

Основы программирования на C++

I семестр, 2023-2024 учебный год

Декабрь 2023 г. – зачет

Содержание: Основы программирования на языке C++

- Основы языка C++:
переменные, типы данных,
управляющие конструкции, структурное программирование,
принципы форматирования и написания кода ...
- Элементарные алгоритмы
- Принципы управления памятью
- Массивы
- Символы и строки
- Функции, процедурный подход
- Структуры
- Основы объектно-ориентированного подхода: классы и
объекты, инкапсуляция, наследование, полиморфизм

Литература



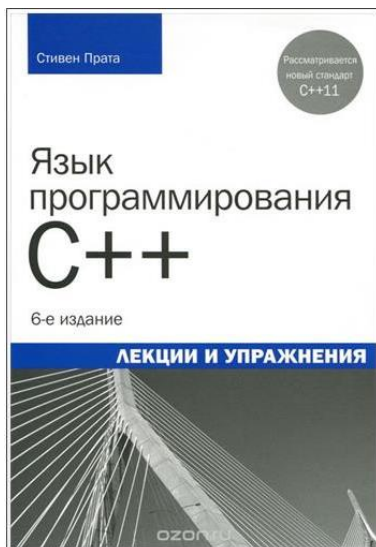
**Г. Шилдт Полное руководство по С++
(Полный справочник по С++)**



Б. Страуструп. Программирование. Принципы и практика с использованием С++



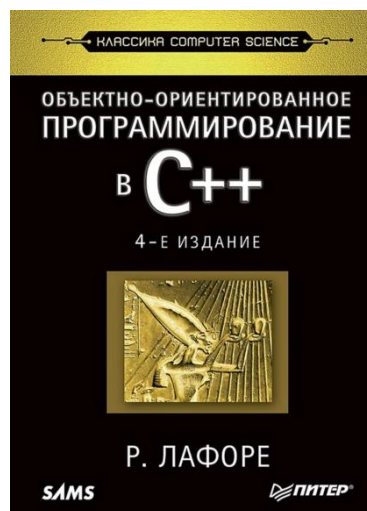
Сидхартха Рао "Освой самостоятельно С++ за 21 день"



Стивен Прата

«Язык программирования C. Лекции и упражнения»

«Язык программирования C++. Лекции и упражнения»



**Роберт Лафоре. Объектно-ориентированное
программирование в C++**

Электронные ресурсы, руководства, справочники, учебные курсы

<https://isocpp.org/>

<https://en.cppreference.com/w/cpp>

<https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170>

<https://devdocs.io/cpp/>

<https://www.learncpp.com>

<https://metanit.com>

<http://cppstudio.com>

<https://cplusplus.com/doc/tutorial/>

[...](#)

Краткая историческая справка



автор - Бьёрн Страуструп, сотрудник фирмы Bell Labs

создан - начало 1980-х годов,

1980-1998 гг. – постепенное развитие языка

1998 г. – стандарт ISO/IEC 14882:1998 (C++98)

2003 г. – стандарт C++ ISO/IEC 14882:2003

2011 г. – C++11

2014 г. – C++14

2017 г. – C++17

декабрь 2020 г. – C++20

<https://ru.wikipedia.org/wiki/C++>

Никто не обладает правами на язык C++, он является свободным

Язык программирования — формальная знаковая система,
предназначенная для записи компьютерных программ

Классификация языков программирования

по уровню:

Низкоуровневые

Высокоуровневые

[Основные языки программирования \(Wiki\)](#)

Машинные
коды,

Ассемблеры

Используй-
мые в
разработке

Академи-
ческие

IEC61131-3

Прочие

Эзотериче-
ские

Ада • АПЛ • ActionScript • ABAP/4 • AutoIt • AWK • BASIC • C • Кобол • C++ • C# • Cω • ColdFusion • Common Lisp • D • dBase • Delphi • Eiffel • Erlang • F# • Forth • Фортран • Gambas • Groovy • Haskell • Icon • Java • JavaScript • Limbo • Lua • MATLAB • Object Pascal • Objective-C • OCaml • Oz • Оберон • Parser • Паскаль • Perl • PHP • PowerBASIC • PureBasic • Python • ПЛ/1 • Пролог • Ruby • Scala • Scheme • Smalltalk • SQL • PL/SQL • Tcl • Vala • Visual Basic • VB.NET

Clean • Curry • Лого • ML • Модула-3 • Рефал • Симула

Instruction List • ST • FBD • Ladder Diagram • SFC

Алгол • Алгол 68 • Модула-2 • Miranda • Hope

HQ9+/HQ9++ • INTERCAL • Brainfuck • Befunge • Malbolge • Piet • Spoon • Unlambda • Whitespace • FALSE

Парадигма программирования - система идей и понятий, определяющих стиль написания программ

Императивное программирование
(структурно-алгоритмическое, процедурное, объектно-ориентированное, ...):

программа - это последовательность команд, которые должен выполнить компьютер

```
#include <iostream>
int main(){
    std::cout << "Hello World";
    return 0;
}
```

```
Begin
    writeln('Hello World')
End.
```

Декларативное программирование
(функциональное, логическое,...):

программа - это описание того, что надо получить

```
<html>
  <body>
    Hello World
  </body>
</html>
```

```
module HelloWorld (main)
where main = putStr "Hello World"
```


Особенности языка C++

C++ — мультипарадигменный, компилируемый, статически типизированный язык программирования общего назначения

C++ поддерживает парадигмы

- процедурное программирование
- объектно-ориентированное программирование
- обобщенное программирование,
- параллельное программирование,
- функциональное программирование

C++ близок к аппаратному обеспечению, может легко манипулировать ресурсами и является очень быстрым

Основные плюсы: высокая производительность скомпилированного приложения и непосредственный доступ к ресурсам «железа»

Минусы: небезопасность (непосредственный доступ к ресурсам «железа»), сложный синтаксис, сложность сборки, зависимости библиотек, проблемы переносимости, ...

Возможные области применения

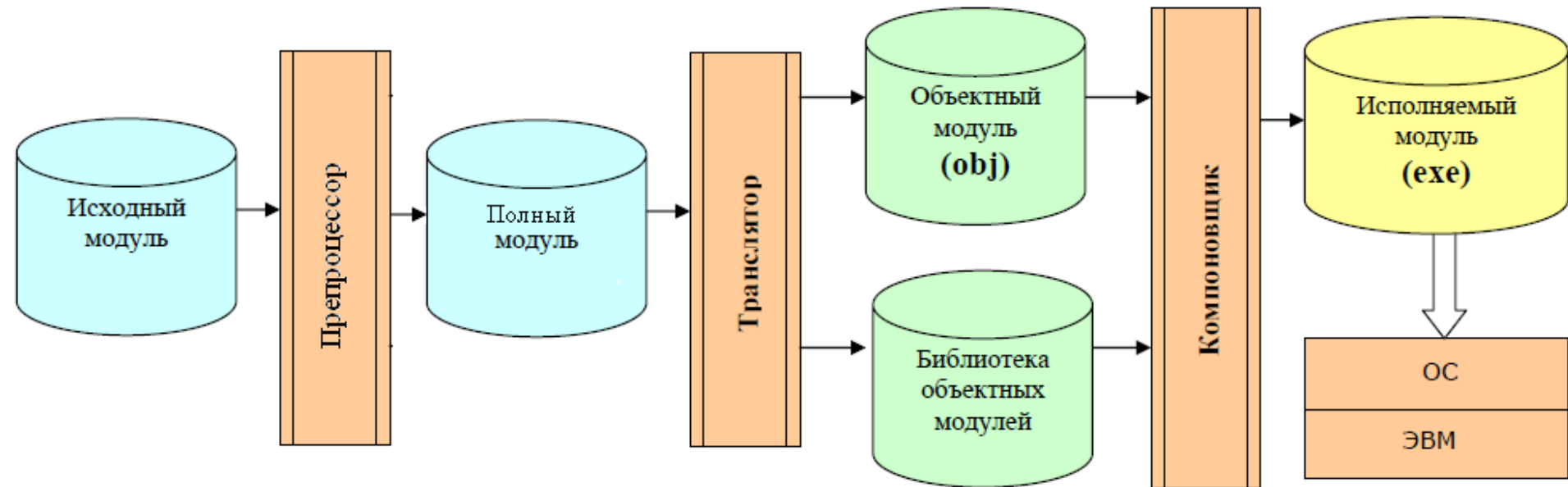
системное низкоуровневое программирование

- операционные системы, драйверы, системное ПО
- компиляторы
- системы реального времени, критичные по времени реакции (управление транспортными, производственными, биржевыми, финансовыми, ... системами)
- ПО встраиваемых систем

прикладная разработка для разнообразных платформ

- настольных приложений,
- мобильных приложений,
- научных приложений,
- AI приложений
- web-приложений,
- игр
- ...

Порядок подготовки и запуска программы



Препроцессор C/C++ — программный инструмент, изменяющий код программы для последующей компиляции и сборки

Компоновщик — программа, которая производит *компоновку*: принимает на вход один или несколько объектных модулей и собирает по ним исполнимый модуль.

Транслятор — программа или техническое средство, выполняющее преобразование программы с одного языка программирования в равносильную программу на другом языке

Компиляция — трансляция программы, составленной на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком машинному коду (абсолютный код, объектный модуль, иногда на язык ассемблера).

Интерпретация — пооператорный (покомандный, построчный) анализ, обработка и тут же выполнение исходной программы или запроса (в отличие от компиляции, при которой программа транслируется без её выполнения)

Комплект для разработки на C++

Компилятор

- g++ от проекта GNU (в составе набора компиляторов GCC)
- Clang (проект LLVM)
- компилятор C++ от компании Microsoft (используется в Visual Studio), ...

+

Редактор текста с подсветкой синтаксиса

- VS Code
- Sublime Text
- Atom, ...

Полноценные IDE

Visual Studio – <https://visualstudio.microsoft.com/ru/>

Clion, Code::Blocks, QtCreator, Eclipse, NetBeans, ...

Online-компиляторы и online-IDE

https://www.onlinegdb.com/online_c++_compiler

<https://www.online-cpp.com/>

...

Простой запуск из консоли (терминала), без IDE – пример для Windows, компилятор g++

1. Подготовить исходный код программы в файле *имяФайла.cpp*

2.

- а) Скомпилировать и запустить его, командой

```
>>> g++ имяФайла.cpp -o имяИсполняемогоФайла.exe & имяИсполняемогоФайла
```

или

- б) Скомпилировать, командой

```
>>> g++ имяФайла.cpp -o имяИсполняемогоФайла
```

- после чего выполнить его командой

```
>>> имяИсполняемогоФайла
```

<https://metanit.com/cpp/tutorial/1.2.php>

<https://metanit.com/cpp/tutorial/1.8.php>

p1 - Microsoft Visual Studio

Файл Правка Вид Проект Построение Отладка Рабочая группа Данные



Обозреватель решений

Обозреватель объектов

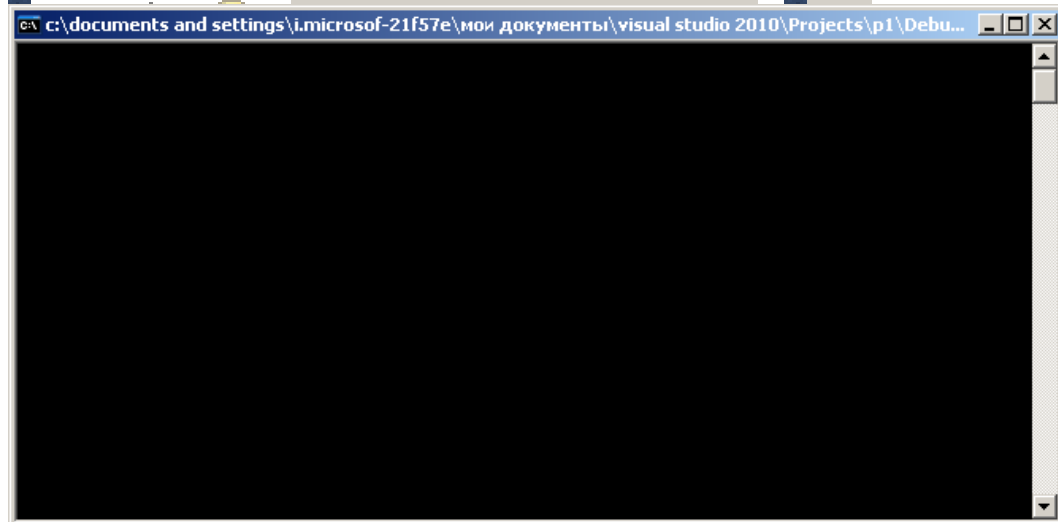
p1_1.cpp



Решение "p1" (проектов: 1)
p1
Внешние зависимости
Заголовочные файлы

(Глобальная область)

```
int main()  
{  
    return 0;  
}
```



Структура простой программы

```
#include <iostream>           // подключаем заголовочный файл iostream

int main()                    // определяем функцию main
{                              // начало функции
    std::cout << "Hello ";    // выводим строку на консоль
    return 0;                 // выходим из функции
}                              // конец функции
```


Пример решения задачи по программированию

Известны стороны треугольника, найти его площадь

I. Дано:

a, b, c

Найти:

S

II.
$$S = \sqrt{p(p-a)(p-b)(p-c)} \quad p = \frac{a+b+c}{2}$$

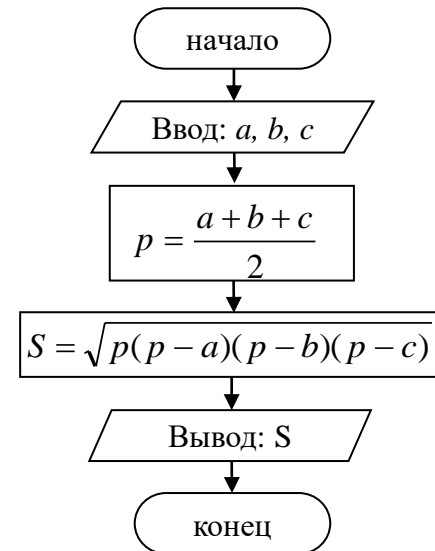
III.

1) узнать чему равны a, b и c

2) вычислить $p = \frac{a+b+c}{2}$

3) вычислить $S = \sqrt{p(p-a)(p-b)(p-c)}$

4) сообщить результат S



Известны стороны треугольника, найти его площадь

```
#include <iostream>
#include <cmath>

using namespace std;

int main() {
    double a, b, c, s, p;
    cout << "a= ";
    cin >> a;
    cout << "b= ";
    cin >> b;
    cout << "c= ";
    cin >> c;
    p = (a + b + c) / 2.0;
    s = sqrt(p * (p - a) * (p - b) * (p - c));
    cout << "s = " << s;
    return 0;
}
```

Основные элементы языка

Алфавит языка включает:

- прописные и строчные латинские буквы и знак подчеркивания:
A-Z, a-z, _
- арабские цифры от 0 до 9
- специальные знаки: * { } , | [] () + - / % . \ ' : ? < = > ! & # ~ ; ^
- пробельные символы: пробел, символы табуляции, символы перехода на новую строку

Лексемы языка (token) – наборы символом, распознаваемые компилятором:

- ключевые слова,
- идентификаторы,
- знаки операций,
- литералы,
- разделители(скобки, точка, запятая, пробельные символы)

Выражение - задает правила вычисления некоторого значения

Оператор - задает законченное описание некоторого действия

Комментарии

служат для вставки в программу пояснений к ней.

Комментарии игнорируются компилятором и могут содержать что угодно.

Комментарием является

- **// строка программы, начинающаяся с двух знаков дробли**

Примеры:

```
// текст комментария (вся строка)
```

```
i = i + 1; // текст комментария (до конца строки)
```

- фрагмент программы, который начинается с **/*** и **заканчивается ими же в обратном порядке */**. Он может занимать несколько строк или вставляться в середину строки

Примеры:

```
/* текст комментария,  
   расположенный в нескольких строках
```

```
*/
```

```
i = i + 1; /* текст комментария (между операторами)*/ j = j + 1;
```

Ключевые (зарезервированные) слова

- имеют особое значение
- не могут использоваться как имена(идентификаторы)

asm	auto	bad_cast	bad_typeid	bool
break	case	catch	char	class
const	const_cast	continue	default	delete
do	double	dynamic_cast	else	enum
extern	float	for	friend	goto
if	inline	int	long	namespace
new	operator	private	protected	public
register	reinterpret_cast	return	short	signed
sizeof	static	static_cast	struct	switch
template	then	this	throw	try
type_info	typedef	typeid	union	unsigned
using	virtual	void	volatile	while
xalloc				

Идентификаторы (имена)

Идентификаторы (имена) служат для именования элементов программы:

- объекта или переменной
- класса, структуры или объединения
- перечисленного типа
- члена класса, структуры, объединения или перечисления
- функции, функции члена класса
- определения типа (typedef)
- метки
- макроса
- параметра макроса

Идентификаторы (имена)

Правила именования:

- идентификатором может быть любая последовательность маленьких и больших латинских букв, цифр, символов подчеркивания (`_`) и денежной единицы (`$` - только для MSVC), а также другие определенные диапазоны кодовых точек Unicode
00A8, 00AA, 00AD, 00AF, 00B2-00B5, 00B7-00BA, 00BC-00BE, 00C0-00D6, 00D8-00F6, 00F8-00FF, 0100-02FF, 0370-167F, 1681-180D, 180F-1DBF, 1E00-1FFF, 200B-200D, 202A-202E, 203F-2040, 2054, 2060-206F, 2070-20CF, 2100-218F, 2460-24FF, 2776-2793, 2C00-2DFF, 2E80-2FFF, 3004-3007, 3021-302F, 3031-303F, 3040-D7FF, F900-FD3D, FD40-FDCF, FDF0-FE1F, FE30-FE44, FE47-FFFF, 10000-1FFFF, 20000-2FFFF, 30000-3FFFF, 40000-4FFFF, 50000-5FFFF, 60000-6FFFF, 70000-7FFFF, 80000-8FFFF, 90000-9FFFF, A0000-AFFFF, B0000-BFFFF, C0000-CFFFF, D0000-DFFFF, E0000-EFFFF
- идентификаторы НЕ должны начинаться с цифры или символа диапазонов 0300-036F, 1DC0-1DFF, 2DD0-2DFF, FE20-FE2F.
- учитывается регистр символов – большие и маленькие буквы **разные**.
- значимыми могут быть только первые N символов имени (например, для MSVC N=2048)
- нельзя использовать ключевые и зарезервированные слова

Идентификаторы (имена)

правильные идентификаторы:

Abc

abc

A12

nameOfPerson

BITES_PER_WORD

_qwert

неправильные идентификаторы:

12X

a-b

Вася

int

причем **abc** и **Abc** – два разных идентификатора

Соглашения об именовании

Пример из [Руководство Google по стилю в C++](#)

- ✓ необходимо давать осмысленные имена, по которым можно догадаться о назначении элемента; локальные имена могут быть краткими, для элементов с широкой областью доступа – более подробные
- ✓ для соединения нескольких слов в имени используется формат camelCase или PascalCase – каждое следующее слово с заглавной буквы без символа `_`, либо только разделение символом `_`, но не надо смешивать эти стили
- ✓ использовать латинские буквы; другие алфавиты возможны, но не рекомендуются
- ✓ Имена всех типов — классов, структур, псевдонимов, перечислений, параметров шаблонов — именуются в одинаковом стиле. Имена типов начинаются с прописной буквы, каждое новое слово также начинается с прописной буквы. Подчёркивания не используются.
- ✓ Имена переменных (включая параметры функций) и членов данных пишутся строчными буквами с подчёркиванием между словами. Члены данных классов (не структур) дополняются подчёркиванием в конце имени.
- ✓ Обычные функции именуются в смешанном стиле (прописные и строчные буквы); функции доступа к переменным (accessor и mutator) должны иметь стиль, похожий на целевую переменную. Обычно имя функции начинается с прописной буквы и каждое слово в имени пишется с прописной буквы
- ✓ Имена констант начинаются с символа «k», далее идёт имя в смешанном стиле (прописные и строчные буквы). Подчёркивание может быть использовано в редких случаях когда прописные буквы не могут использоваться для разделения.
- ✓ Пространство имён называется строчными буквами. Пространство имён верхнего уровня основывается на имени проекта.
- ✓ Элементы перечисления (как с ограничениями на область видимости (scoped), так и без (unscoped)) должны

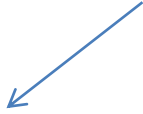
Программа -

последовательность инструкций, предназначенных для исполнения устройством управления вычислительной машины.

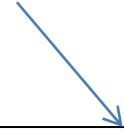
Програ́мма (от греч. *προ* — пред и *γραμμή* — запись) — «предписание», предварительное описание предстоящих событий или действий (Wiki)

= Данные + Инструкции

Данные: константы и переменные



**Не меняют значения в
ходе работы программы**



**Могут менять значения в
ходе работы программы**

- **Литералы**
- **Макроконстанты**
- **Именованные константы**

Литеральные константы (литералы)

Значения, используемые непосредственно в тексте программы

```
double price = 134.75;  
double cost = price * 1000000;  
cout << "Название : \n" << "Программировать на языке C++" << endl;  
cout << "Размер : \t" << 357 << endl;  
cout << "Цена $: \t" << price << endl;  
return 0;
```

Числовые литералы

Целые	Десятичный — последовательность цифр, не начинающаяся с нуля	123 1999
	Восьмеричный — последовательность цифр от нуля до семерки, начинающаяся с нуля	011 0177
	Шестнадцатеричный — последовательность шестнадцатеричных цифр (0 - 9 и A - F), перед которой стоит 0X или 0x	0X9A 0xffff
	Двоичный — последовательность цифр 0 или 1, начинающаяся с 0b (C++14)	0b11
	Разряды в длинных числах можно разделять	
Вещественные	Десятичный — [цифры].[цифры]	123. 3.14 .99
	Экспоненциальный — [цифры]E e[+ -] цифры	3e-7 1.17e6

Числовые литералы разных типов

По умолчанию (без суффикса):

134.75 – тип `double`

15 – тип `int`

Тип данных	Суффикс	Пример
<code>unsigned int</code>	u или U	15u
<code>long</code>	l или L	15L
<code>unsigned long</code>	ul, uL, Ul, UL, lu, lU, Lu или LU	15ul
<code>long long</code>	ll или LL	15LL
<code>unsigned long long</code>	ull, uLL, Ull, ULL, llU, llU, LLu или LLU	15ull
<code>float</code>	f или F	134.75f
<code>long double</code>	l или L	134.75L

Символьные и строковые литералы

Символьный	Одиночный символ, заключенный в апострофы	'W' '?' 'Ю' '5' '\101', '\x041'																										
Строковый	<p>Последовательность символов, заключенная в двойные кавычки.</p> <p>Может включать escape-последовательности:</p> <table><tr><td>\a</td><td>Звуковой сигнал</td></tr><tr><td>\b</td><td>Возврат на шаг</td></tr><tr><td>\f</td><td>Перевод страницы (формата)</td></tr><tr><td>\n</td><td>Перевод строки</td></tr><tr><td>\r</td><td>Возврат каретки</td></tr><tr><td>\t</td><td>Горизонтальная табуляция</td></tr><tr><td>\v</td><td>Вертикальная табуляция</td></tr><tr><td>\\</td><td>Обратная косая черта</td></tr><tr><td>\'</td><td>Апостроф</td></tr><tr><td>\"</td><td>Кавычка</td></tr><tr><td>\?</td><td>Вопросительный знак</td></tr><tr><td>\ooo</td><td>Восьмеричный код символа</td></tr><tr><td>\xhhh</td><td>Шестнадцатеричный код символа</td></tr></table>	\a	Звуковой сигнал	\b	Возврат на шаг	\f	Перевод страницы (формата)	\n	Перевод строки	\r	Возврат каретки	\t	Горизонтальная табуляция	\v	Вертикальная табуляция	\\	Обратная косая черта	\'	Апостроф	\"	Кавычка	\?	Вопросительный знак	\ooo	Восьмеричный код символа	\xhhh	Шестнадцатеричный код символа	"Вася" "Это строка \n" "При печати такой строки будет звуковой сигнал \a" "Эта строка содержит апостроф \' и вопросительный знак \?"
\a	Звуковой сигнал																											
\b	Возврат на шаг																											
\f	Перевод страницы (формата)																											
\n	Перевод строки																											
\r	Возврат каретки																											
\t	Горизонтальная табуляция																											
\v	Вертикальная табуляция																											
\\	Обратная косая черта																											
\'	Апостроф																											
\"	Кавычка																											
\?	Вопросительный знак																											
\ooo	Восьмеричный код символа																											
\xhhh	Шестнадцатеричный код символа																											

Макроконстанты

Литералы, именованные через директиву #define

```
#include <iostream>
```

```
#define TITLE "Программировать на языке C++"
```

```
#define CLASS_SIZE 125
```

```
#define PRICE 22.95
```

```
using namespace std;
```

```
int main(){
```

```
    cout << "Название : " << TITLE << endl;
```

```
    cout << "Размер : " << CLASS_SIZE << endl;
```

```
    cout << "Цена: $" << PRICE << endl;
```

```
    cout << "Название : " << TITLE << endl;
```

```
    cout << "Размер : " << 234 << endl;
```

```
    cout << "Цена: $" << PRICE << endl;
```

```
    return 0;
```

```
}
```


Именованные константы

Должна быть объявлена прежде чем будет использоваться

Объявление типизированной именованной константы:

<i>const</i>	<i>тип данных</i>	<i>имя_константы = константное [, ...] ; выражение</i>
---------------------	-----------------------	--

Примеры:

```
const double pi = 3.14159;  
const int count = 10 * 543;  
const double sq_count = sqrt(count);
```

Переменные

Переменная - именованная типизированная область памяти

Создается программистом по мере необходимости

(в любом месте программы)

Должна быть объявлена прежде чем будет использоваться

Объявление, определение, инициализация переменных:

<i>[спецификаторы]</i>	тип	идентификатор	[= <i>начальное</i>] [, ...] ;
	данных		<i>значение</i>

Примеры:

```
int a;  
int b, c = 7, d;  
  
double x, y = 7.0, z;  
  
char s = 'Ю', ss;  
  
bool f1, f2 = true;
```

Автоматический вывод типа

```
#include <iostream>
#include <iomanip>
#include <typeinfo>
#include <cmath>

using namespace std;
int main()
{
    auto a = 6;
    auto d = 7.8;
    auto x = 287 + 100'000 + sin(1);
    auto str = string("some String");

    cout << a << " : type = " << typeid(a).name() << endl;
    cout << d << " : type = " << typeid(d).name() << endl;
    cout << fixed << setprecision(3) << x <<
        " type = " << typeid(x).name() << endl;
    cout << str << " : type = " << typeid(str).name() << endl;

    return 0;
}
```

Типы данных

Определяют:

- Внутренне представление данных в памяти
(размер ячейки памяти, формат данных)
- Множество возможных значений переменной
- Операции, которые можно совершать с данными

Типы данных:

- 1) простые (скалярные, базовые, встроенные)
- 2) составные (агрегатные, структурированные, определяемые пользователем-программистом)

Простые встроенные типы данных

- Целые: **int**,
с модификаторами `short`, `long`, `signed`, `unsigned`
- Вещественные: **double**, **float**
- Логический: **bool**
- Символьные: **char**, ...

Целые типы данных

Тип данных	Размер (бит)	Диапазон
short signed short int	16	-32768 .. 32767
unsigned short unsigned short int	16	0 .. 65535
int	32	-2147483648 .. 2147483647
unsigned int	32	0 .. 4294967295
long signed long int	32	-2147483648 .. 2147483647
unsigned long unsigned long int	32	0 .. 4294967295
long long signed long long int	64	-9 223 372 036 854 775 808 .. +9 223 372 036 854 775 807
unsigned long long unsigned long long int	64	0 .. 18 446 744 073 709 551 615

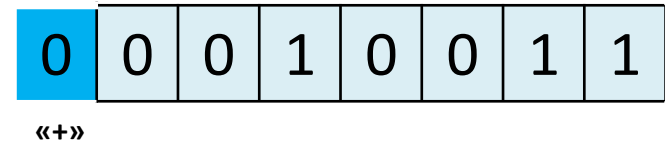
Форматы с фиксированной точкой

[Вики от ИТМО](#)

- **Целые без знака:** `unsigned` и
положительные знаковые: `+signed`

Прямой код: двоичная форма числа выравнивается в ячейке памяти по правому краю, старшие разряды заполняются нулями

Число -19: Двоичный код числа: 10011

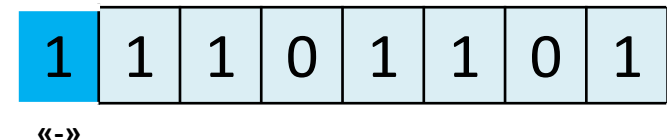


- **Знаковые отрицательные:** `-signed`

Дополнительный код:

- 1) старший разряд – знаковый: 0 – плюс; 1 – минус
- 2) Прямой код отрицательного числа - двоичная форма модуля числа выравнивается в ячейке памяти по правому краю, старшие разряды заполняются нулями, в знаковом разряде – 1
- 3) Обратный код – все разряды кроме знакового инверсирую
- 4) **Дополнительный код** – к обратному коду прибавить 1

Число -19: Код модуля числа: 0 0010011
Обратный код числа: 1 1101100



```

#include <iostream>
#include <bitset>
#include <format>

using namespace std;
int main()
{
    short a = 19, b = -19;
    cout << "a = " << a << " = " << bitset<16>(a) << endl;
    cout << "b = " << b << " = " << bitset<16>(b) << endl << endl;

    short aa = 32767 + 1, bb = -32768 - 1;
    cout << "aa = " << aa << " = " << bitset<16>(aa) << endl;
    cout << "bb = " << bb << " = " << bitset<16>(bb) << endl;

    return 0;
}

```

```

a = 19 = 00000000000010011
b = -19 = 11111111111101101

aa = -32768 = 10000000000000000
bb = 32767 = 01111111111111111

```


Вещественные типы данных (с плавающей точкой)

Тип данных	Размер (бит)	Диапазон
float	32	$3.4 \cdot 10^{-38} - 3.4 \cdot 10^{38}$
double	64	$1.71010 \cdot 10^{-308} - 1.710 \cdot 10^{308}$
long double	80	$3.4 \cdot 10^{-4932} - 3.4 \cdot 10^{4932}$

Форматы с плавающей точкой/запятой

$$X_p = M \cdot p^E$$

X_p – число в системе с основанием p

M – мантисса

E – порядок

p – основание системы счисления

$-13,125_{10} =$

$=$

$=$

Нормализованная форма

Нормализованная форма: $|M| \in [1; p)$

$$-13,125_{10} = \quad |M| \in [1; 10)$$

В такой форме любое число (кроме 0) записывается единственным образом

0 - ?

Нормализованная форма в двоичном представлении

$$-13,125_{10} = -1101,001_2$$

$$= -1,101001_2 \cdot 2^3 = -1,101001_2 \cdot 10^{11}$$

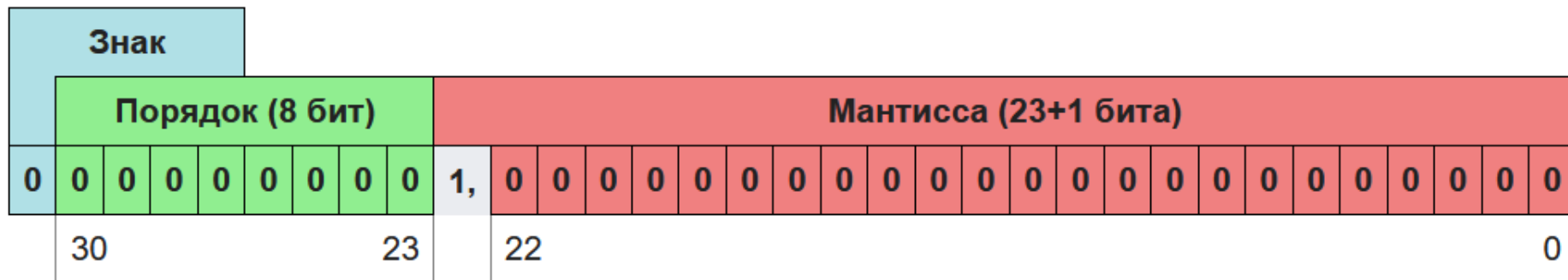
– нормализованная форма

при $|M| \in [1; 2)$

старший разряд мантиссы
всегда равен 1

Числа с плавающей точкой (по IEEE 754)

на примере одинарной точности (32 бита - *Single precision, float*)



Алгоритм записи двоичного представления с плавающей точкой

- представить число в нормализованной двоичной

форме;

$$-13,125_{10} = -1,101001_2 \cdot 2^{11}$$

- записать в сетку все разряды мантиссы, кроме старшего

(он всегда =1, подразумевается, но не пишется)

$|M| = \underline{1},101001$

- прибавить к порядку смещение и перевести смещенный порядок в двоичную систему;

$$E = 3_{10} = 11_2$$

$$3_{10} + 127_{10} =$$

$$130_{10} = 10000010_2$$

- записать знак числа в знаковый разряд
(0 -положительное; 1 - отрицательное)

знак мантииссы – минус;
в знаковом разряде **1**


[illegible]

```
#include <iostream>
#include <bitset>
using namespace std;
int main() {

float f = -3.125;
int nf = *((int*)&f);

cout << f << endl;
cout << bitset<32>(nf);

return 0;
}
```

 Microsoft Visual Studio Debug Console

-3.125

11000000010010000000000000000000

```
cout << 1 / 0 << endl; // не скомпилируется  
cout << 1.0 / 0.0 << endl; // не скомпилируется (msvc)
```

```
int a = 1, b = 0;  
cout << a / b << endl; // exception - Integer division by zero
```

```
double p = 1.0, q = 0.0;  
cout << p / q << endl; // Inf  
cout << 1.0 / (p / q) << endl;  
cout << -p / q << endl;  
cout << sin(p / q) << endl;
```

```
inf  
0  
-inf  
-nan(ind)
```


Специальные значения

Ноль (со знаком)

Знак																				
Порядок						Мантисса														
$0/1$	0	0	0	0	0	1,	0	0	0	0	0	0	0	0	0	0	$= \pm 0$			
	14				10		9									0				

Бесконечности (*Inf*)

Знак																= ±∞
Порядок						Мантисса										
0/1	1	1	1	1	1	1,	0	0	0	0	0	0	0	0	0	
	14		10				9									0

Неопределенность (*NaN*)

Знак																	= NaN
Порядок						Мантисса											
0/1	1	1	1	1	1	1,	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	
	14				10		9										0

Погрешности вычислений на данных с плавающей точкой

```
#include <iostream>
#include <iomanip>
#include <bitset>
using namespace std;
int main() {
    setlocale(0, "");

    float x = 0.1 + 0.2;

    if (0.3 == x) {
        cout << "0.1 + 0.2 = 0.3";
    }
    else {
        cout << "что-то пошло не так\n";
    }
    cout << "x = " << x << endl;
    cout << setprecision(8) << "x = " << x << endl;

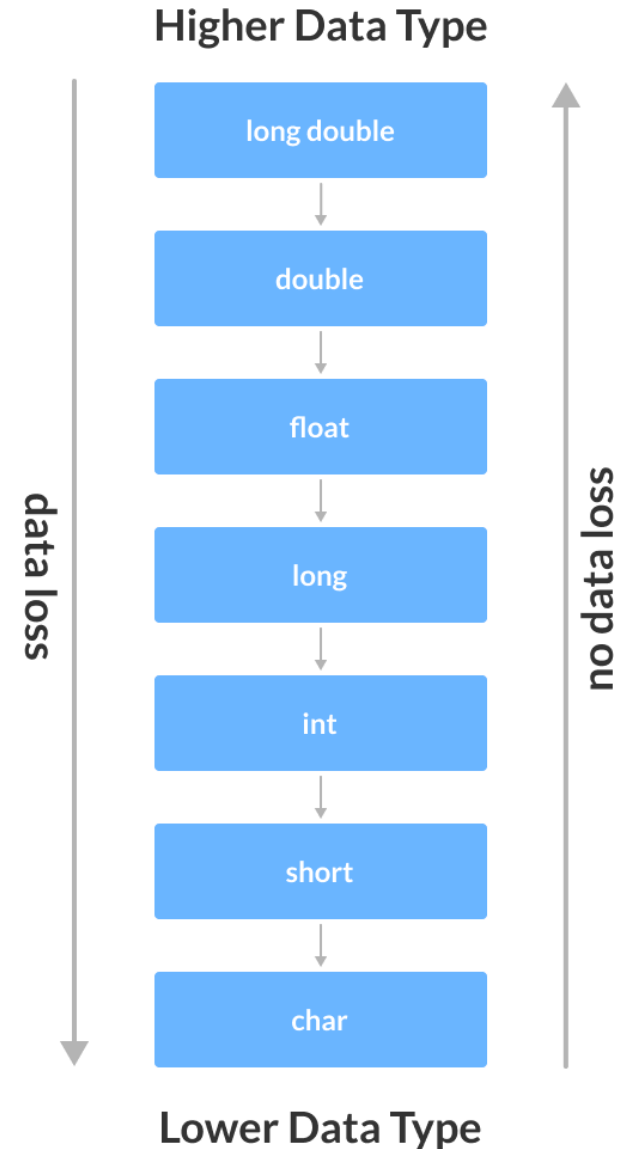
    if ((0.1 + 0.2) + 0.3 == 0.1 + (0.2 + 0.3)) {
        cout << "они равны";
    }
    else {
        cout << "что-то пошло не так\n";
    }
    cout << setprecision(48) << (0.1 + 0.2) + 0.3 << endl;
    cout << setprecision(48) << 0.1 + (0.2 + 0.3) << endl;
```

Преобразования типов

Числовое продвижение (расширяющее преобразование типа) – это преобразование более узкого числового типа (например, `char`) в более широкий числовой тип (обычно `int` или `double`)

Сужающее преобразование –числовое преобразование, которое может привести к потере данных:

- из типа с плавающей запятой в целочисленный тип;
- из более широкого типа с плавающей запятой в более узкий тип с плавающей запятой, если преобразовываемое значение не `constexpr` и не находится в диапазоне целевого типа (даже если оно не может быть представлено точно);
- из целочисленного типа в тип с плавающей запятой, если только преобразуемое значение не `constexpr` и не находится в диапазоне целевого типа и не может быть преобразовано обратно в исходный тип без потери данных;
- из более широкого целочисленного типа в более узкий целочисленный тип, если преобразовываемое значение не `constexpr` и после целочисленного продвижения не будет соответствовать целевому типу;



Явные преобразования (приведения) типов

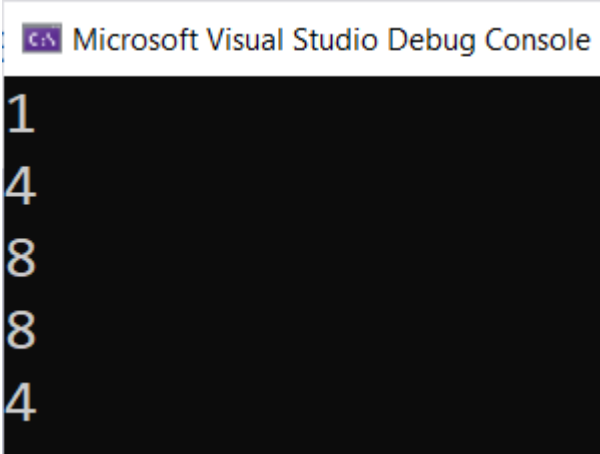
Операции приведения типа в стиле C

Определение размера занимаемой памяти

Оператор sizeof – это унарный оператор, который принимает тип или переменную и возвращает ее размер в байтах.

```
cout << sizeof(bool) << endl;  
cout << sizeof(long) << endl;  
cout << sizeof(double) << endl;  
cout << sizeof(long double) << endl;
```

```
float price = 134.75f;  
cout << sizeof(price) << endl;
```



The screenshot shows the Microsoft Visual Studio Debug Console with a black background and white text. The output of the C++ code is displayed as a vertical list of numbers: 1, 4, 8, 8, and 4, each on a new line. These numbers represent the size in bytes of bool, long, double, long double, and float, respectively.

```
Microsoft Visual Studio Debug Console  
1  
4  
8  
8  
4
```

Операции языка (operators – операторы)

Операция – элементарное действие над данными

Арифметические

+, -, *, /, %, ++, --

Присваивания

простая =

составные +=, -=, *=, /=, ...

Сравнения

<, >, <=, >=, ==, !=

Логические

!, &&, ||

Побитовые

~, &, |, ^, <<, >>

Работа с указателями и членами класса

, &, [], ->, .., ->*, .

Другие

::, sizeof, (), new, delete, (type), ...

-арность – количество операндов, задействованных в операции

- **унарные** операции - один операнд

-x

++y

!true

...

- **бинарные** операции - два операнда

x - y

cin >> x

true && false

...

- **тернарная** операция - три операнда

x < y ? 0 : x

Приоритет – порядок, в котором выполняются операции.

- Чем выше приоритет, тем раньше выполняется операция
- Для изменения порядка применяются круглые скобки

Операция	Описание	Приоритет	Ассоциация
Первичные и постфиксные операции			
[]	индексация массива	16	слева направо
()	вызов функции	16	слева направо
.	элемент структуры	16	слева направо
->	элемент указателя	16	слева направо
++	постфиксный инкремент	15	слева направо
--	постфиксный декремент	15	слева направо
Одноместные операции			
++	префиксный инкремент	14	справа налево
--	префиксный декремент	14	справа налево
sizeof	размер в байтах	14	справа налево
(тип)	приведение типа	14	справа налево
~	поразрядное NOT	14	справа налево
!	логическое NOT	14	справа налево
-	унарный минус	14	справа налево
&	взятие адреса	14	справа налево
*	разыменование указателя	14	справа налево
Арифметические			
*	умножение	13	слева направо
/	деление	13	слева направо
%	взятие по модулю	13	слева направо
+	сложение	12	слева направо
-	вычитание	12	слева направо
Поразрядного сдвига			
<<	сдвиг влево	11	слева направо
>>	сдвиг вправо	11	слева направо

Отношения			
<	меньше	10	слева направо
<=	меньше или равно	10	слева направо
>	больше	10	слева направо
>=	больше или равно	10	слева направо
==	равно	9	слева направо
!=	не равно	9	слева направо
Поразрядные			
&	поразрядное AND	8	слева направо
^	поразрядное XOR	7	слева направо
	поразрядное OR	6	слева направо
Логические			
&&	логическое И	5	слева направо
	логическое ИЛИ	4	слева направо
Условные			
?:	условная операция	3	справа налево
Присваивания			
=	присваивание	2	справа налево
*=	присвоение произведения	2	справа налево
/=	присвоение частного	2	справа налево
%=	присвоение модуля	2	справа налево
+=	присвоение суммы	2	справа налево
-=	присвоение разности	2	справа налево
<<=	присвоение левого сдвига	2	справа налево
>>=	присвоение правого сдвига	2	справа налево
&=	присвоение AND	2	справа налево
^=	присвоение XOR	2	справа налево
=	присвоение OR	2	справа налево
,	запятая	1	слева направо

Операторы языка C++

(statements - команды, инструкции)

- элементы программы, которые контролируют способ и порядок обработки объектов

Простые инструкции:

1. Объявление
 - переменных, констант, ...
2. Выражение
 - expression и null statement
3. Простая передача управления
 - goto, break, continue, return

Структурные операторы передачи управления

Блок {}

1. Условный переход
 - if .. else
 - switch
2. Циклы
 - while
 - do .. while
 - for
3. Обработка исключений
 - try .. throw .. catch

Простые операторы: объявление и определение

Все элементы программы должны быть названы и размещены в памяти

- **Объявление** указывает уникальное имя элемента программы, а также сведения о его типе и других характеристиках. Точка объявления имени, является началом его видимости компилятору.

```
extern int x; // объявление переменной
```

```
int func(); // объявление функции
```

- **Определение** дает компилятору все сведения, необходимые для создания машинного кода при последующем использовании сущности в программе.

```
int y; // объявление и определение переменной
```

```
int func() { // объявление и определение функции
```

```
    int z, w = 7, q; // объявление и определение переменных  
    return 5;
```

```
}
```

Все сущности программы (переменные, константы, функции, типы,...)
объявляются и определяются по своим правилам

Простые операторы: выражение

Выражение – конструкция, которая задает правило вычисления некоторого значения

Выражение состоит из

- операндов:
 - константы
 - переменные
 - вызовы функций
 - другие выражения
 - знаков операций (+, -, *, /, %, <, >, <=, >=, ...)
 - скобок
-

Пример:

$x = (\text{first} - \text{second} * \text{sqrt}(2 * y)) / 3$

Оператор выражение -

Выражение;

Простые операторы:

простая передача управления

goto *метка*;

...

метка : оператор;

передает управление на оператор, отмеченный меткой

break;

прерывает выполнение структурного оператора (if, switch, while, do..while, for), передает управление на следующий после структурного оператор

continue;

прерывает выполнение текущей итерации цикла (while, do..while, for), начинает следующую итерацию (шаг) цикла

return [*выражение*];

завершает работу функции, возвращает результат в точку вызова

Структурные операторы

сформулирована и доказана итальянскими математиками Коррадо Бёмом и Джузеппе Якопини, 1966 г

Программа может быть представлена с помощью трех управляющих структур:

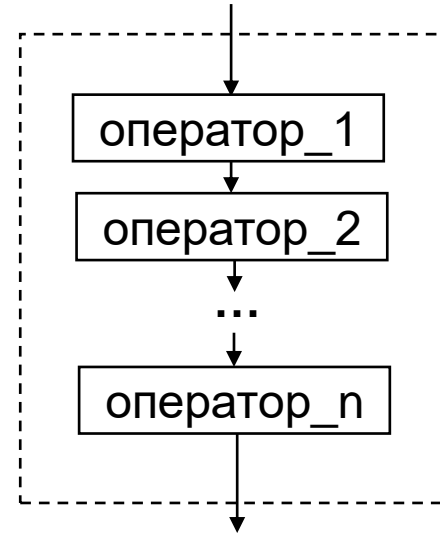
последовательность

ветвление

цикл

Структурные операторы. Блоки

```
{  
    оператор_1;  
    оператор_2;  
    ...  
    оператор_n;  
}
```



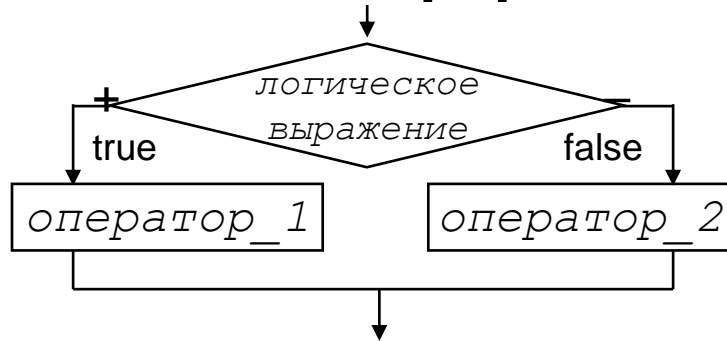
Составной оператор (блок) - последовательность операторов, в фигурных скобках { }.

Блок понимается как один оператор и может использоваться везде, где допускается применение только одного оператора.

Переменные и константы, объявленные внутри блока считаются локальными объектами этого блока и доступны только в нем.

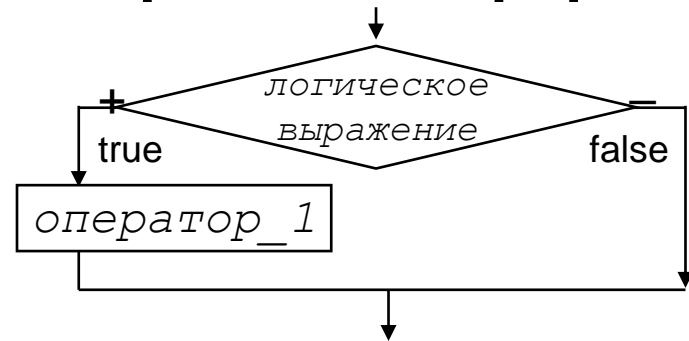
Условный оператор **if**: полная и сокращенная формы

Полная форма



```
if (логическое_выражение)  
    оператор_1;  
else  
    оператор_2;
```

Сокращенная форма



```
if (логическое_выражение)  
    оператор_1;
```

Порядок выполнения:

1. Вычисляется значение *логического_выражения*
2. Если значение *логического_выражения* **ИСТИННО** (равно true), то выполняется *оператор_1*, иначе выполняется *оператор_2*
3. Управление передается на следующий за **if** оператор

1. Вычисляется значение *логического_выражения*
2. Если значение *логического_выражения* **ИСТИННО** (равно true), то выполняется *оператор_1*
3. Управление передается на следующий за **if** оператор

Условный оператор `if`: правила синтаксиса

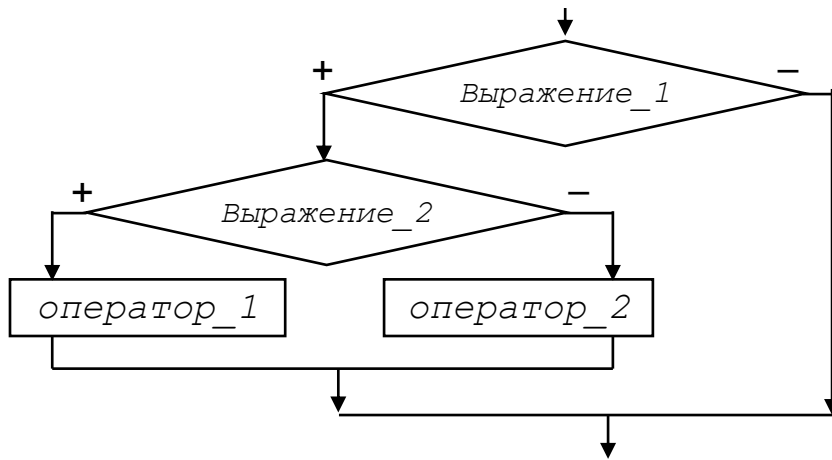
```
if (логическое_выражение)
    оператор_1;
else
    оператор_2;
```

1. *Логическое_выражение* должно быть в круглых скобках ()
2. Если в какой-то ветви надо выполнить несколько операторов, то их надо заключить в фигурные скобки { }, оформить как составной оператор – блок
Многие разработчики всегда оформляют блок даже для одной инструкции в ветви
3. *Оператор_2* вместе с ключевым словом ***else*** может быть опущен – сокращенная форма оператора `if`

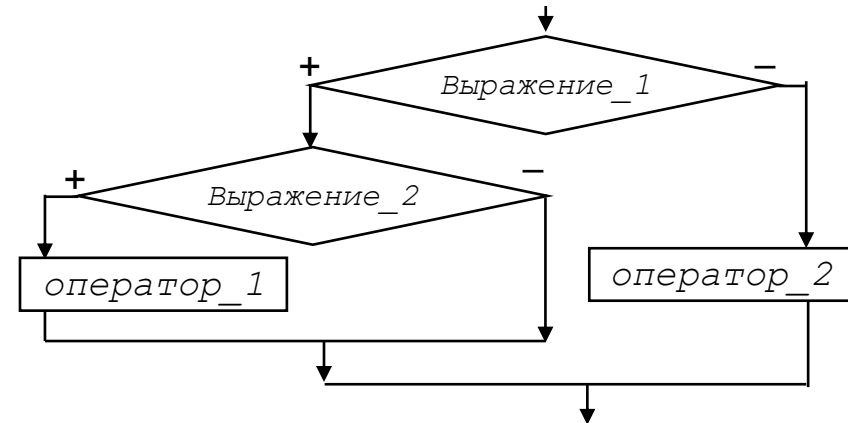
Вложенные условные операторы `if`

Во вложенных операторах `if` ключевое слово `else` относится к ближайшему к нему сверху слову `if`.

Если необходима иная последовательность выполнения операторов, то их порядок регулируется фигурными скобками.



```
if (Выражение_1)
  if (Выражение_2)
    оператор_1;
else
  оператор_2;
```



```
if (Выражение_1) {
  if (Выражение_2)
    оператор_1;
}
else
  оператор_2;
```

Пример использования оператора if

Задача. Известно вещественное число x . Вычислить и вывести $y = \frac{1}{x}$

I. Постановка задачи

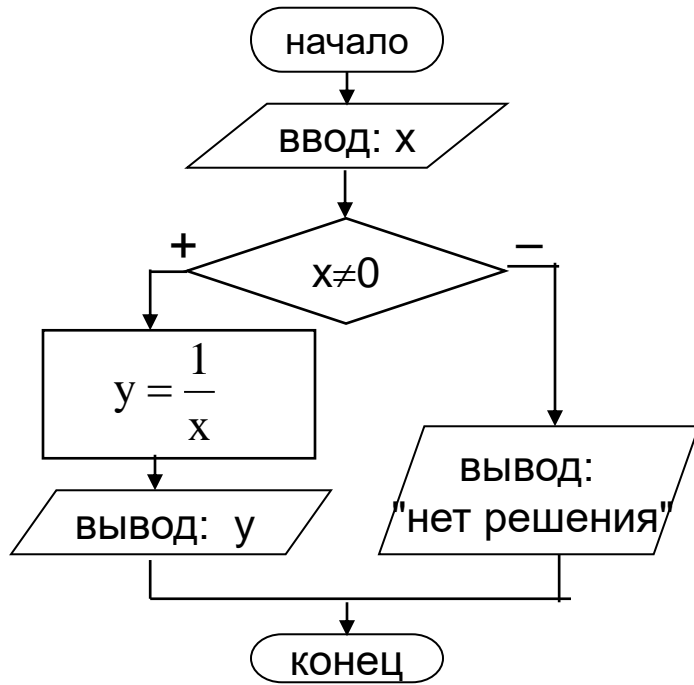
Дано: x

Найти: y

II. Математическая формулировка

$$y = \begin{cases} \frac{1}{x}, & \text{если } x \neq 0 \\ \text{не определено,} & \text{если } x = 0 \end{cases}$$

III. Блок-схема алгоритма



IV. Программа

```
#include <iostream>
using namespace std;
int main()
{
    double x, y;
    cout<<"Введите число x";
    cin>>x;
    if (x!=0)
    {
        y=1./x;
        cout<<"y=1/x="<<y;
    }
    else
        cout<<"нет решения";
    return 0;
}
```

Пример полной формы if

Известны цена и количество товара. Если общая стоимость покупки превышает 1000 руб., то предоставляется скидка в 5% от этой стоимости. Вывести стоимость данной покупки с учетом возможной скидки и сообщение о том, была ли скидка предоставлена.

I. Постановка задачи

Дано: c , руб. – цена товара
 k , шт. – количество

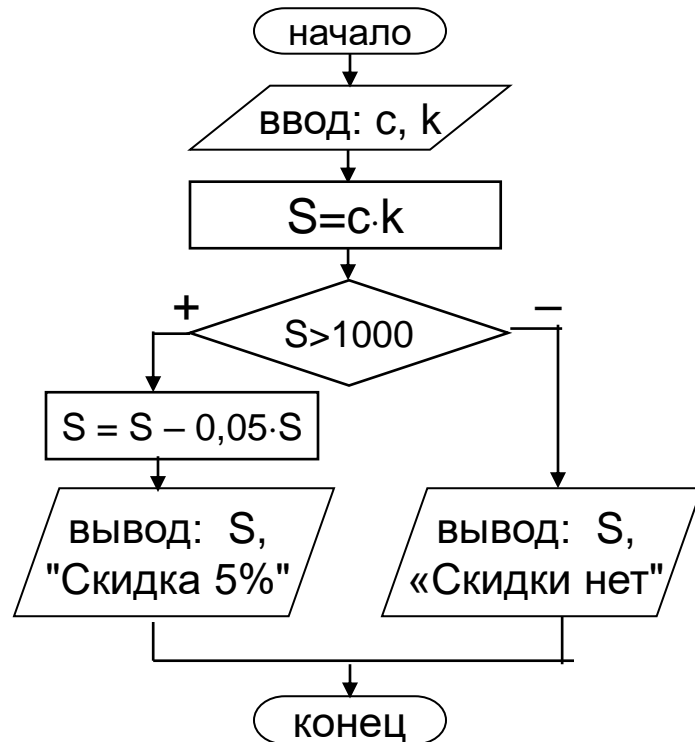
Найти: S , руб. – стоимость
 M – текстовое сообщение

II. Математическая формулировка

$$S = \begin{cases} c \cdot k - 0,05c \cdot k, & \text{если } c \cdot k > 1000 \\ c \cdot k, & \text{иначе} \end{cases}$$

$$M = \begin{cases} \text{"Скидка 5\%"}, & \text{если } c \cdot k > 1000 \\ \text{"Нет скидки"}, & \text{иначе} \end{cases}$$

III. Блок-схема алгоритма



IV. Программа

```
#include <iostream>
using namespace std;
int main()
{
    double c, k, S;
    cout<<"Введите цену товара"; cin>>c;
    cout<<"Введите количество"; cin>>k;
    S=c*k;
    if (S>1000) {
        S=S-0.05*S;
        cout<<"Стоимость товара со скидкой в 5%"<<S;
    }
    else
        cout<<"Скидки нет. Стоимость товара "<<S;
    return 0;
}
```

Пример сокращенной формы if

Известны цена и количество приобретаемого товара. Если общая стоимость покупки превышает 1000 руб., то на нее предоставляется скидка в 5% от этой стоимости. Вычислить и вывести стоимость данной покупки с учетом возможной скидки

I. Постановка задачи

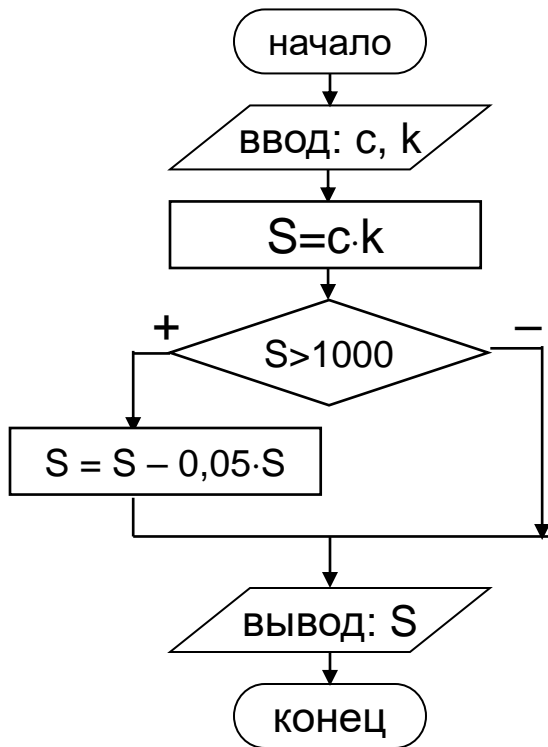
Дано: c , руб. - цена товара
 k , шт. – количество

Найти: S , руб. - стоимость

II. Математическая формулировка

$$S = \begin{cases} c \cdot k - 0,05c \cdot k, & \text{если } c \cdot k > 1000 \\ c \cdot k, & \text{иначе} \end{cases}$$

III. Блок-схема алгоритма



IV. Программа

```
#include <iostream>
using namespace std;
int main()
{
    double c, k, S;
    cout<<"Введите цену товара";
    cin>>c;
    cout<<"Введите количество товара";
    cin>>k;
    S=c*k;
    if (S>1000) {
        S=S-0.05*S;
    }
    cout<<"Стоимость товара с учетом скидки "<<S;
    return 0;
}
```

Пример использования вложенных операторов **if**

Известна координата точки на прямой x . Вывести сообщение о том, попала ли эта точка в интервал $[a, b]$, a и b – известные числа

I. Постановка задачи

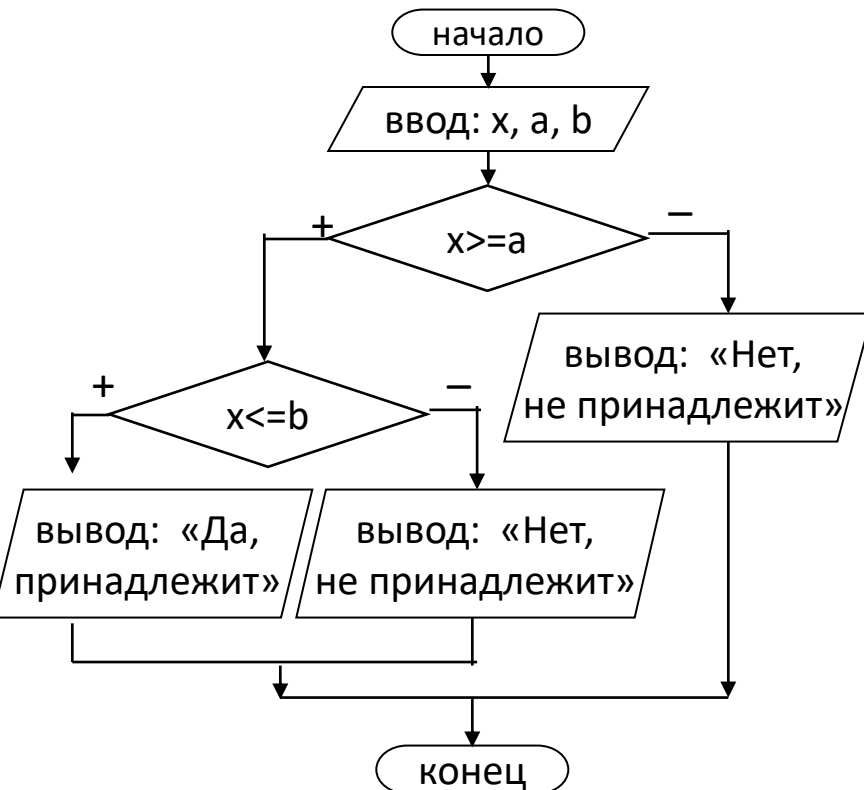
Дано: x

Найти: S – текстовое сообщение

II. Математическая формулировка

$$S = \begin{cases} \text{"Да, принадлежит интервалу",} & \text{если } a \leq x \text{ И } x \leq b \\ \text{"Нет, не принадлежит интервалу",} & \text{иначе} \end{cases}$$

III. Блок-схема алгоритма



IV. Программа

```
#include <iostream>
using namespace std;
int main()
{
    double x, a, b;
    cout << "Введите координату точки";
    cin >> x;
    cout << "Введите границы отрезка";
    cin >> a >> b;

    if (x >= a)
        if (x <= b)
            cout << "Да, принадлежит интервалу";
        else
            cout << "Нет, не принадлежит интервалу";
    else
        cout << "Нет, не принадлежит интервалу";

    return 0;
}
```

Логические операции И (&&), ИЛИ (||), НЕ (!)

используются для построения сложных условий на основе простых

&& (логическое И) истинно тогда и только тогда, когда оба операнда истинны

|| (логическое ИЛИ) истинно тогда и только тогда, когда хотя бы один операнд принимает значение истина

! (логическое НЕ) истинно тогда и только тогда, когда операнд принимает ложное значение

Простые условия		Результат применения операций		
A	B	A && B	A B	! A
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Примеры логических выражений с операциями И (&&), ИЛИ (||), НЕ (!)

Операция	1-ый операнд	2-ой операнд	Сложное условие
	его значение	его значение	его значение
&&	5<7 true	6>=3 true	5<7 && 6>=3 true
	5>=7 false	6>=3 true	5>=7 && 6>=3 false
	5>=7 false	6>=3 true	5>=7 6>=3 true
	5>=7 false	6<3 false	5>=7 6<3 false
!	5==5 true		!(5==5) false
	5==9 false		!(5==9) true

Пример использования операции Логическое И

Известна координата точки на прямой x . Вывести сообщение о том, попала ли эта точка в интервал $[a, b]$, a и b – известные числа

I. Постановка задачи

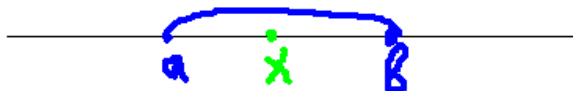
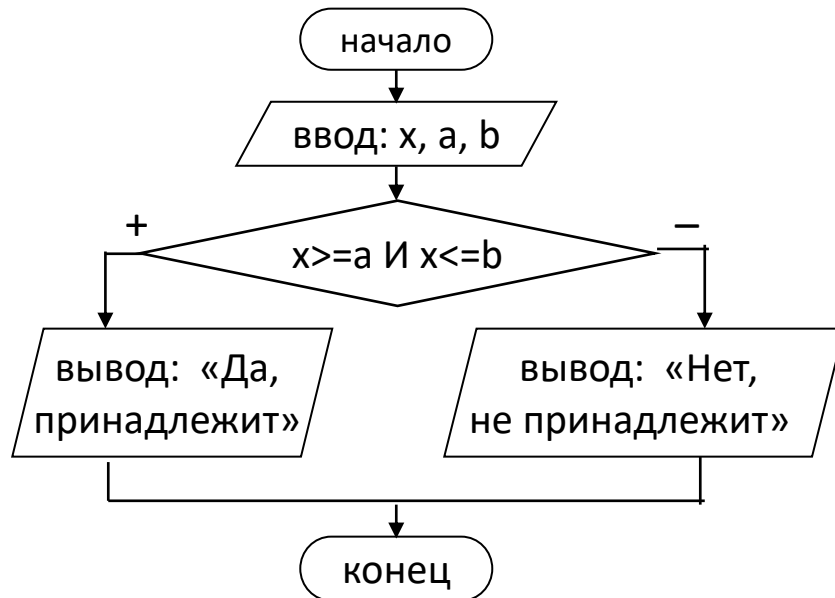
Дано: x

Найти: S – текстовое сообщение

II. Математическая формулировка

$$S = \begin{cases} \text{"Да, принадлежит интервалу",} & \text{если } a \leq x \text{ И } x \leq b \\ \text{"Нет, не принадлежит интервалу",} & \text{иначе} \end{cases}$$

III. Блок-схема алгоритма



IV. Программа

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

int main()
{
    double x, a, b;
    cout << "Введите координату точки";
    cin >> x;
    cout << "Введите границы отрезка";
    cin >> a >> b;

    if (x >= a && x <= b)
        cout << "Да, принадлежит интервалу";
    else
        cout << "Нет, не принадлежит интервалу";
    return 0;
}
```


Пример использования операции Логическое ИЛИ

Известна координата точки на прямой x . Вывести сообщение о том, попала ли эта точка в интервал $[a, b]$, a и b – известные числа

I. Постановка задачи

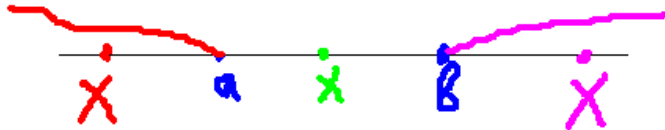
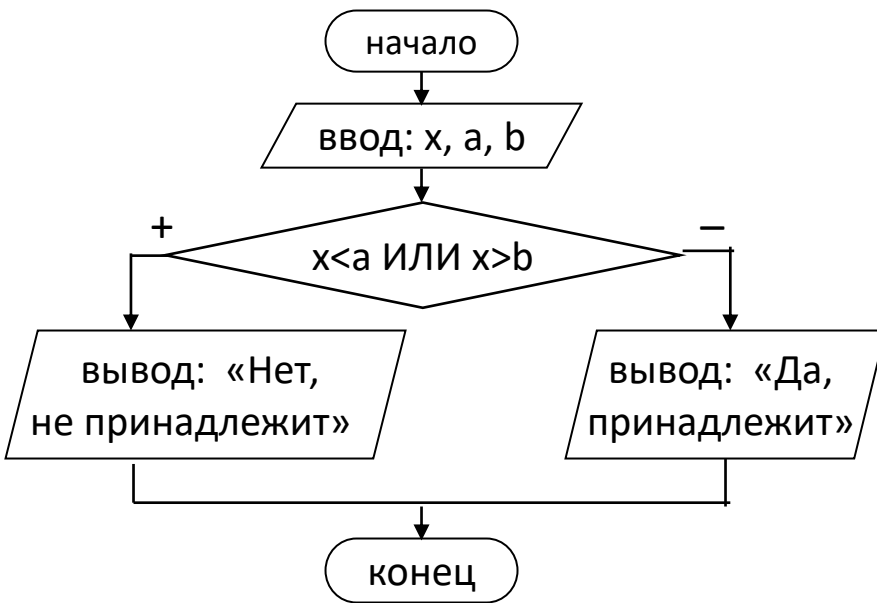
Дано: x

Найти: S – текстовое сообщение

II. Математическая формулировка

$$S = \begin{cases} \text{"Да, принадлежит интервалу",} & \text{если } a \leq x \text{ И } x \leq b \\ \text{"Нет, не принадлежит интервалу",} & \text{иначе} \end{cases}$$

III. Блок-схема алгоритма



IV. Программа

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

int main()
{
    double x, a, b;
    cout << "Введите координату точки";
    cin >> x;
    cout << "Введите границы отрезка";
    cin >> a >> b;

    if (x < a || x > b)
        cout << "Нет, не принадлежит интервалу";
    else
        cout << "Да, принадлежит интервалу";
    return 0;
}
```

Известна координата точки на прямой x.
Найти координату y, которая вычисляется как значение функции

$$y = f(x) = \begin{cases} x^2, & \text{если } x < 0 \\ 1\frac{2}{3}, & \text{если } 0 \leq x < 3 \\ \frac{1}{x+1}, & \text{иначе } (x \geq 3) \end{cases}$$

```
#include <iostream>
using namespace std;
int main()
{
    double x, y;
    cout << "Введите координату точки  x= ";
    cin >> x;

    if (x < 0)
        y = x * x;
    else
        if (x < 3)
            y = 1. + 2./3.;
        else
            y = 1. / (x + 1.);

    cout << "y = f(" << x << ")= " << y << endl;

    return 0;
}
```

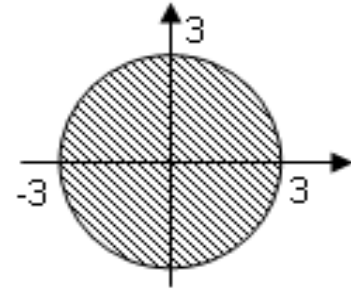
Известна координата точки на прямой x.
Найти координату y, которая вычисляется как значение функции

```
#include <iostream>
using namespace std;
int main()
{
    double x, y;
    cout << "Введите координату точки  x= ";
    cin >> x;
    if (x < 0)
    {
        y = x * x;
        cout << "y = f(" << x << ")= " << y << endl;
    }
    else
        if (x < 3.)
        {
            y = 1. + 2./3.;
            cout << "y = f(" << x << ")= " << y << endl;
        }
        else
            if (x != 4.)
            {
                y = 1. / (x - 4.);
                cout << "y = f(" << x << ")= " << y << endl;
            }
            else
                cout << "деление на 0, нет решения " << endl;
    return 0;
}
```

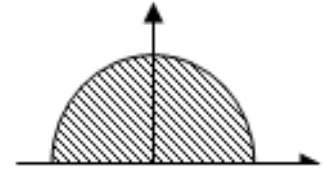
$$y = f(x) = \begin{cases} x^2, & \text{если } x < 0 \\ 1\frac{2}{3}, & \text{если } 0 \leq x < 3 \\ \frac{1}{x-4}, & \text{иначе (} x \geq 3 \text{)} \end{cases}$$

Примеры условий для проверки попадания точки в заданную область

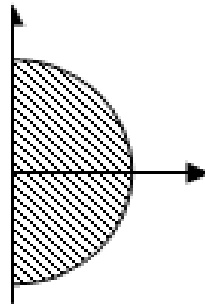
```
if (x*x + y*y<=9) cout<<"точка попала в область ";  
else cout<<"точка не попала в область ";
```



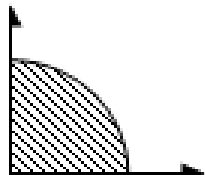
```
if (x*x + y*y<=9 && y>=0) cout<<"точка попала в область ";  
else cout<<"точка не попала в область ";
```



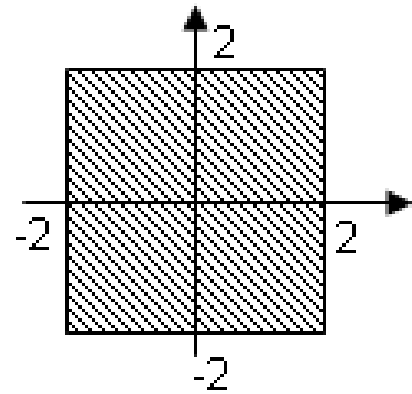
```
if (x*x + y*y<=9 && x>=0) cout<<"точка попала в область ";  
else cout<<"точка не попала в область ";
```



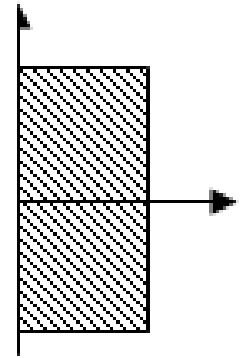
```
if (x*x + y*y<=9 && x>=0 && y>=0) cout<<"точка попала в область ";  
else cout<<"точка не попала в область ";
```



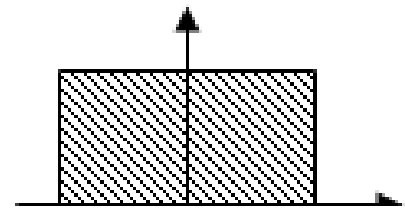
```
if (x<=2 && x>=-2 && y>=-2 && y<=2)
    cout<<"точка попала в область ";
else cout<<"точка не попала в область ";
```



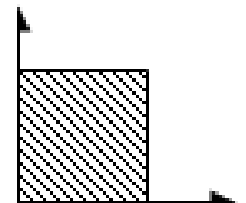
```
if (x<=2 && x>=0 && y>=-2 && y<=2)
    cout<<"точка попала в область ";
else cout<<"точка не попала в область ";
```



```
if (x<=2 && x>=-2 && y>=0 && y<=2)
    cout<<"точка попала в область ";
else cout<<"точка не попала в область ";
```



```
if (x<=2 && x>=-2 && y>=0 && y<=2)
    cout<<"точка попала в область ";
else cout<<"точка не попала в область ";
```



```
if (fabs(x)+fabs(y)<=2)
```

```
    cout<<"точка попала в область ";
```

```
else cout<<"точка не попала в область ";
```

или

```
if (y<=-x+2 && y>=x-2 && y>=-x-2 && y<=x+2)
```

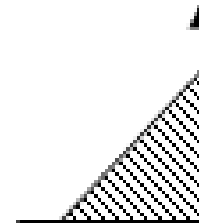
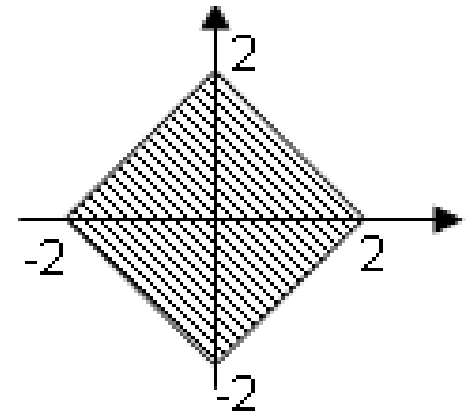
```
    cout<<"точка попала в область ";
```

```
else cout<<"точка не попала в область ";
```

```
if (y<=-x+2 && y>=0 && x<=0)
```

```
    cout<<"точка попала в область ";
```

```
else cout<<"точка не попала в область ";
```



Ввести оценку ученика по пятибальной системе, вывести подходящий текстовый комментарий.

Решение 1. – Множественный if

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

int main()
{
    int x; //переменная для хранения оценки
    cout << "Введите свою оценку: "; cin>>x;

    //цепочка if..else
    if (5 == x)    cout << "отлично \n";
    else if (4 == x)    cout << "хорошо \n";
    else if (3 == x)    cout << "удовлетворительно \n";
    else if (2 == x || 1 == x)    cout << "плохо \n";
    else cout << "неверные данные "<<"\n";

    return 0;
}
```

Многонаправленное ветвление. Оператор switch

```
switch (управляющее_выражение)
{
    case константа_1 :
        операторы_1;
        [ break; ]
    case константа_2 :
        операторы_2;
        [ break; ]

    ...

    case константа_n :
        операторы_n;
        [ break; ]
    [ default : операторы_по_умолчанию ]
}
```


Оператор switch. Правила синтаксиса

```
switch (управляющее_выражение)
{
    case константа_1 :
        операторы_1;
        [ break; ]

    ...

    case константа_n :
        операторы_n;
        [ break; ]
    [ default : операторы_по_умолчанию ]
}
```

1. *Управляющее_выражение*

должно быть целого типа, char, bool или константой типа enum

2. Среди *констант*, указанных после слов case

не должно быть одинаковых значений

3. Если одинаковые наборы операторов надо выполнять для нескольких разных значений *констант*, то эти константы надо указать друг за другом и только один раз записать операторы для них:

```
case константа_1 :
case константа_2 :
...
case константа_m :
    операторы
```

Работа оператора switch

```
switch (управляющее_выражение)
{
    case константа_1 :
        операторы_1;
        [ break; ]

    ...

    case константа_n :
        операторы_n;
        [ break; ]
    [ default : операторы_по_умолчанию ]
}
```

1. Вычисляется значение *управляющего_выражения*
2. Это значение последовательно сравнивается с *константами*, стоящими после **case**, если есть совпадение, то
 - 1) выполняются операторы, стоящие после совпавшей константы
 - 2) выполняются все операторы, стоящие ниже после остальных констант и слова **default** (независимо от того совпадают эти *константы* со значением *логического_выражения* или нет)
 - 3) выполнение операторов прекратится на операторе **break**;
если оператор **break**; не встретится ни разу, то будут выполнены все операторы до конца списка

Если значение *управляющего_выражения* не совпало ни с одной из *констант*, то выполняются только *операторы*, указанные после ключевого слова **default**

Ввести оценку ученика по пятибальной системе, вывести подходящий текстовый комментарий.

Решение 1. – switch

```
#include <iostream>
using namespace std;
int main()
{
    int x; //переменная для хранения оценки
    cout << "Введите свою оценку: ";
    cin>>x;
    //оператор множественного выбора
    switch (x)
    {
        case 5 : cout<<"отлично \n"; break;
        case 4 : cout<<"хорошо \n"; break;
        case 3 : cout<<"удовлетворительно \n"; break;

        case 2 :
        case 1 : cout<<"плохо \n"; break;
        default : cout<<"неверные данные "<<"\n";
    }

    return 0;
}
```