариант заменяет элементы, равные old_value, второй вариант заменяет элементы, для которых предикат p возвращает true: 1. **replace(first, last, old_value, new_value);** 2. **replace_if(first, last, p, new_value);** Пример **array<int, 10> s{5, 7, 4, 2, 8, 6, 1, 9, 0, 3};** **replace(s.begin(), s.end(), 8, 88);** **for (int a : s) {** **cout << a << " ";**

	`}`
	`cout << '\n';`
	`replace_if(s.begin(), s.end(), bind(less<int>(), std::placeholders::_1, 5), 55);`

swap()	Обмен переданных значений. Определено в заголовочном файле <utility> 1) Меняет местами значения a и b. 2) Обмен массивов a и b. `swap(a, b);`

swap_ranges()	Обмен элементов между диапазоном [first1, last1) и другим диапазоном, который начинается с first2. Количество элементов в этих двух диапазонах должно совпадать. Возвращает итератор, указывающий на элемент, после последнего во втором диапазоне, начинающимся с first2. Итераторы — ForwardIterator.
	Пример:
	`vector<int> v = {1, 2, 3, 4, 5};`
	`list<int> l = {-1, -2, -3, -4, -5};`
	`swap_ranges(v.begin(), v.begin() + 3, l.begin());` /* поменяются местами три элемента от начала */

iter_swap()	Меняет местами значения элементов, на которые указывают два итератора (ForwardIterator).
reverse()	Меняет порядок следования элементов в диапазоне [first, last) на противоположный. Итераторы — BidirectionalIterator `reverse(first, last);`

reverse_copy()	Копирует элементы из диапазона [first, last) в диапазон, начинающийся с d_first, таким образом, что элементы в новом диапазоне располагаются в обратном порядке. Возвращает output-итератор на элемент, следующий за последним скопированным. First и last — BidirectionalIterator:
	`reverse_copy(first, last, d_first);`
	Пример:
	`vector<int> v({1,2,3});`
	`for_each(begin(v), end(v), [&](int value){cout << value << " "; });`
	`cout << endl;`
	`vector<int> destiny(3);`
	`reverse_copy(begin(v), end(v), begin(destiny));`
	`for_each(begin(destiny), end(destiny), [&](int value){cout << value << " "; });`
	`cout << endl;`

rotate()	Меняет местами элементы в диапазоне [first, last) таким образом, что элемент n_first становится первым в новом диапазоне, а n_first-1 — последним: `rotate(first, n_first, last);` Возвращается итератор (ForwardIterator), равный first + (last - n_first)

unique()	Удаляет все последовательно повторяющиеся элементы из диапазона [first, last) и возвращает итератор на элемент, следующий за последним элементом нового диапазона. Первая версия использует operator== для сравнения элементов, вторая версия использует предоставленный бинарный предикат p:
	`unique(first, last);`
	`unique(first, last, p);`
	Удаление производится путем сдвига диапазона таким образом, что элементы которые должны быть удалены будут перезаписаны. Относительный порядок элементов сохраняется , а размер контейнера не изменяется. Итераторы, указывающие на элементы после нового диапазона становятся недействительными, а значения этих элементов становятся неопределёнными. Итераторы: ForwardIterator.

unique_copy()	Копирует элементы из диапазона [first, last), в другой диапазон с

	началом в d_first таким образом, что в нем не будет последовательных одинаковых элементов. Итераторы InputIterator, OutputIterator **unique_copy(first, last, d_first);** **unique_copy(first, last, d_first, p);**
Операции сортировки	
sort()	Сортировка элементов в диапазоне [first, last) в порядке возрастания. Итераторы — RandomAccessIterator. Первый вариант использует для сравнения элементов operator<, вторая версия использует функтор comp: 1. **sort(first, last);** 2. **sort(first, last, comp);** Пример: **array<int, 10> s{5, 7, 4, 2, 8, 6, 1, 9, 0, 3};** **sort(s.begin(), s.end());** **for (int a : s) {** **cout << a << " "; }** **cout << '\n';** **sort(s.begin(), s.end(), greater<int>()); /* Сортировка по убыванию*/** **for (int a : s) {** **cout << a << " "; }** **cout << endl;**
partial_sort()	Сортировка элементов в диапазоне [first, last), но упорядочены (в порядке возрастания) будут только элементы в диапазоне [first, middle). Итераторы — RandomAccessIterator. Первый вариант использует для сравнения элементов оператор <, вторая версия использует функтор comp: 1. **partial_sort(first, middle, last);** 2. **partial_sort(first, middle, last, comp);** Пример: **array<int, 10> s{5, 7, 4, 2, 8, 6, 1, 9, 0, 3};** **partial_sort(s.begin(), s.begin() + 3, s.end());** **for (int a : s) {** **cout << a << " "; }** **cout << endl;**
Операции двоичного поиска (на отсортированных диапазонах)	
binary_search()	Осуществляет поиск в отсортированном диапазоне [first, last) элемента, равного value. Итераторы — ForwardIterator. Первый вариант использует для сравнения элементов оператор <, вторая версия использует для сравнения функтор comp: 1. **binary_search(first, last, value);** 2. **binary_search(first, last, comp);** Пример: **vector<int> haystack {1, 3, 4, 5, 9};** **vector<int> needles {1, 2, 3};** **for (auto needle : needles) {** **cout << "Searching for " << needle << '\n';** **if (binary_search(haystack.begin(), haystack.end(), needle)){** **cout << "Found " << needle << '\n';** **} else {** **cout << "no dice!\n"; }** **}**
Операции над множествами (на отсортированных диапазонах)	
merge()	Объединяет два отсортированных диапазона [first1, last1) и [first2, last2) в один отсортированный диапазон с началом в d_first. Первый вариант используется operator< для сравнения элементов, вторая версия использует данную функцию сравнения comp: **merge(first1, last1, first2, last2, d_first);** **merge(first1, last1, first2, last2, d_first, comp);** Итераторы: InputIterator, OutputIterator

includes()	Возвращает `true`, если каждый элемент из отсортированного диапазона `[first2, last2)` находится в отсортированном диапазоне `[first, last)`. Также возвращает `true`, если `[first2, last2)` пуст. Первая версия ожидает, что оба диапазона должны быть отсортированы по возрастанию, вторая версия ожидает, что они должны быть отсортированы с заданной функцией сравнения `comp`: **`includes(first1, last1, first2, last2);`** **`includes(first1, last1, first2, last2, comp);`**
set_intersection()	Создает отсортированный диапазон начало в `d_first`, состоящей из элементов, которые встречаются в обоих отсортированых диапазонах `[first1, last1)` и `[first2, last2)` (пересечение). Первая версия ожидает, что оба входных диапазона отсортированы по возрастанию, вторая версия ожидает, что они должны быть отсортированы с данной функцию сравнения `comp`: **`set_intersection(first1, last1, first2, last2, d_first);`** **`set_intersection(first1, last1, first2, last2, d_first, comp);`** Итераторы: `InputIterator`, `OutputIterator`
set_difference()	Копирует элементы из отсортированного диапазона `[first1, last1)`, которые не встречаются в отсортированном диапазоне `[first2, last2)` в диапазон начиная с `d_first` (разница). Диапазон также будет отсортирован. Первая версия ожидает, что оба входных диапазонах должны быть отсортированы по возрастанию, вторая версия ожидает, что они должны быть отсортированы с данной функцией сравнения `comp`. **`set_difference(first1, last1, first2, last2, d_first );`** **`set_difference(first1, last1, first2, last2, d_first, comp);`**
Операции минимума/максимума	
max()	Возвращает большее из двух значений a и b. Если они эквивалентны, то возвращается a. Наибольшее значение в `ilist`. Если несколько значений эквивалентны наибольшему, то возвращается самый первый. **`max(a, b); max(a, b, comp); max(ilist); max(ilist, comp)`**
max_element()	Возвращает итератор (`ForwardIterator`), указавающий на наибольший элемент в диапазоне `[first, last)`. Если элементов эквивалентных наибольшему в диапазоне несколько, то возвращается итератор на первый такой элемент. Возвращает `last`, если диапазон пуст. **`max_element(first, last); max_element(first, last, comp);`**
min()	Возвращает наименьшее из двух значений a и b. Если они эквивалентны, то возвращается a. Наименьшее значение в `ilist`. Если несколько значений эквивалентны наименьшему, то возвращается самый первый. **`min(a, b); min(a, b, comp); min(ilist); min(ilist, comp)`**
min_element()	Возвращает итератор (`ForwardIterator`), указавающий на наименьший элемент в диапазоне `[first, last)`. Если элементов эквивалентных наименьшему в диапазоне несколько, то возвращается итератор на первый такой элемент. Возвращает `last`, если диапазон пуст. **`min_element(first, last);`** **`min_element(first, last, comp);`**
minmax()	Возвращает результат `std::make_pair(a, b)` если a<b или если a эквивалентно b. Возвращает результат `std::make_pair(b, a)` если b<a. **`minmax(const T& a, const T& b);`** **`minmax(const T& a, const T& b, comp);`** **`minmax(ilist); minmax(ilist, comp);`**
minmax_element()	Возвращает пару, состоящую из итератора (`ForwardIterator`) наименьшего элемента в качестве первого элемента и итератор наибольшего элемента в качестве второго элемента пары. Возвращает make_pair(first, first), если диапазон пуст. Если несколько элементов, эквивалентно наименьшему элементу, то возвращается итератор на первый такой элемент. Если

	несколько элементов, эквивалентно наибольшему элементу, то возвращается итератор на последний такой элемент. `minmax_element(first, last); minmax_element(first, last, comp);`

Таблица 20. Контейнеры

Контейнер	Описание
Последовательные	
Последовательные контейнеры реализуют структуры данных с возможностью последовательного доступа.	
array	Статический массив. Нужно воспринимать, как более безопасную замену С-массива. Размер и эффективность `array<T,N>` такие же, как у С-массива T[N]. Итераторы произвольного доступа. Доступны все функции из библиотеки algorithm. Единственный контейнер в котором производится инициализация по умолчанию (если значения не передаются, то элементы имеют неопределенные значения). Этот контейнер можно использовать как кортеж.

	`array<int, 10> mas {1, 5, 3, 8, 2, 6, 4, 7, 2, 9};`
	`auto first = mas.begin();`
	`auto last = mas.end();`
	`while (first != last) {`
	`  if (*first % 2 == 0)`
	`  cout << *first << " ";`
	`  first++;`
	`}`

vector	Динамический массив. Хранилище вектора обрабатывается автоматически, расширяясь и сужаясь по мере необходимости. Быстрый произвольный доступ. Вставка элементов в конец осуществляется очень быстро. Вставка в середину или в начало является неэффективной, ввиду перераспределения памяти. Для предотвращения перераспределений необходимо использовать функцию `reserve()`. Итераторы произвольного доступа.

	`vector<string> S {};`
	`cout << S.max_size() << endl;`
	`S.reserve(10);`
	`cout << S.capacity() << endl;`
	`string s {};`
	`for (size_t i = 0; i < 20; ++i) {`
	`  s += "a";`
	`  S.push_back(s); }`
	`for  (size_t i = 0; i < 20; ++i) {`
	`  cout << S.at(i) << endl; }`
	`S.resize(5);`
	`S.shrink_to_fit();`
	`cout << S.capacity() << endl;`
	`for  (size_t i = 0; i < 5; ++i) {`
	`  cout << S.at(i) << endl; }`

deque	Дек (двухсторонняя очередь). Возможность быстрой вставки и удаления элементов на обоих концах. При этом указатели и ссылки на остальные элементы остаются действительными (а итераторы - нет). Хранилище дека обрабатывается автоматически, расширяясь и сужаясь по мере необходимости. Не поддерживается управление емкостью (но shrink_to_fit использовать можно). Расширение дека более эффективно, по сравнению с вектором. Итераторы произвольного доступа. Если осуществляется вставка или удаление, то итераторы становятся недействительными. Для обхода дека нельзя использовать обычные указатели, а только интеллектуальные, т. е. итераторы.
	`deque<int> D {10};`

Контейнер	Описание

	`for (int i = 9; i >= 0; --i) {`
	`  D.push_front(i);`
	`  D.push_back(i);`
	`}`
	`for (auto &d : D)`
	`  cout << d << " ";`

forward_list	Реализован в виде однонаправленного списка. Не предоставляет произвольный доступ (таким образом отсутствуют операции [] и at()). Однонаправленные итераторы. Не поддерживается функция size(). Поддерживает быструю вставку и удаление элементов из любой позиции в контейнере. При этом ссылки, указатели и итераторы остаются действительными. Операции изменения емкости и перераспределения памяти не поддерживаются.
list	Двусвязный список. Не предоставляет произвольный доступ (таким образом отсутствуют операции [] и at()). Двунаправленные итераторы. Обход производится как с начала, так и с конца. Доступ к первому и последнему элементу осуществляется быстро. Поддерживает быструю вставку и удаление элементов из любой позиции в контейнере. При этом ссылки, указатели и итераторы остаются действительными. Операции изменения емкости и перераспределения памяти не поддерживаются. Функции-члены list (например, sort и reverse) перемещения и удаления элементов работают быстрее функций библиотеки algorithm.
Ассоциативные	
Ассоциативные контейнеры создают упорядоченные структуры данных с возможностью быстрого поиска. Реализуются с помощью сбалансированных бинарных деревьев.	
set	Set — это ассоциативный контейнер, который содержит упорядоченный набор уникальных объектов (автоматическая сортировка). При вставке или удалении элементов множества итераторы остаются действительными. Элементы множества могут иметь любой тип совместимый с критерием сортировки (по умолчанию критерий задается функтором less<>, упорядочивающим с помощью операции <). Критерий сортировки может являться частью типа. Итератор двунаправленный. Для доступа к элементам необходимо использовать тип цикла for основанного на диапазоне или итераторы.

	`#include <iostream>`
	`#include <iterator>`
	`#include <functional>`
	`#include <set>`
	`using namespace std;`
	`int main() {`
	`  istream_iterator<int> in_it(cin), eof;`
	`  // Получаем потоковым итератором произвольный набор целых чисел`
	`  set<int, greater<int>> S(in_it, eof);`
	`  // Другим потоковым итератором выводим копию упорядоченного набора`
	`  copy(S.cbegin(), S.cend(), ostream_iterator<int>(cout, " "));`
	`  return 0;`
	`}`

multiset	Доступен в заголовке <set>. Может содержать дубликаты
map	map — упорядоченный ассоциативный контейнер, который содержит пары ключ-значение с неповторяющимися ключами. Порядок следования элементов определяется ключами. При этом тип ключа должен реализовывать оператор сравнения. В остальном аналогичен set. Set можно рассматривать как особый вид map. И для set, и для map не используются (модифицирующие) функции algorithm, так как ключи константные. Функцию поиска необходимо брать как функцию-член класса map (find()), поскольку она работает гораздо эффективнее.
	`multimap<string, int> aster;`

Контейнер	Описание

	`// сортировка по убыванию`
	`multimap<int, string, greater<int>> temp;`
	`ifstream fin("input");`
	`ofstream fout("output");`
	`// Пропускаем первую строку`
	`fin.seekg(2, ios::beg);`
	`// Заполняем карту:`
	`while (!fin.eof()) {`
	`int m;`
	`string s;`
	`fin >> m >> s;`
	`aster.emplace(s, m);`
	`}`
	`auto start = aster.cbegin();`
	`auto finish = aster.cend();`
	`while (start != finish) {`
	`string s = start -> first;`
	`int m = 0;`
	`// Поиск производить не нужно, поскольку`
	`// массив уже упорядочен по ключам`
	`while (s == start -> first) {`
	`m += start -> second;`
	`start++;`
	`}`
	`// Записываем в рейтинговую карту`
	`temp.emplace(m,s);`
	`}`
	`auto beg = temp.cbegin();`
	`int k = beg -> first;`
	`string s;`
	`// На случай, если команды будут иметь одинаковый рейтинг`
	`while (k == beg -> first) {`
	`s = beg -> second;`
	`beg++;`
	`}`
	`// Имя команды победителя отправляем в файл`
	`fout << s << endl;`
	`fin.close(); fout.close();`
	`// Посмотреть что хранится в рейтинговой карте`
	`for (auto &r : temp)`
	`cout << r.first << " " << r.second << endl;`

multimap	Доступен в заголовке <map>. Может содержать дубликаты ключей
Адаптеры	
Адаптеры контейнеров предоставляют различные интерфейсы для последовательных контейнеров.	
stack	Класс является контейнером-адаптером, который имеет функциональность структуры данных стека - FILO (первым вошел, последним вышел). Внутренне stack может использовать любой из контейнеров, который поддерживает функции-члены back, push-back и pop-back (по умолчанию — это deque) .
queue	Класс является контейнерным адаптером, который имеет функциональность структуру данных очереди - FIFO (первым вошел, первым вышел). Внутренне queue может использовать любой из контейнеров, который поддерживает функции-члены front, back, push-back и push-front (по умолчанию — это deque) .
priority_queue	Очередь с приоритетом, организованная так, что самый большой элемент всегда стоит на первом месте. По умолчанию элементы выстраиваются в убывающем порядке. Определена в том же заголовочном файле, что и queue. Внутренне priority_queue может использовать любой из контейнеров, который поддерживает функции-члены front, push-back и pop-back (по умолчанию — это vector).

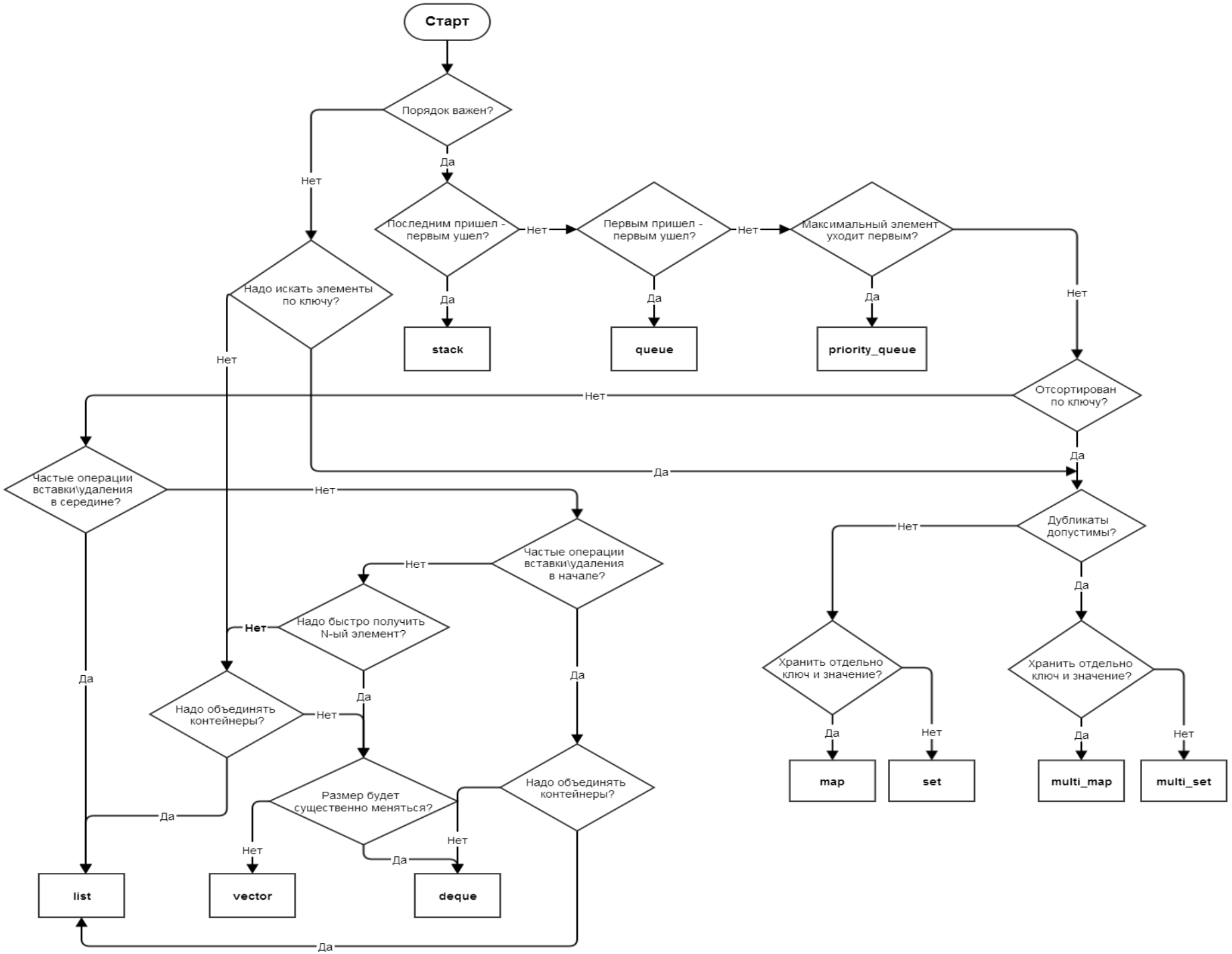
Таблица 21. Избранные функции-члены не являющиеся общими

Функция	Описание
deque	
push_back()	Добавляет данный элемент `value` до конца контейнера. Все итераторы становятся нерабочими. Указатели остаются в рабочем состоянии.
	void push_back(T&& value);
emplace_back()	Добавляет новый элемент в конец контейнера. Элемент создается на месте, без использования операций копирования или перемещения. Аргументы передаются конструктору класса
	template<class... Args> **void emplace_back(Args&&... args);**
pop_back()	Удаляет последний элемент из контейнера. Итераторы и указатели остаются в рабочем состоянии.
	void pop_back();
push_front()	Добавляет данного элемента `value` на начало контейнера. Все итераторы становятся нерабочими. Указатели остаются в рабочем состоянии.
	void push_front(T&& value);
emplace_front()	Вставляет новый элемент в начало контейнера. Элемент создается на месте, без использования операций копирования или перемещения. Аргументы передаются конструктору класса
	template<class... Args> **void emplace_front(Args&&... args);**
pop_front()	Удаляет первый элемент контейнера. Все итераторы и указатели становятся недействительными.
	void pop_front();
forward_list	
insert_after()	Вставляет элементы после указанной позиции в контейнере.
	iterator insert_after(const_iterator pos, const T& value); **iterator insert_after(const_iterator pos, T&& value);** **iterator insert_after(const_iterator pos, size_type count, const T& value);** **template<class InputIt>** **iterator insert_after(const_iterator pos, InputIt first, InputIt last);** **iterator insert_after(const_iterator pos, std::initializer_list<T> ilist);**
emplace_after()	Вставляет новый элемент после указанной позиции в контейнере. Элемент создается на месте, без использования операций копирования или перемещения.
	template<class... Args> **iterator emplace_after(const_iterator pos, Args&&... args);**
erase_after()	Удаляет указанные элементы из контейнера: следующий, после итератора или в диапазоне `(first; last)`
	iterator erase_after(const_iterator position); **iterator erase_after(const_iterator first, const_iterator last);**

Функция	Описание
list, forward_list	
merge()	Слияние двух отсортированных списков в один. Списки должны быть отсортированы в порядке возрастания. Контейнер other становится пустым.
	void merge(forward_list&& other); **template <class Compare>** **void merge(forward_list& other, Compare comp);**
remove() **remove_if()**	Удаляет все элементы, удовлетворяющие определенному условию. Первый вариант удаляет все элементы, равные value, второй вариант удаляет все элементы, для которых предикат p возвращает true
	void remove(const T& value); **template<class UnaryPredicate>** **void remove_if(UnaryPredicate p);**
reverse()	Меняет порядок элементов в контейнере. Нет ссылок или итераторы становятся недействительными.
	void reverse();
unique()	Удаляет все последовательные повторяющиеся элементы. Остаётся только первый элемент в каждой группе равных элементов. Первый вариант используется operator== для сравнения элементов, вторая версия использует данный бинарный предикат p.
	void unique(); **template<class BinaryPredicate>** **void unique(BinaryPredicate p);**
sort()	Сортировка элементов в порядке возрастания. Порядок равных элементов будет сохранен. Первый вариант для сравнения элементов использует operator<, вторая версия использует для сравнения функцию comp.
	void sort(); **template< class Compare>** **void sort(Compare comp);**
set, multiset, map, multimap	
count()	Возвращает количество элементов с ключом key.
	size_type count(const Key& key) const;
find()	Находит элемент с ключом key.
	iterator find(const Key& key); **const_iterator find(const Key& key) const;**
equal_range()	Возвращает диапазон, содержащий все элементы с ключами key в контейнере.
	std::pair<iterator, iterator> equal_range(const Key& key); **std::pair<const_iterator, const_iterator> equal_range(const Key& key) const;**
lower_bound()	Возвращает итератор, указывающий на первый элемент, который не меньше, чем key.
	iterator lower_bound(const Key& key); **const_iterator lower_bound(const Key& key) const;**
upper_bound()	Возвращает итератор, указывающий на первый элемент, который больше, чем key.
	iterator upper_bound(const Key& key);

Функция	Описание
	`const_iterator upper_bound(const Key& key) const;`
stack, priority_queue	
top()	Возвращает ссылку на верхний элемент в стеке или на верхний элемент в очереди с приоритетом
	`reference top();` `const_reference top() const;`
stack, queue, priority_queue	
pop()	Удаляет верхний элемент из стека, удаляет первый элемент очереди или удаляет верхний элемент из очереди с приоритетом.
	`void pop();`
push()	Добавляет элемент в вершину стека, или в конец очереди, или вставляет элемент и сортирует очередь с приоритетом
	`void push( T&& value );`

Для заметок


Старт
Порядок важен?
Да
Нет
Последним пришел - первым ушел?
Нет
Первым пришел - первым ушел?
Нет
Максимальный элемент уходит первым?
Да
Нет
Надо искать элементы по ключу?
Нет
stack
queue
priority_queue
Отсортирован по ключу?
Нет
Да
Да
Частые операции вставки\удаления в середине?
Нет
Да
Дубликаты допустимы?
Нет
Да
Частые операции вставки\удаления в начале?
Нет
Да
Надо быстро получить N-ый элемент?
Нет
Да
Надо объединять контейнеры?
Нет
Да
Размер будет существенно меняться?
Нет
Да
Надо объединять контейнеры?
Нет
Да
Хранить отдельно ключ и значение?
Да
Нет
Хранить отдельно ключ и значение?
Да
Нет
map
set
multi_map
multi_set
list
vector
deque

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

asciicharstable.com

Знаки функций по четвертям

sin α	cos α	tg α, ctg α
+	+	+
+	−	−
−	−	+
−	+	−

Набор выполнил ОВИЛИЧ

Копирование ЗАПРЕЩЕНО

1) $\sin^2\alpha+\cos^2\alpha=1$

2) $tg\alpha=\dfrac{\sin\alpha}{\cos\alpha}$

3) $ctg\alpha=\dfrac{\cos\alpha}{\sin\alpha}$

4) $tg\alpha\cdot ctg\alpha=1$

5) $1+tg^2\alpha=\dfrac{1}{\cos^2\alpha}$

6) $1+ctg^2\alpha=\dfrac{1}{\sin^2\alpha}$

7) $\sin^2\dfrac{\alpha}{2}=\dfrac{1-\cos\alpha}{2}$

8) $\cos^2\dfrac{\alpha}{2}=\dfrac{1+\cos\alpha}{2}$

9) $tg^2\dfrac{\alpha}{2}=\dfrac{1-\cos\alpha}{1+\cos\alpha}$

10) $tg\dfrac{\alpha}{2}=\dfrac{\sin\alpha}{1+\cos\alpha}$

11) $tg\dfrac{\alpha}{2}=\dfrac{1-\cos\alpha}{\sin\alpha}$

$-\dfrac{\pi}{2}\le arcsin a\le\dfrac{\pi}{2}$

$0\le arccos a\le\pi$

$-\dfrac{\pi}{2}<arctg\,a<\dfrac{\pi}{2}$

$0<arcctg\,a<\pi$

$\sin(arcsin a)=a$

$\cos(arccos a)=a$

$tg(arctg\,a)=a$

$ctg(arcctg\,a)=a$

Теорема Виета

$x^2+px+q=0$

$\begin{cases}x_1+x_2=-p\\ x_1\cdot x_2=q\end{cases}$

$x_1^2+x_2^2=p^2-2q$

$x_1^3+x_2^3=3pq-p^3$

Подстановки симметрических систем уравнений:

$x+y=u,\ x\cdot y=v$, тогда

$x^2+y^2=u^2-2v$

$x^3+y^3=u^3-3uv$

$x^4+y^4=$

$=u^4-4u^2v+2v^2$

12) $\sin2\alpha=2\sin\alpha\cdot\cos\alpha$

13) $\cos2\alpha=2\cos^2\alpha-1=$

$=\cos^2\alpha-\sin^2\alpha=1-2\sin^2\alpha$

14) $tg2\alpha=\dfrac{2tg\alpha}{1-tg^2\alpha}=\dfrac{2ctg\alpha}{ctg^2\alpha-1}$

15) $\sin3\alpha=3\sin\alpha-4\sin^3\alpha$

16) $\cos3\alpha=4\cos^3\alpha-3\cos\alpha$

Millennium 2004 год	0	π/6	π/4	π/3	π/2	π	3π/2
	0°	30°	45°	60°	90°	180°	270°
sin α	0	1/2	$\sqrt{2}/2$	$\sqrt{3}/2$	1	0	−1
cos α	1	$\sqrt{3}/2$	$\sqrt{2}/2$	1/2	0	−1	0
tg α	0	$\sqrt{3}/3$	1	$\sqrt{3}$	нет	0	нет
ctg α	нет	$\sqrt{3}$	1	$\sqrt{3}/3$	0	нет	0

Другие версии 2004 года ОТЗЫВАЮТСЯ!

17) $\sin(-\alpha)=-\sin\alpha$

18) $\cos(-\alpha)=\cos\alpha$

19) $tg(-\alpha)=-tg\alpha$

20) $ctg(-\alpha)=-ctg\alpha$

21) $arcsin(-a)=-arcsin a$

22) $arccos(-a)=\pi-arccos a$

23) $arctg(-a)=-arctg\,a$

24) $arcctg(-a)=\pi-arcctg\,a$

Универсальные подстановки 35), 36), 37)

$\sin\alpha=\dfrac{2tg\frac{\alpha}{2}}{1+tg^2\frac{\alpha}{2}}$ $\cos\alpha=\dfrac{1-tg^2\frac{\alpha}{2}}{1+tg^2\frac{\alpha}{2}}$ $tg\alpha=\dfrac{2tg\frac{\alpha}{2}}{1-tg^2\frac{\alpha}{2}}$

25) $\sin\alpha\cdot\cos\beta=\frac{1}{2}[\sin(\alpha-\beta)+\sin(\alpha+\beta)]$

26) $\sin\alpha\cdot\sin\beta=\frac{1}{2}[\cos(\alpha-\beta)-\cos(\alpha+\beta)]$

27) $\cos\alpha\cdot\cos\beta=\frac{1}{2}[\cos(\alpha-\beta)+\cos(\alpha+\beta)]$

28) $\sin(\alpha\pm\beta)=\sin\alpha\cdot\cos\beta\pm\cos\alpha\cdot\sin\beta$

29) $\cos(\alpha\pm\beta)=\cos\alpha\cdot\cos\beta\mp\sin\alpha\cdot\sin\beta$

38) $tg(\alpha\pm\beta)=\dfrac{tg\alpha\pm tg\beta}{1\mp tg\alpha\cdot tg\beta}$ 39) $ctg(\alpha\pm\beta)=\dfrac{ctg\alpha\cdot ctg\beta\mp1}{ctg\beta\pm ctg\alpha}$

30) $\sin\alpha\pm\sin\beta=2\sin\dfrac{\alpha\pm\beta}{2}\cdot\cos\dfrac{\alpha\mp\beta}{2}$

31) $\cos\alpha+\cos\beta=2\cos\dfrac{\alpha+\beta}{2}\cdot\cos\dfrac{\alpha-\beta}{2}$

32) $\cos\alpha-\cos\beta=-2\sin\dfrac{\alpha+\beta}{2}\cdot\sin\dfrac{\alpha-\beta}{2}$

33) $tg\alpha\pm tg\beta=\dfrac{\sin(\alpha\pm\beta)}{\cos\alpha\cdot\cos\beta}$

34) $\sin\alpha+\cos\beta=2\sin\dfrac{\alpha-\beta+90^\circ}{2}\cdot\cos\dfrac{\alpha+\beta-90^\circ}{2}$

Тригонометрические уравнения

1) $\sin t=a$; при $a\in[-1;1]$, $t=(-1)^k\,arcsin a+\pi k$

$\sin t=0;\ t=\pi k$

$\sin t=1;\ t=\frac{\pi}{2}+2\pi k$

$\sin t=-1;\ t=-\frac{\pi}{2}+2\pi k$

$\cos t=0;\ t=\frac{\pi}{2}+\pi k$

$\cos t=1;\ t=2\pi k$

$\cos t=-1;\ t=\pi+2\pi k$

2) $\cos t=a$; при $a\in[-1;1]$, $t=\pm arccos a+2\pi k$

3) $tg\,t=a$; при любом a, $t=arctg\,a+\pi k$

4) $ctg\,t=a$; при любом a, $t=arcctg\,a+\pi k$

Линейное уравнение $a\cdot\sin t+b\cdot\cos t=c$. Ответ:

$t=-arcsin\dfrac{b}{\sqrt{a^2+b^2}}+(-1)^k\,arcsin\dfrac{c}{\sqrt{a^2+b^2}}+\pi k$

$arcsin(\sin\alpha)=\alpha$, если $-\pi/2\le\alpha\le\pi/2$

$arccos(\cos\alpha)=\alpha$, если $0\le\alpha\le\pi$

$arctg(tg\,\alpha)=\alpha$, если $-\pi/2<\alpha<\pi/2$

$arcctg(ctg\,\alpha)=\alpha$, если $0<\alpha<\pi$

$arcsin\,a+arccos a=arctg\,a+arcctg\,a=\pi/2$

$\sin(\alpha\pm360^\circ)=\sin\alpha$, $tg(\alpha\pm180^\circ)=tg\alpha$

$\cos(\alpha\pm360^\circ)=\cos\alpha$, $ctg(\alpha\pm180^\circ)=ctg\alpha$

Формулы приведения	(90°±α)	(180°±α)	(270°±α)
sin (…)	cos α	∓ sin α	− cos α
cos (…)	∓ sin α	− cos α	± sin α
tg (…)	∓ ctg α	± tg α	∓ ctg α
ctg (…)	∓ tg α	± ctg α	∓ tg α

$a^2+b^2\ne$

$(a\pm b)^2=a^2\pm2ab+b^2$ $(a\pm b)^3=a^3\pm3a^2b+3ab^2\pm b^3$

$a^2-b^2=(a-b)\cdot(a+b)$ $a^3\pm b^3=(a\pm b)\cdot(a^2\mp ab+b^2)$

Координаты и векторы

$\vec a\cdot\vec b=|\vec a|\cdot|\vec b|\cdot\cos(\vec a,\vec b)=a_1b_1+a_2b_2+a_3b_3$

$\cos(\vec a,\vec b)=\dfrac{a_1b_1+a_2b_2+a_3b_3}{\sqrt{a_1^2+a_2^2+a_3^2}\cdot\sqrt{b_1^2+b_2^2+b_3^2}}$

$A(x_1;y_1;z_1),\ B(x_2;y_2;z_2)$

$\vec a(a_1;a_2;a_3),\ \vec b(b_1;b_2;b_3)$

$\overrightarrow{AB}(x_2-x_1;y_2-y_1;z_2-z_1)$

Если $\vec a\parallel\vec b$, то $\dfrac{a_1}{b_1}=\dfrac{a_2}{b_2}=\dfrac{a_3}{b_3}$

Если $\vec a\perp\vec b$, то $\vec a\cdot\vec b=0$

$|AB|=\sqrt{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2}$

$(x_0;y_0;z_0)$ – коорд. середины AB

$x_0=\dfrac{x_1+x_2}{2};\ y_0=\dfrac{y_1+y_2}{2};\ z_0=\dfrac{z_1+z_2}{2}$

$\vec a\pm\vec b=\vec c(a_1\pm b_1;a_2\pm b_2;a_3\pm b_3)$

$k\cdot\vec a=\vec d(k\cdot a_1;k\cdot a_2;k\cdot a_3)$

$|\vec a|=\sqrt{\vec a^2}=\sqrt{a_1^2+a_2^2+a_3^2}$

Логарифмы

1) $\log_a1=0$

2) $\log_a a=1$

3) $k\cdot\log_a b=\log_a b^k$

4) $\frac{1}{k}\cdot\log_a b=\log_{a^k}b$

5) $\log_a b+\log_a c=\log_a(b\cdot c)$

6) $\log_a b-\log_a c=\log_a\left(\frac{b}{c}\right)$

7) $\dfrac{\log_c b}{\log_c a}=\log_a b$

8) $\dfrac{1}{\log_b a}=\log_a b$

9) $a^{\log_a b}=b$

10) $a^{\log_b c}=c^{\log_b a}$

11) $a^{\log_{a^2}b}=b^{\log_a b}$

$(x^k)'=k\cdot x^{k-1}$. Частные случаи:

$\left(\frac{1}{x}\right)'=-\frac{1}{x^2}$, $(\sqrt{x})'=\frac{1}{2\sqrt{x}}$, $(\sqrt[3]{x})'=\frac{1}{3\cdot\sqrt[3]{x^2}}$

$(\sin x)'=\cos x$, $(\cos x)'=-\sin x$,

$(tg\,x)'=\dfrac{1}{\cos^2x}$, $(ctg\,x)'=-\dfrac{1}{\sin^2x}$

$(e^x)'=e^x$, $(a^x)'=a^x\cdot\ln a$,

$(\ln x)'=\frac{1}{x}$, $(\log_a x)'=\dfrac{1}{x\cdot\ln a}$

Ур-е касат-й: $y=f(x_0)+f'(x_0)\cdot(x-x_0)$, где $k=f'(x_0)=tg\,\varphi$ – угл. коэфф.

$l_1\parallel l_2\Rightarrow k_1=k_2$ $l_1\perp l_2\Rightarrow k_1\cdot k_2=-1$

1) $C'=0$, $(C\cdot u)'=C\cdot(u)'$

2) $(u\pm v)'=u'\pm v'$

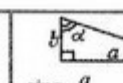
3) $(u\cdot v)'=u'\cdot v+u\cdot v'$

4) $\left(\frac{u}{v}\right)'=\dfrac{u'\cdot v-u\cdot v'}{v^2}$

5) $u'(v(x))=u'(v)\cdot v'(x)$

Логарифмическое дифференцирование:

6) $u'=u\cdot\ln'(u)$



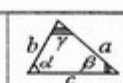
$\sin\alpha=\dfrac{a}{c}$ $\cos\alpha=\dfrac{b}{c}$

$tg\,\alpha=\dfrac{a}{b}$ $ctg\,\alpha=\dfrac{b}{a}$

т. Пифагора: $c^2=a^2+b^2$

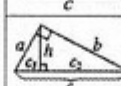
$R=\dfrac{c}{2}$ $a+b=2\cdot r+2\cdot R$

$S_{\Delta прям}=\frac{1}{2}a\cdot b$



$a^2=b^2+c^2-2bc\cdot\cos(\alpha)$ – т. косинусов

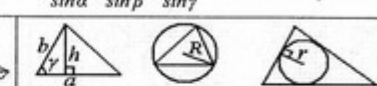
$\dfrac{a}{\sin\alpha}=\dfrac{b}{\sin\beta}=\dfrac{c}{\sin\gamma}=2R$ – т. синусов



$h^2=c_1\cdot c_2$

$a^2=c_1\cdot c$

$b^2=c_2\cdot c$



$S_\Delta=\frac{1}{2}ah=\frac{1}{2}ab\cdot\sin\gamma=\frac{1}{2}(a+b+c)r=\dfrac{abc}{4R}$

$S_\Delta=\sqrt{p(p-a)(p-b)(p-c)}$, где $p=\dfrac{a+b+c}{2}$

$S_{\Delta прав}=\dfrac{a^2\sqrt{3}}{4}$, $R=\dfrac{abc}{4S}$, $r=\dfrac{2S}{a+b+c}$

$F(x)$ – первообразная функции $f(x)$, если $F'(x)=f(x)$.

$f(x)$	x^k, $k\ne-1$	$\frac{1}{x}$	$\cos x$	$\sin x$	$\frac{1}{\cos^2x}$	$\frac{1}{\sin^2x}$	e^x	a^x	f'	$k\cdot f$	$f\pm g$	$f\cdot g$	$\frac{f}{g}$	$f(kx+b)$
$F(x)$	$\frac{x^{k+1}}{k+1}$	$\ln\lvert x\rvert$	$\sin x$	$-\cos x$	$tg\,x$	$-ctg\,x$	e^x	$\frac{a^x}{\ln a}$	f	$k\cdot F$	$F\pm G$	нет	нет	$\frac{1}{k}\cdot F(kx+b)$

Арифметическая прогрессия.

$a_n=a_1+d(n-1)$, $n=\dfrac{a_n-a_1}{d}+1$

$S_n=\dfrac{2a_1+d(n-1)}{2}\cdot n=\dfrac{a_1+a_n}{2}\cdot n$

$F(x)+C$ – общий вид первообразной

$\int_a^b f(x)dx=F(b)-F(a)$

теорема: $d_1^2+d_2^2=2\left(a^2+b^2\right)$

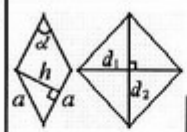
$S=ah=ab\sin(a,b)=$

$=\frac{1}{2}d_1\cdot d_2\sin\gamma$

Геом. прогрессия. $b_n=b_1q^{n-1}$

$S_n=\dfrac{b_1\left(q^n-1\right)}{q-1}=\dfrac{b_nq-b_1}{q-1}$

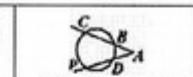
$S=\dfrac{b_1}{1-q}$, если $|q|<1$



$S=\frac{1}{2}d_1\cdot d_2$

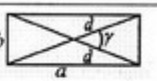
$S=a\cdot h$

$S=a^2\sin\alpha$



$AB\cdot AC=AD\cdot AP$

$НОД(a,b)\cdot НОК(a,b)=a\cdot b$

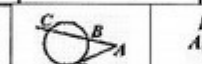


$S=ab=\frac{1}{2}d^2\sin\gamma$

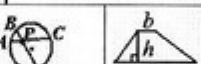
Геометрия

$S=\pi\cdot R^2=\pi\dfrac{d^2}{4}$

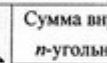
$L=2\pi\cdot R=\pi\cdot d$



$AB\cdot AC=AD^2$



$AP\cdot CP=BP\cdot DP$



$S=\frac{a+b}{2}h$

Сумма внутрен. углов выпуклого n-угольника равна $180^\circ\cdot(n-2)$

Правильный n-угольник:

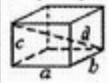
a_n – сторона, γ_n – внутренний угол

$a_n=2R\cdot\sin\left(180^\circ/n\right)=2r\cdot tg\left(180^\circ/n\right)$

1) Произвольная призма

$S_{бок}=P_{\perp сеч}\cdot l$; $V=S_{осн}\cdot h$

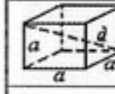
2) Прямоугольный парал-пед



$S_{бок}=2(a+b)c$

$V=abc$

Теорема: $d^2=a^2+b^2+c^2$

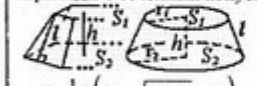


3) Куб

$V=a^3$

$d=a\sqrt{3}$

4) Произвольная усеченная пирамида. Усеченный конус.



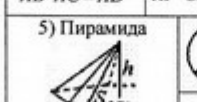
$V=\frac{1}{3}h\left(S_1+\sqrt{S_1\cdot S_2}+S_2\right)$

Прав. усеченная пирамида

$S_{бок}=\frac{1}{2}\left(P_{осн\,1}+P_{осн\,2}\right)\cdot l$

Усеч. конус $S_{бок}=\pi(r_1+r_2)\cdot l$

5) Пирамида



$V=\frac{1}{3}S_{осн}\cdot h$

Прав. пирамида

$S_{бок}=\frac{1}{2}P_{осн}\cdot l$

6) Шар, сфера

$S=4\pi r^2$

$V=\frac{4}{3}\pi r^3$

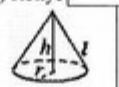
7) Цилиндр

$S_{бок}=2\pi rh$

$V=\pi r^2h$

02.05.04 г.

8) Конус

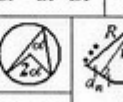


$S_{бок}=\pi rl$

$V=\frac{1}{3}\pi r^2h$

Теорема.

Отрезки подобных фигур пропорциональны коэф-ту подобия, площади – квадрату коэф-та подобия, объемы – кубу коэфф-та подобия.

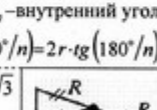


$\gamma_n=\dfrac{n-2}{2}\cdot180^\circ$

$a_3=R\sqrt{3}=2r\sqrt{3}$

$a_4=R\sqrt{2}=2r$

$a_6=R=\left(2r\sqrt{3}\right)/3$

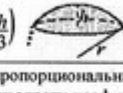


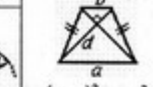
9) Шаровой сектор

$V=\frac{2}{3}\pi r^2h$

10) Шаровой сегмент

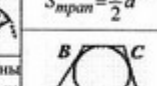
$V=\pi h^2\left(r-\frac{h}{3}\right)$





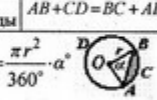
$(a+b)^2=2d^2$

$S_{трап}=\frac{1}{2}d^2$



$AB+CD=BC+AD$

Теорема. Если все боковые ребра пирамиды наклонены к плоскости основания под одним углом (или все боковые ребра равны), то вершина пирамиды проектируется в центр описанной окруж-ти вокруг основания; если все грани (или все апофемы (высоты боковых граней) равны)– в центр вписанной окружности.



$S_{сект\,OACB}=\dfrac{\pi r^2}{360^\circ}\cdot\alpha^\circ$

$S_{сегм\,ABC}=S_{сект OACB}-S_{\Delta OAB}$

$S_{сегм\,ABD}=S_{сект OADB}+S_{\Delta OAB}$

$\frac{a}{m}=\frac{b}{n}$ Биссектриса внутр-го угла тр-ка делит противоположную сторону на отрезки, пропорцион-ные боковым сторонам.

Шпаргалка: *Millennium*