

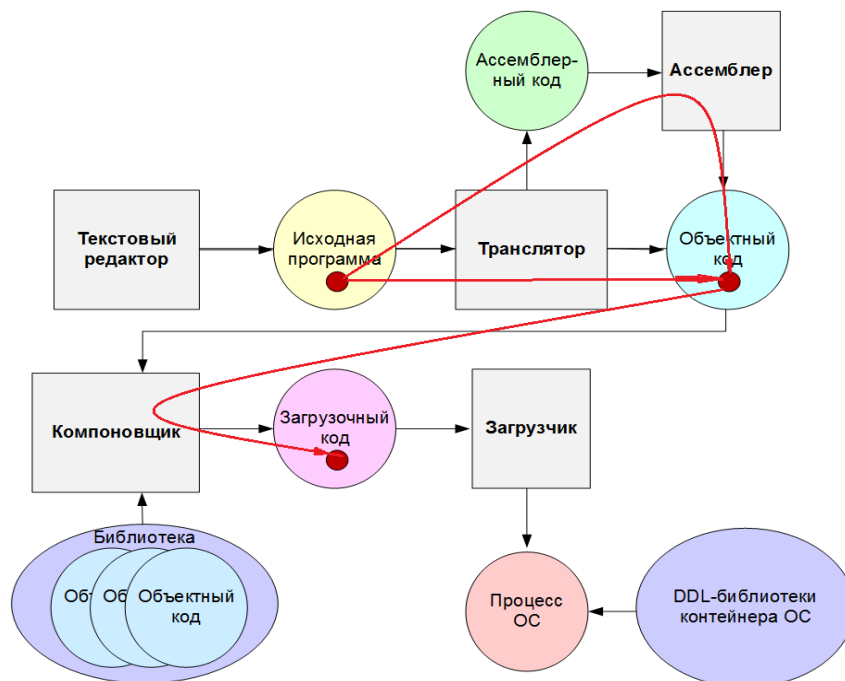
**Введение в разработку программного обеспечения (ИС и ЦД)**

**Современные системы программирования. Стили программирования.  
Структурное программирование. Модульное программирование**

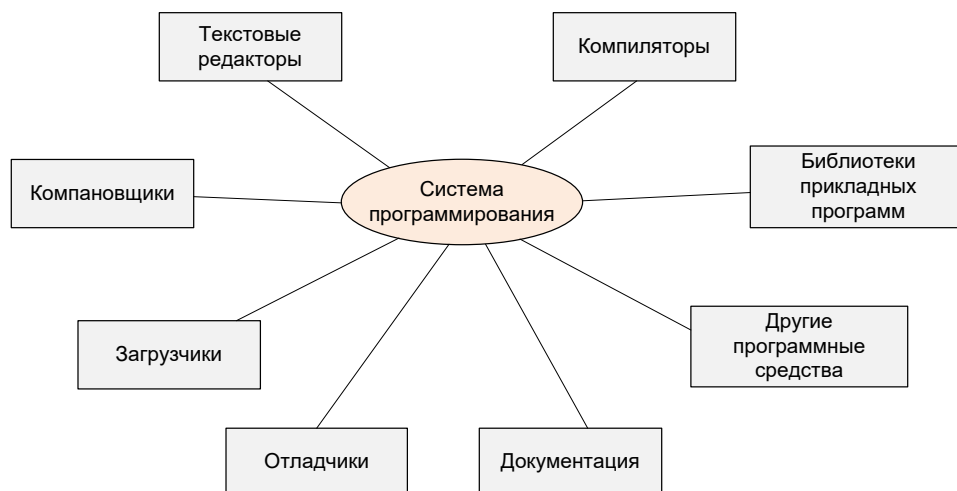
План лекции:

- система программирования, язык программирования;
- система программирования Windows;
- система программирования Linux (UNIX-подобная);
- стандарт POSIX;
- стандарты языков программирования;
- парадигмы программирования.
- парадигмы программирования;
- императивное программирование;
- декларативное программирование;
- структурное программирование.
- модульное программирование
- стили программирования (оформление кода).

**От исходного кода к исполняемому модулю:**



**Система программирования** – инструментальное ПО, предназначенное для разработки программного продукта на этапах программирования и отладки. Каждая система программирования должна иметь некоторый встроенный в нее язык программирования, предназначенный для общения разработчика с используемыми инструментами.



Система программирования является основным инструментом программиста.

## 1. Классическая система программирования

С развитием сервисных средств ОС появились командные процессоры, что позволило объединять последовательность вызовов системных программ в единые командные файлы. Это упростило работу по запуску компонентов систем.

Специализированные командные процессоры (*координатор make* или командный *интерпретатор shell*) представляли собой интерпретаторы, на вход которым подавались файлы, записанные на особом командном языке.

Ядром системы программирования является язык программирования.

## 2. Интегрированная среда разработки




**Интегрированная среда разработки (integrated development environment – IDE):** – набор инструментов для разработки и отладки программ, имеющий общую интерактивную графическую оболочку, поддерживающую выполнение всех основных функций жизненного цикла разработки программы.

**Примеры IDE:** Visual Studio, NetBeans, Eclipse, Embarcadero Delphi и пр.


**JDeveloper** — бесплатная интегрированная среда разработки ПО. Для разработки на языках программирования Java, JavaScript, BPEL, PHP, SQL, PL/SQL; на языках разметки HTML, XML.

	1998
разработчик	Oracle
написана на	Java
ОС	кроссплатформенная
платформа	Java Virtual Machine
последняя версия	12c (12.2.1.2.0) (19.10.2016)
сайт	<a href="http://www.oracle.com/technetwork/developer-tools/jdev/">http://www.oracle.com/technetwork/developer-tools/jdev/</a>


**NetBeans** — свободная интегрированная среда разработки ПО на языках программирования Java, Python, PHP, JavaScript, C, C++, Ада и др.

	1997
разработчик	Apache Software Foundation, Oracle и Sun Microsystems
написана на	Java
ОС	Microsoft Windows, Linux, macOS и Solaris
платформа	Java Virtual Machine
последняя версия	12.5 (13 сентября 2021)
сайт	<a href="http://netbeans.apache.org">netbeans.apache.org</a>


**Eclipse** — свободная интегрированная среда разработки модульных кроссплатформенных приложений.

	7 ноября 2001
разработчик	Eclipse Foundation
написана на	Java
ОС	GNU/Linux, macOS, Microsoft Windows, Solaris
платформа	Java Virtual Machine
последняя версия	4.21.0 (15 сентября 2021)
сайт	<a href="http://eclipse.org">eclipse.org</a>


**Oracle Sun Studio** — интегрированная среда разработки для языков программирования Си, С++ и Фортран

	
разработчик	Oracle Corporation
написана на	SPARC, x86, x86-64
ОС	Solaris, OpenSolaris, Linux
последняя версия	Oracle Developer Studio 12.6 (5 июля 2017 года)
сайт	<a href="http://www.oracle.com/technetwork/server-storage/solarisstudio/overview">http://www.oracle.com/technetwork/server-storage/solarisstudio/overview</a>

**Embarcadero Delphi** — интегрированная среда разработки ПО для Microsoft Windows, Mac OS, iOS и Android на языке Delphi (раньше Object Pascal). Embarcadero Delphi является частью пакета Embarcadero RAD Studio

		1995
разработчик	Embarcadero Technologies	
написана на	Delphi и Object Pascal	
ОС	Microsoft Windows	
последняя версия	10.4.2 Sydney (24.02.2021)	
сайт	<a href="http://embarcadero.com/ru/product">embarcadero.com/ru/product</a>	

**Microsoft Visual Studio** — линейка продуктов компании Microsoft, включающих интегрированную среду разработки ПО и другие инструменты. Для разработки консольных приложений, игр, приложений с графическим интерфейсом, веб-сайтов, веб-приложений, веб-служб как в нативном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

		1997
разработчик	Microsoft	
написана на	C++ и C#	
ОС	Microsoft Windows, macOS	
последняя версия	16.11.3 (14 сентября 2021)	
сайт	<a href="http://visualstudio.microsoft.com">visualstudio.microsoft.com</a>	

# Visual Studio 2019

Быстрое написание кода. Автоматизация работы. Интегрированная среда разработки будущего.

Полный набор инструментов для всех этапов разработки, от начального замысла до финального развертывания

- ✓ Оптимизация работы IntelliSense в файлах C++
- ✓ Локальная разработка с поддержкой множества популярных эмуляторов
- ✓ Упрощенный доступ к тестам в обозревателе решений

## Меньше ошибок при написании кода

Используйте рекомендации IntelliSense для быстрого и точного ввода нужных переменных при возникновении затруднений. Сохраняйте высокий темп работы вне зависимости от сложности за счет быстрого перехода к любому файлу, типу, элементу или символу. Используйте значки лампочек, которые рекомендуют действия по улучшению кода, например предлагают переименовать функцию или добавить параметр.

Все функции разработки >



Разработка



Анализ



Отладка



Тестирование



Управление версиями



Совместная работа



Развертывание

## ОС и поддерживаемые платформы

Windows	ARM, IA-64, Itanium, MIPS, DEC Alpha, PowerPC и x86	Microsoft
Linux	DEC Alpha, x86, x86_64, ARM, PowerPC, RISC-V и MIPS	Свободное программное обеспечение
macOS	Motorola 68k, PowerPC, IA-32, x86-64, ARM	Apple Inc.
OS/2	x86	IBM, Microsoft
Unix	Intel x86	
Solaris	SPARC, x86, x86-64, PowerPC	Sun Microsystems

### 3. Стандарты языков программирования

Стандарт языка программирования: Visual C++ 2017 версия 15.3 – это реализация стандарта C++17 или ISO/IEC 14882:2017.

Standards

ICS > 35 > 35.060

# ISO/IEC 14882:2017

## Programming languages — C++

THIS STANDARD HAS BEEN REVISED BY ISO/IEC 14882:2020

### ABSTRACT

ISO/IEC 14882:2017 specifies requirements for implementations of the C++ programming language. The first such requirement is that they implement the language, so this document also defines C++. Other requirements and relaxations of the first requirement appear at various places within this document.

C++ is a general purpose programming language based on the C programming language as described in ISO/IEC 9899:2011 Programming languages ? C (hereinafter referred to as the C standard). In addition to the facilities provided by C, C++ provides additional data types, classes, templates, exceptions, namespaces, operator overloading, function name overloading, references, free store management operators, and additional library facilities.

### GENERAL INFORMATION

Status :  Withdrawn	Publication date : 2017-12
Edition : 5	Number of pages : 1605
Technical Committee : ISO/IEC JTC 1/SC 22 Programming languages, their environments and system software interfaces	
ICS : 35.060 Languages used in information technology	

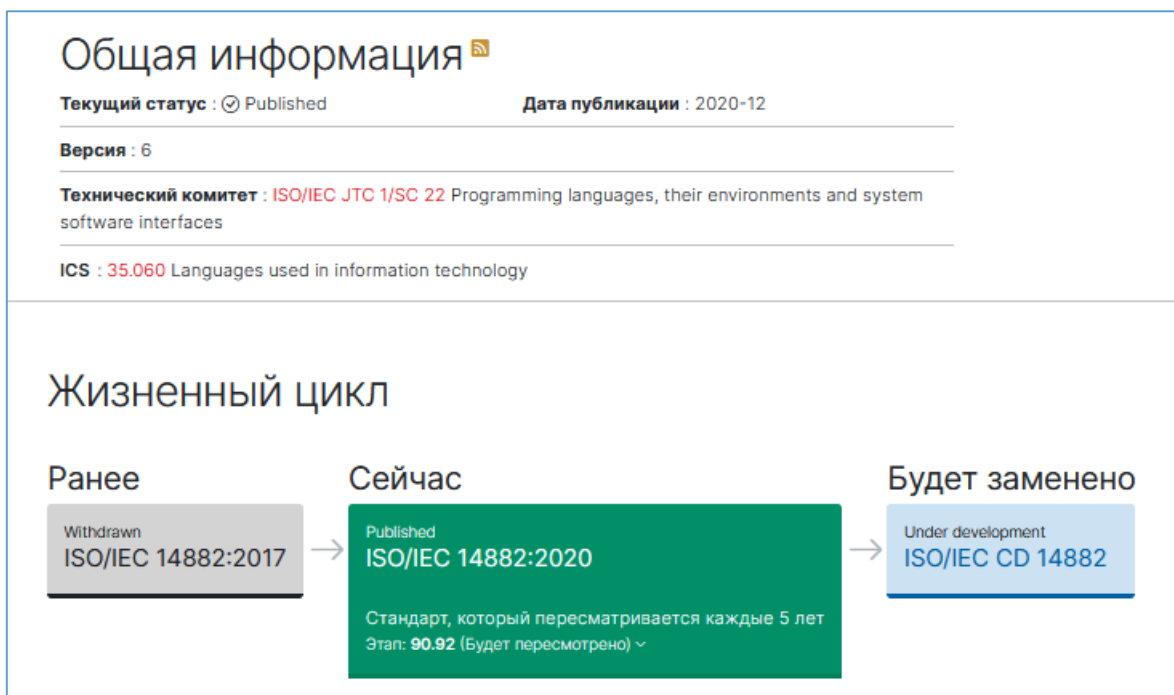
### LIFE CYCLE

PREVIOUSLY	NOW	REVISED BY
 WITHDRAWN ISO/IEC 14882:2014	 WITHDRAWN ISO/IEC 14882:2017 Stage: 95.99 ~	 PUBLISHED ISO/IEC 14882:2020




Международная организация по стандартизации (ИСО) одобрила C++ 20, последнюю версию объектно-ориентированного языка программирования. Официальный стандарт опубликован в конце 2020 года.



ISO/IEC 14882:2020  
Programming languages — C++




Новая версия C++ выходит примерно каждые три года, ей присваивают номер года. Стандарт языка C++ 20 является преемником C++ 17.


←  <https://www.ecma-international.org/publications/standards/Ecma-262.htm>

Most Visited    Телепрограмма на с...    Соглашение о вызов...    Mezzo - программа ...    intel Introduction to x64 As...    Ассембл...

 **Standards** 

 What is Ecma    Activities    News    Standards

[Standards Index](#)  
[Standards List](#)  
[Withdrawn Standards](#)  
[Tech. Reports Index](#)  
[Tech. Reports List](#)  
[Withdrawn Tech. Reports](#)  
[Mementos](#)

Printer Friendly Version  
 [Back](#)

## Standard ECMA-262

### ECMAScript® 2019 Language Specification

10<sup>th</sup> edition (June 2019)

---

This Standard defines the ECMAScript 2019 general-purpose programming language.

---

The following files can be freely downloaded:

File name	Size (Bytes)	Content
<a href="#">ECMA-262 edition 10</a>		Browsable HTML
<a href="#">ECMA-262.pdf</a>	14 423 437	Acrobat (r) PDF file

ECMAScript — это встраиваемый расширяемый язык программирования, основа для построения других скриптовых языков. Стандартизирован

международной организацией Ecma в спецификации ECMA-262. Расширениями языка являются JavaScript (Netscape), JScript (Microsoft) и ActionScript.

ECMAScript — это официальный стандарт языка JavaScript (Слово JavaScript не могло быть использовано, потому что слово Java является торговой маркой компании Sun)

Это восьмое издание спецификации языка ECMAScript.

Java Language and Virtual Machine Specifications

<https://docs.oracle.com/javase/specs/>



**ORACLE**

[Java SE](#) > Java SE Specifications

## Java Language and Virtual Machine Specifications

**Java SE 17**

Released September 2021 as [JSR 392](#)

 **The Java Language Specification, Java SE 17 Edition**

- [HTML](#) | [PDF](#)
- Preview feature: [Pattern Matching for switch](#)

 **The Java Virtual Machine Specification, Java SE 17 Edition**


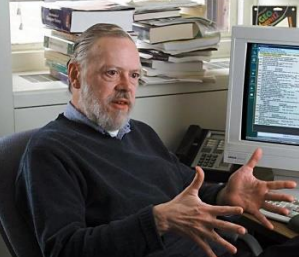
- [HTML](#) | [PDF](#)

#### 4. Системы программирования Windows и UNIX

**Unix** («UNIX» — зарегистрированная торговая марка The Open Group) — семейство переносимых, многозадачных и многопользовательских операционных систем, которые основаны на идеях оригинального проекта AT&T Unix, разработанного в 1970-х годах в исследовательском центре Bell Labs Кеном Томпсоном, Деннисом Ритчи и другими.

*Дизайнеры языков программирования:*

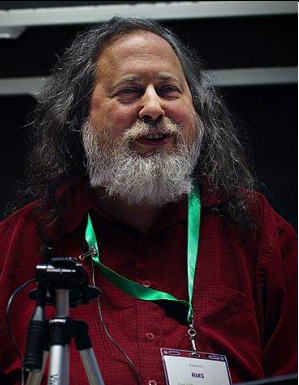


	<p>Кеннет Лейн Томпсон – пионер компьютерной науки, известен своим вкладом в создание языка программирования С и операционной системы UNIX. Написал язык программирования В, предшественник языка С, участвует в создании языка программирования Go</p>
	<p>Деннис Ритчи – создатель языка программирования Си. Вместе с Кеном Томпсоном разработал Си для создания операционной системы UNIX. «У Ньютона есть фраза о стоящих на плечах гигантов. Мы все стоим на плечах Денниса», – Брайан Керниган.</p>

Операционные системы семейства Unix характеризуются модульным дизайном, в котором каждая задача выполняется отдельной утилитой, взаимодействие осуществляется через единую файловую систему, а для работы с утилитами используется командная оболочка.

Unix является мультиплатформенной системой. Ядро системы разработано таким образом, что его легко можно приспособить практически под любой микропроцессор.

**Linux** (GNU/Linux) – семейство Unix-подобных операционных систем на базе ядра Linux, включающих тот или иной набор утилит и программ проекта **GNU** (проект по разработке свободного программного обеспечения, запущенный известным программистом и сторонником СПО Ричардом Столлманом 27 сентября 1983 года в Массачусетском технологическом институте).

	<p>Ричард Мэттью Столлман – основатель движения свободного программного обеспечения, проекта <b>GNU</b>, Фонда свободного программного обеспечения и Лиги за свободу программирования. Автор концепции «копилефта» (в противоположность традиционному подходу к авторскому праву, при котором ограничивается свобода копирования произведений, копилефт стремится использовать законы об авторском праве для расширения прав и свобод людей).</p>
---	---

Linux-системы реализуются на модульных принципах, стандартах и соглашениях, заложенных в Unix. Подобные системы используют монолитное ядро для управления процессами, сетевыми функциями, периферией и доступом к файловой системе. Драйверы устройств либо интегрированы непосредственно в ядро, либо добавлены в виде модулей, загружаемых во время работы системы.

Linux доминирует на рынке интернет-серверов и смартфонов (операционная система Android имеет в основе Linux ядро). Linux полностью бесплатная система, в основном построенная на открытом программном обеспечении.

### **Принципы взаимодействия программ в Windows.**

**Интерфейс вызовов функций в Windows** – доступ к системным ресурсам осуществляется через целый ряд системных функций. Совокупность таких функций называется прикладным программным интерфейсом или **API** (Application Programming Interface). Для взаимодействия с Windows приложение запрашивает функции API, с помощью которых реализуются все необходимые системные действия, такие как выделение памяти, вывод на экран, создание окон и т.п.

Функции API содержатся в библиотеках динамической загрузки (Dynamic Link Libraries, или DLL), которые загружаются в память только в тот момент, когда к ним происходит обращение, т.е. при выполнении программы.

### **Многозадачность в Windows**

В Windows два типа многозадачности: основанный на процессах и основанный на потоках.

Процесс – это программа, которая выполняется. При многозадачности такого типа две или более программы могут выполняться параллельно.

Поток – это отдельная часть исполняемого кода. В многозадачности данного типа отдельные потоки внутри одного процесса также могут выполняться одновременно.

## **5. Стандарт POSIX**

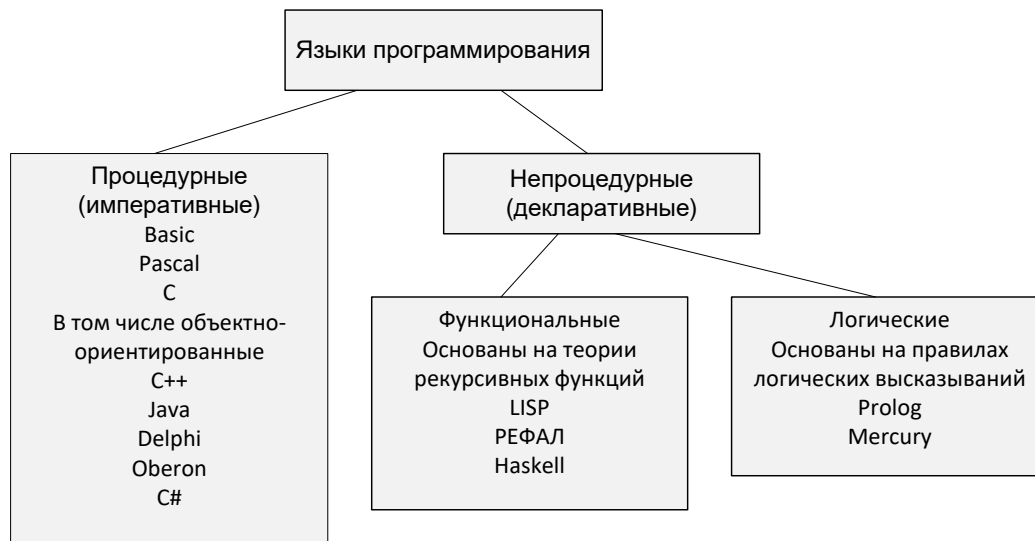
**Стандарт POSIX** – переносимый интерфейс операционных систем) — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой (системный API), библиотеку языка C и набор приложений и их интерфейсов.

Традиционно говорят о двух мирах, двух системах мировоззрения, присущих пользователям операционных систем Windows и UNIX.

В 1985 году принят стандарт POSIX. Это стандарт на интерфейсы UNIX-подобных ОС.

Однако никто не запрещает системам, не являющимся клонами (потомками) UNIX, поддерживать стандарт POSIX.

## 6. Парадигмы (стили) программирования



Язык программирования строится в соответствии с базовой моделью вычислений и парадигмой программирования (пример парадигмы: объектно-ориентированное программирование).

**Языки классифицируют по важнейшим признакам:**

- **эволюционным** – поколения языков (1GL, 2GL, 3GL, 4GL, 5GL, ...);
- **функциональным** – по назначению, исполняемым функциям (описательные, логические, математические);
- **уровню языка** – то есть уровню обобщения в словах-операторах языка (низкого, среднего, высокого, ...);
- **области применения** – системные, сетевые, встроенные и пр.

### Поколения языков (Generations of Languages)

Каждое из последующих поколений по своей функциональной мощности качественно отличается от предыдущего.

- 1GL - первое поколение: **Машинные языки**. Появились в середине 40-х годов XX века.
- 2GL - второе поколение: **Ассемблеры**. Фактически это те же машинные языки, но более красиво «обернутые». Появились в конце 50-х годов XX века
- 3GL - третье поколение: **Процедурные языки**. Появились в начале 60-х годов XX века. К этому поколению относят универсальные языки высокого уровня, с помощью которых можно решать задачи из любых областей (например, Algol-60).
- 4GL - четвертое поколение: **Языки поддержки сложных структур данных** (например, SQL). Появились в конце 60-х годов XX века.
- 5GL - пятое поколение: **Языки искусственного интеллекта** (например, Prolog). Появились в начале 70-х годов XX века.

- 6GL - шестое поколение: **Языки нейронных сетей** (самообучающиеся языки). Исследовательские работы в этой области начались в середине 80-х годов XX века.

Классификация языков по поддерживаемым методологиям появилась примерно в 80-х годах XX века

**Парадигма программирования** – это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию).

#### Императивное программирование:

Программа = последовательность действий, дающих указания компьютеру о том, **как** получить решение.

Конструкции языка:

последовательность действий;  
условная конструкция;  
цикл.

##### Пример:

Функция с именем `add`, принимает на вход массив и возвращает сумму всех его элементов.

##### Псевдокод:

```
function add(array)
{
  let result = 0
  for (let i = 0; i < array.length; i++)
    result += array[i];
  return result;
}
```

#### Декларативное программирование:

Программа = описание действий, которые необходимо выполнить компилятору для получения результата.

Отвечает на вопрос **что** надо выполнить.

##### Пример:

Функция с именем `add`, принимает на вход массив и возвращает сумму всех его элементов.

##### Псевдокод:

```
function add(array)
{
  return array.reduce((prev, current) => prev + current, 0)
}
```

#### Функциональное программирование:

Программа = система определений и функций, описывающих что нужно вычислить, а как это сделать – решает транслятор; последовательность действий не прослеживается.

Раздел дискретной математики. Основой функционального программирования является лямбда-исчисление

#### Объектно-ориентированное программирование:

Программа = несколько взаимодействующих объектов + функциональность (действия и данные распределяются между этими объектами).
<b>Распределённое (параллельное) программирование:</b> Программа = совокупность описаний процессов, которые могут выполняться как параллельно (при наличии нескольких процессоров), так и в псевдопараллельном режиме (при наличии одного процессора).
<b>Логическое программирование:</b> Программа = система определений вида «условие => новый факт». Программа представляет собой описание фактов и правил вывода в некотором логическом исчислении. Результат, (который часто записывается как вопрос), получается системой путем логического вывода. Раздел математической логики.
<b>Распределённое (параллельное) программирование:</b> Программа = совокупность описаний процессов, которые могут выполняться как параллельно (при наличии нескольких процессоров), так и в псевдопараллельном режиме (при наличии одного процессора).
<b>Визуальное программирование:</b> Программа = способ создания программы для ЭВМ путём манипулирования графическими объектами вместо написания её текста. Визуальное программирование позволяет программировать на уровне алгоритмов, а не программного кода. Пакет визуального программирования генерирует, написанный на языках программирования (1GL, 2GL, 3GL), на основании составленной программистом «блок-схемы» в автоматическом режиме.
<b>Аспектно-ориентированное программирование:</b> Программа = к уже существующему коду добавляется дополнительного поведение, так называемой сквозной функциональности.

**Пример** функции возведения в квадрат в некоторых языках программирования.

Императивный C:	Функциональный Scheme:
<pre>int square(int x) {     return x * x; }</pre>	<pre>(define square   (lambda (x)     (* x x)))</pre>
Конкатенативный Joy:	Конкатенативный Factor:
<pre>DEFINE square == dup * .</pre>	<pre>: square ( x -- y ) dup *;</pre>

## 7. TIOBE определила кандидатов на «язык года»

Голландская компания TIOBE Software BV известна как автор регулярно рассчитываемого рейтинга популярности языков программирования.

Индекс TIOBE — индекс, который оценивает популярность языков программирования, основываясь на количестве поисковых запросов, содержащих название языка. Расчет индекса происходит ежемесячно.

Очередной выпуск рейтинга языков программирования TIOBE показал стабильность среди лидеров списка

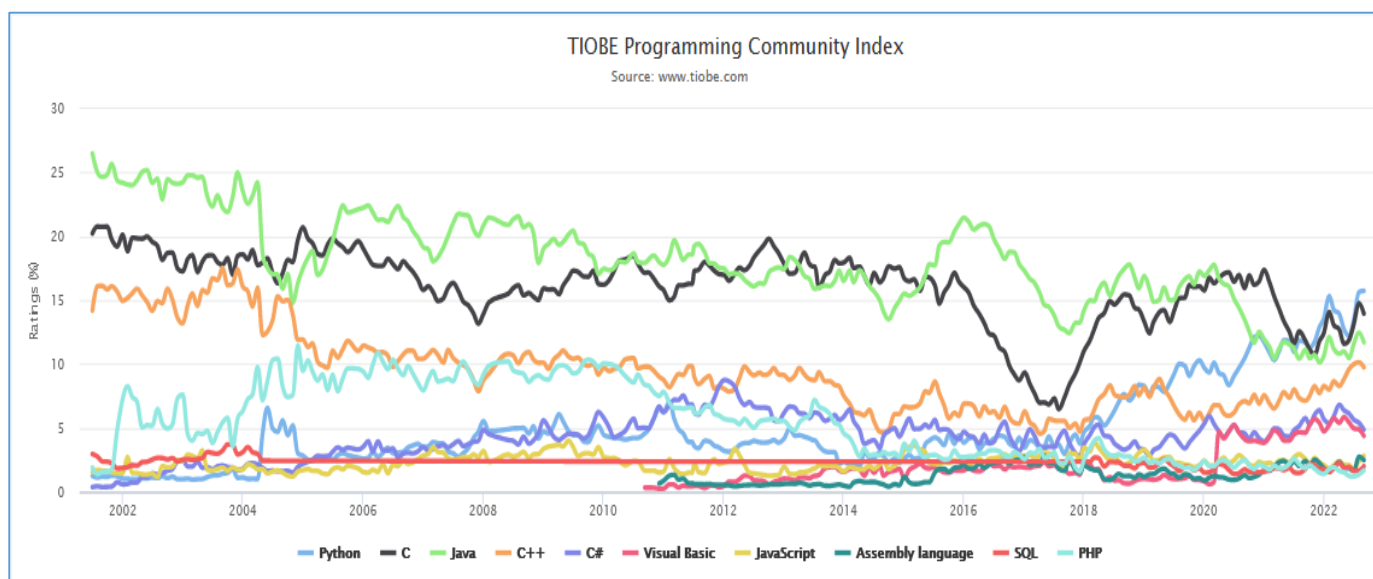


Иллюстрация: TIOBE, сентябрь 2022 года.

Четвёрка лидеров осталась неизменной: Python, C, Java и C++.

**!!** Языком 2020 и 2021 года является Python.

Заголовок сентября: Julia приближается к топ-20 индекса TIOBE

«Язык программирования Julia всего на 0,05% от топ-20. Язык Julia предназначен для численного анализа и вычислительной науки. В этой области существует множество конкурирующих языков: Так чем же выделяется Julia? Julia превосходит Matlab, потому что он намного современнее и его можно использовать бесплатно. Julia превосходит Python и R, потому что он намного быстрее. Поскольку спрос на работу с цифрами и модельным бизнесом огромен, у Julia есть серьезные шансы войти в топ-20 уже в ближайшем будущем. Отметим, что язык Rust тоже уже довольно давно стучится в двери топ-20, но пока безуспешно. Время покажет, постигнет ли Julia та же участь.» - Пол Янсен, генеральный директор TIOBE Software

**8. Неструктурное программирование** характерно для наиболее ранних языков программирования. Сложилось в середине 40-х с появлением первых языков программирования.

Основные признаки:

- строки как правило нумеруются;
- из любого места программы возможен переход к любой строке;

Выделяют четыре базовых стиля программирования.

1. Структурное программирование (действия и условия локальны);
2. Программирование от состояний (действия глобальны, условия локальны);
3. Программирование от событий (действия локальны, условия глобальны);
4. Сентенциальное программирование (действия и условия глобальны).

В программировании от *состояний* процесс представляется как смена состояний системы.

Новое состояние возникает в результате действия, изменяющего старое состояние.

Выбор этого действия зависит от проверки условий.

Математической моделью программы служит конечный автомат.

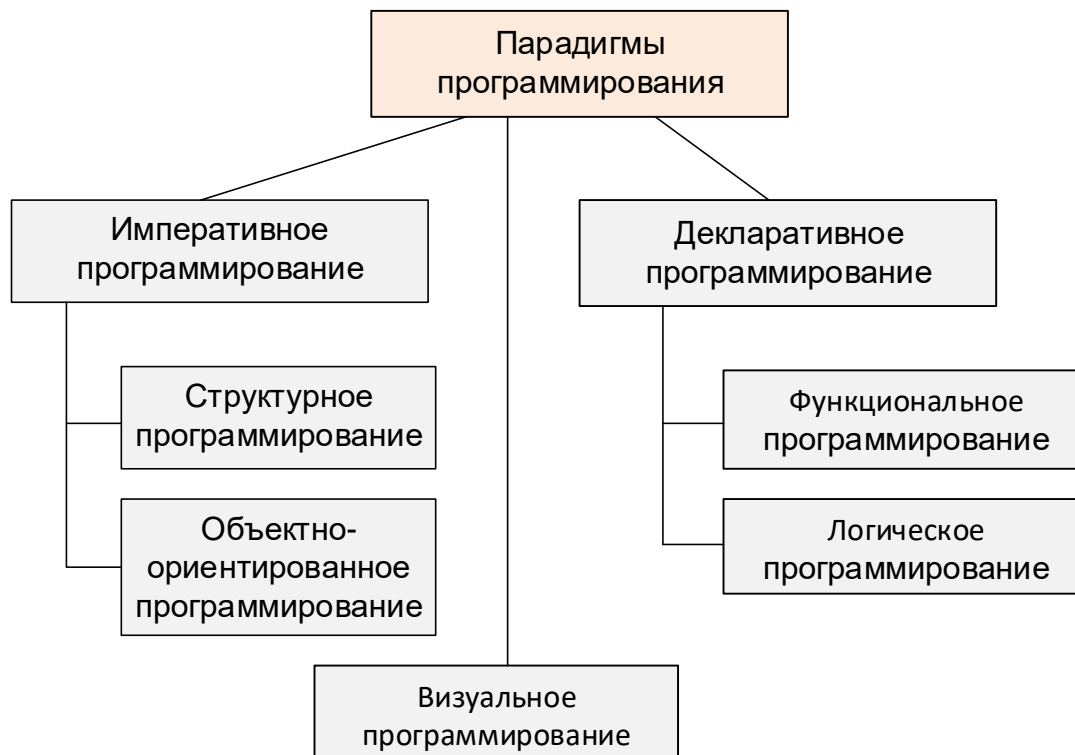
Разрешено использование оператора безусловного перехода `goto` либо использование объектов, обменивающихся информацией через общее поле памяти.

В *сентенциальном* стиле каждый шаг программы проверяет весь объем текстовой информации на соответствие образцу, находит и, согласно найденному правилу преобразования, производит преобразование.

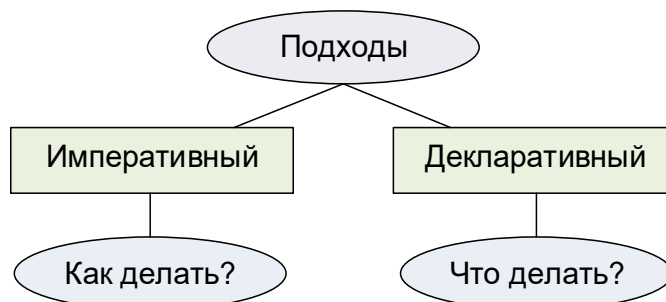
В программировании от *событий* условие состоит в том, что в системе произошло некоторое событие, которое влечет обработку данного действия обработчиком события.



## 9. Основные парадигмы программирования:



## Основные подходы к программированию:



**Императивное программирование** (от греч. *imperi* – действие) предполагает, что программа явно описывает алгоритм решения конкретной задачи (действия исполнителя), т.е. описывает как решать поставленную задачу.

**Декларативное программирование** (лат. *declaratio* – объявление) – это предварительная реализация «**решателя**» для целого класса задач.

Тогда для решения **конкретной задачи** этого класса достаточно декларировать в терминах данного языка только её условие:

(исходные данные + необходимый вид результата)

«**Решатель**» сам выполняет процесс получения результата, реализуя известный ему алгоритм решения.



## Структурное программирование

**Структурное программирование** – методология и технология разработки программных средств, основанная на трёх базовых конструкциях:

- следование;
- ветвление;
- цикл.

### **Цели структурного программирования:**

- повысить надежность программ (улучшить структуру программы);
- создание понятной, читаемой программы, которая выполняется, тестируется, конфигурируется, сопровождается и модифицируется без участия автора (создание ПП).

### **Принципы разработки:**

- программирование «сверху-вниз» (нисходящее программирование);
- модульное программирование с иерархическим упорядочением связей между модулями/подпрограммами «От общего к частному»

### **Этапы проектирования:**

- формулировка целей (результатов) работы программы;
- представление процесса работы программы (модель);
- выделение из модели фрагментов: определение переменных и их назначения, стандартных программных контекстов.

**Технология структурного программирования** базируется на следующих методах:

- нисходящее проектирование (формализация алгоритма «сверху вниз»: движение от общего к частному);
- пошаговое проектирование (нисходящая пошаговая детализация программы);
- структурное проектирование (замена формулировки алгоритма на одну из синтаксических конструкций – *последовательность*, *условие* или *цикл*; программирование без goto);
- одновременное проектирование алгоритма и данных (процессы детализации алгоритма и введение данных, необходимых для работы, идут параллельно);
- *модульное проектирование*;
- *модульное, нисходящее, пошаговое тестирование*.

**<цель\_результата> = <действие> + <цель\_результата вложенной конструкции>**

Программирование **без goto**.

Использование операторов (вида **continue, break, return**) для более изменения структурированной логики выполнения программы.

<b>императивное</b>	программа = последовательность действий, связанных условными и безусловными переходами
<b>процедурное</b>	программа = последовательность процедур, каждая из которых есть последовательность элементарных действий и вызовов процедур, структурированных с помощью структурных операторов ветвления и цикла
<b>объектно-ориентированное</b>	программа = несколько взаимодействующих объектов, функциональность (действия) и данные распределяются между этими объектами
<b>функциональное</b>	программа = система определений функций, описание того, что нужно вычислить, а как это сделать – решает транслятор; последовательность действий не прослеживается
<b>логическое</b>	программа = система определений и правил вида «условие => новый факт»
<b>сентенциальное</b>	программа = система правил вида «шаблон => трансформирующее действие»
<b>событийное</b>	программа = система правил вида «событие => новые события» + диспетчер событий
<b>автоматное</b>	программа = конечный автомат или автомат специального типа

### Типы входных данных в различных парадигмах

Тип входных данных	Парадигмы программирования
аргументы	функциональное
глобальные	автоматное, сентенциальное, продукционное
локальные	(создание временных локальных объектов используется во многих стилях)
события	событийное

### Организация хранилища

Парадигмы программирования	структура хранимых данных	тип императива
автоматное	<i>состояние</i>	диаграмма переходов
логическое	<i>факты</i> (n-арные отношения)	правила вида (логическое условие → новые факты)
сентенциальное	<i>текст</i> (произвольные данные, обычно записанные на некотором формальном языке)	правила вида (шаблон → трансформация)

**Пример:** телефон имеет следующие режимы (макро-состояния):

- ожидание звонка
- входящий звонок
- установлено соединение (идет разговор)
- просмотр телефонной книжки
- навигация по меню

Поток **входных данных** — это последовательность нажимаемых клавиш и принимаемые (в параллельном режиме) телефоном сигналы.

**Глобальными** переменными (памятью) являются все настройки, адресная книга, списки вызовов, SMS сообщения, флаги (информация) о пропущенных вызовах или полученных SMS сообщениях и др.

[Множество **состояний** телефона] =

[Множество режимов] × [Множество состояний памяти].

Проектирование системы заключается в следующем:

- выделение базовых сущностей (функций, объектов, состояний);
- установление взаимных связей (распределении ответственности – функциональности) между этими сущностями;
- определение, какие данные в каких функциях (объектах, состояниях) будут видны (доступны для чтения и модификации).

**Императивное программирование** — программирование от «глаголов». Программы представляют собой последовательность действий с условными и безусловными переходами.

Программист *мыслит* в терминах действий и выстраивает последовательности действий в более сложные макро-действия (процедуры).

**Пример:**

Procedure Вскипятить\_чайник

begin

    Зажечь плиту;

    Взять чайник;

    Налить в чайник воды;

    Поставить на плиту;

    Подождать 5 минут;

end

begin

    if Чайник не пуст then

        Вylить из чайника воду;

        Вскипятить\_чайник;

end.

Базовые признаки языка программирования и стиля:

- тип локализации (видимости) данных:
  - данные глобальны (Global);
  - данные локальны (Local);
  - данные передаются как аргументы функции (Flow);
  - данные отсутствуют (NONE);
- структура хранимых данных:
  - хранилище фактов;
  - хранилище объектов (переменные, структуры данных, объекты);
  - состояние;
- принцип доступа к данным:
  - именованный;
  - адресный;
  - сложный (по запросам):
    - SQL (Relational DBs);
    - Logical queries (Prolog);
  - Линейный:
    - FIFO (очередь);
    - FILO (стек);
- метод описания логики работы программы:
  - действия + условные переходы;
  - математические зависимости функций друг от друга;
  - диаграмма переходов;
  - правила вида «логическое условие → изменение данных»;
  - правила вида «шаблон → изменение данных»;
- принцип взаимной организации императивных и декларативных данных:
  - инкапсуляция в одном (объектно-ориентированный подход);
  - декларативных данных как таковых нет, есть аргументы, поступающие на вход функциям (функциональный подход);
  - простая логика действий привязана к состояниям глобальных данных;
  - ....

Программировать в различных стилях программирования можно в рамках одного о того же языка.

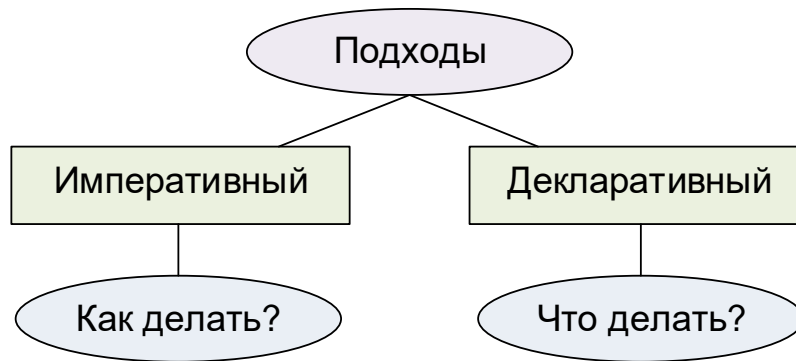


Коррадо Бём

Концепция структурного программирования основана на теореме Бёма-Якопини: любая вычислимая функция может быть представлена комбинацией трёх управляющих структур:

- последовательности действий;
- ветвления;
- повторения (итерации).

## Основные подходы к программированию:



**Императивное программирование** (от греч. *impeo* — действие) предполагает, что программа явно описывает алгоритм решения конкретной задачи (действия исполнителя), т.е. описывает как решать поставленную задачу.



В основе императивного программирования находятся два основных понятия:

- алгоритм (отсюда название алгоритмические языки)
- архитектура ЭВМ, основанная на модели вычислений фон Неймана.

Допускается выполнение действий над данными определённого типа.

- используются допустимые для данного типа операции и функции:
  - арифметические;
  - логические;
  - символьные;
  - строковые.

Управление ходом исполнения алгоритма.

- Используются операторы, реализующие основные управляющие структуры:
  - следование;
  - ветвление;
  - цикл;
  - вызов подпрограммы (возможно).

**Декларативное программирование** (лат. *declaratio* – объявление, подход возник в 60-х годах) – это предварительная реализация «решателя» для целого класса задач.

Тогда для решения конкретной задачи этого класса достаточно декларировать в терминах данного языка только её условие:

(исходные данные + необходимый вид результата)

«Решатель» сам выполняет процесс получения результата, реализуя известный ему алгоритм решения.

**Пример.** Найти сумму элементов массива.

**Псевдокод:**

```
// императивная парадигма
let array = [1, 2, 3, 4, 5, 6];
let result = 0
for (let i = 0; i < array.length; i++)
    result += array[i];
```

**Псевдокод:**

```
// декларативная парадигма
let array = [1, 2, 3, 4, 5, 6];
array.reduce((prev, current) =>
    prev + current, 0)
```

В **императивной** парадигме разработчик пишет для компьютера инструкции, которым тот следует. Инструкции будут примерно следующие:

- определить массив `array` из 6 элементов с заданными значениями;
- определить переменную `result` и присвоить ей значение 0;
- определить индекс цикла `i` со значением 0;
- начало цикла;
- добавить к переменной `result` элемент массива `array[i]`;
- прибавить к переменной `i` единицу;
- повторять цикл, пока значение переменной `i` меньше количества элементов массива `array`;
- конец цикла;

То есть, программист говорит, что нужно сделать (программирование от глаголов) и в каком порядке, а компьютер выполняет приказы.

В **декларативной** парадигме программист просто пишет следующее:

- дан массив `array` из 6 элементов с заданными значениями;
- получить сумму элементов массива `array`.

Приведенный в примере метод используется для последовательной обработки каждого элемента массива с сохранением промежуточного результата. Здесь `prev` – последний результат вызова функции (промежуточный результат). `current` – текущий элемент массива, элементы перебираются по очереди слева-направо.

При первом запуске `prev` – исходное значение, с которого начинаются вычисления и оно равно нулю.

## Структурное программирование



Термин структурное программирование ввёл Э.Дейкстра в 1975 году:

- Э.Дейкстра «Заметки по структурному программированию» (в составе сборника «Структурное программирование» / М.: Мир, 1975.

Структурное программирование – стиль написания программ без *goto*.

В структурном программировании необходимо:

составить правильную логическую схему программы;

реализовать ее средствами языка программирования;

Программа – множество вложенных блоков (иерархия блоков), каждый из которых имеет один вход и один выход.

Передача управления между блоками и операторами внутри блока (на каждом уровне дерева) выполняется последовательно.

**Технология структурного программирования** – нисходящее проектирование, т.е. от общего к частному, от внешней конструкции к внутренней.

**Теоретическим фундаментом структурного программирования** является теорема о структурировании Бёма-Якопини.

Структурное программирование характеризуется:

- ограниченным использованием условных и безусловных переходов;
- широким использованием подпрограмм и управляющих структур (циклов, ветвлений, и т.п.);
- блочной структурой.

Любая вычислимая конструкция может быть представлена комбинацией трёх управляющих структур:

- последовательности;
- ветвления;
- повторения (итерации).

Последовательность – это последовательное выполнение инструкций/блоков.

Ветвление – это выполнение либо одной, либо другой инструкции/блока в зависимости от значения некоего булева (логического) выражения.

Итерация – это многократное выполнение инструкции/блока пока некое булево выражение истинно.

**Структурное программирование** – методология и технология разработки программных средств, основанная на трёх базовых конструкциях:

- следование;
- ветвление;
- цикл.

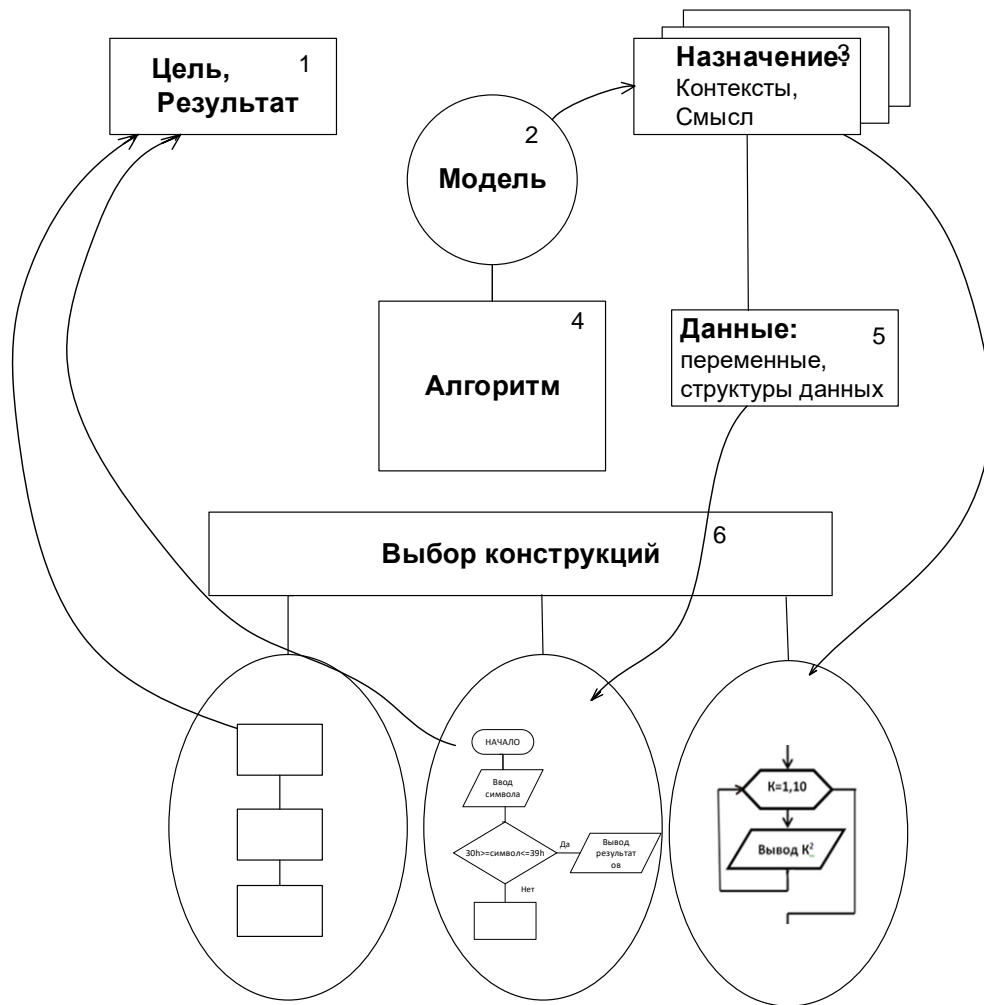
**Принципы разработки:**

- программирование «сверху-вниз» (нисходящее программирование);
- модульное программирование с иерархическим упорядочением связей между модулями/подпрограммами «От общего к частному»

Этапы проектирования:

- формулировка целей (результатов) работы программы;
- представление процесса работы программы (модель);
- выделение из модели фрагментов: определение переменных и их назначения, стандартных программных контекстов.

## Этапы структурного проектирования



1. Исходным состоянием процесса проектирования является формулировка цели алгоритма, или результата, который должен быть получен при его выполнении. Формулировка производится на естественном языке.
2. Создается модель происходящего процесса, используются графические или другие способы представления (например, образные изображения, позволяющие лучше понять выполнение алгоритма в динамике).
3. Выполняется сбор информации, касающейся характеристик алгоритма. Например, наличие определенных переменных и их «смысл», а также соответствующие им программные контексты.
4. В модели выделяется наиболее существенная часть, для которой строится алгоритм (в одной из форм представления).
5. Определяются переменные, необходимые для формального представления алгоритма и формулируется их назначение.
6. Выбирается одна из конструкций - *простая последовательность действий*, *условная конструкция* или *цикл*.
7. Для вложенных конструкций необходимо вернуться на предыдущие этапы проектирования.



**Технология** структурного программирования базируется на следующих методах:

- нисходящее проектирование;
- пошаговое проектирование;
- структурное проектирование (программирование без goto);
- одновременное проектирование алгоритма и данных;
- *модульное проектирование;*
- *модульное, нисходящее, пошаговое тестирование.*

**Нисходящее проектирование** программы состоит в процессе формализации от самой внешней синтаксической конструкции алгоритма к самой внутренней; в движении от общей формулировки алгоритма к частной формулировке, составляющей его действия;

**Структурное проектирование** заключается в замене словесной формулировки алгоритма на одну из синтаксических конструкций – *последовательность, условие* или *цикл*.

Использование оператора безусловного перехода goto запрещается из принципиальных соображений;

**Пошаговое проектирование** состоит в том, что на каждом этапе проектирования в текст программы вносится только одна конструкция языка.

**Одновременное проектирование алгоритма и структур данных.** При нисходящей пошаговой детализации программы необходимые для работы структуры данных и переменные появляются по мере перехода от неформальных определений к конструкциям языка, то есть процессы детализации алгоритма и данных идут параллельно.

**Цель (результат) = действие + цель (результат) вложенной конструкции.**

**Последовательность действий, связанных результатом.**

- представить действие в виде последовательности шагов;
- между различными шагами существуют связи через общие переменные, результат выполнения шага и последующие шаги используют этот результат.

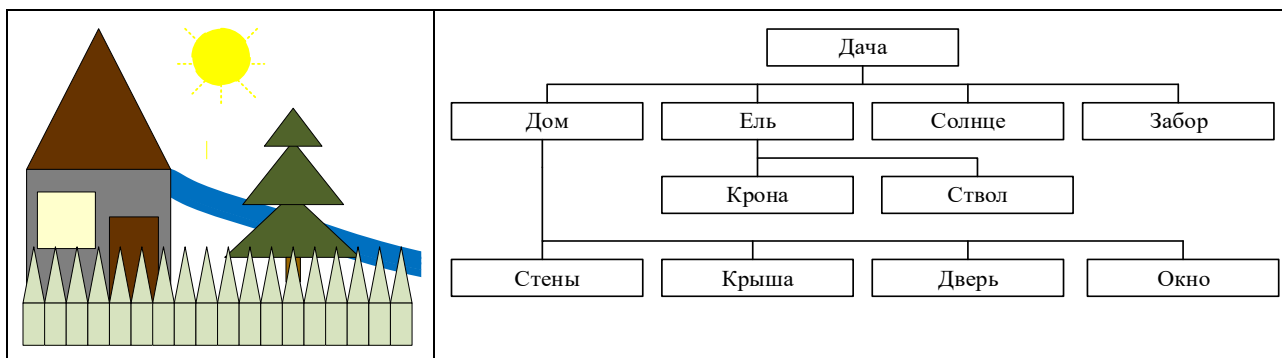
**Программирование без goto.**

**Операторы continue, break и return:** служат для более «мягкого» нарушения структурированной логики выполнения программы:

- **continue** – переход завершающую часть цикла;
- **break** – выход из внутреннего цикла;
- **return** – выход из текущего модуля (функции).

## 10. Модульное программирование

«Разделяй и властвуй» - латинская формулировка принципа, лежащего в основе модульного проектирования.



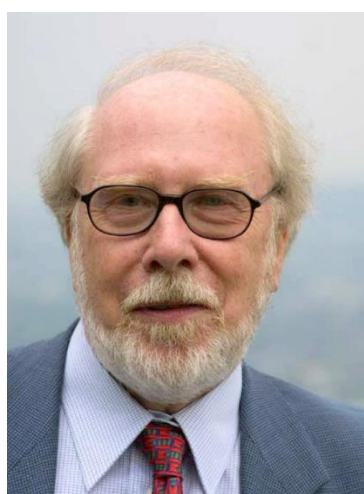
**Модульное программирование** – это организация программы как совокупности небольших независимых блоков, называемых модулями.

**Модуль** – функционально законченный фрагмент программы, оформленный в виде отдельного файла с исходным кодом.

**Функциональная декомпозиция задачи** – разбиение большой задачи на ряд более мелких, функционально самостоятельных подзадач – модулей.

Каждый модуль в функциональной декомпозиции представляет собой «черный ящик» с одним входом и одним выходом.

Модуль – это фрагмент описания процесса, оформленный как самостоятельный программный продукт, пригодный для многократного использования.



Профессор Никлаус Вирт

Никлаус Вирт – швейцарский учёный, специалист в области информатики, один из известнейших теоретиков в области разработки языков программирования, профессор компьютерных наук Швейцарской высшей технической школы Цюриха, лауреат премии Тьюринга

Создатель и ведущий проектировщик языков Pascal, Modula-2, Oberon, участвовал в разработке многих языков программирования.

Разработал язык Modula (1975 г.), в котором реализовал идеи модульного программирования с хорошо определенными межмодульными интерфейсами и параллельного программирования.

**Цели модульного программирования:** уменьшить сложность программ; предотвратить дублирование кода, упростить тестирование программы и обнаружение ошибок.

В языках высокого уровня, как правило, реализация данного механизма выполняется путем разделения модулей на интерфейсную и реализующую части.

**Пример** проекта, состоящего из основного файла и одного модуля на C/C++:

**Главный модуль** (требуется подключить интерфейс модуля)

```
1  #include "hello.h"
2  int main()
3  {
4      hello();
5      return 0;
6  }
```

**Заголовочный файл модуля hello**  
(интерфейс - заголовочный файл hello.h)

```
1  #include <iostream>
2  void hello();
```

**Файл реализации модуля hello**  
(файл с реализацией hello.cpp)

```
1  #include "hello.h"
2  void hello()
3  {
4      printf("Hello!\n");
5  }
```

**Плюсы модульного программирования:**

- ускорение разработки (позволяет изменять реализацию функциональности модуля, не затрагивая при этом взаимодействующие с ним модули);
- повышение надежности (локализует влияние потенциальных ошибок рамками модуля);
- упрощение тестирования и отладки;
- взаимозаменяемость.

**Минусы модульного программирования:**

- модульность требует дополнительной работы программиста и определенных навыков проектирования программ.

**Модуль, основные характеристики:**

- один вход и один выход (на вход программный модуль получает набор исходных данных, выполняет их обработку и возвращает набор выходных данных);
- функциональная завершенность (модуль выполняет набор определенных операций для реализации каждой отдельной функции, достаточных для завершения начатой обработки данных);
- логическая независимость (результат работы данного фрагмента программы не зависит от работы других модулей);
- слабые информационные связи с другими программными модулями (обмен информацией между отдельными модулями должен быть минимален);
- размер и сложность программного элемента должна быть в разумных рамках.

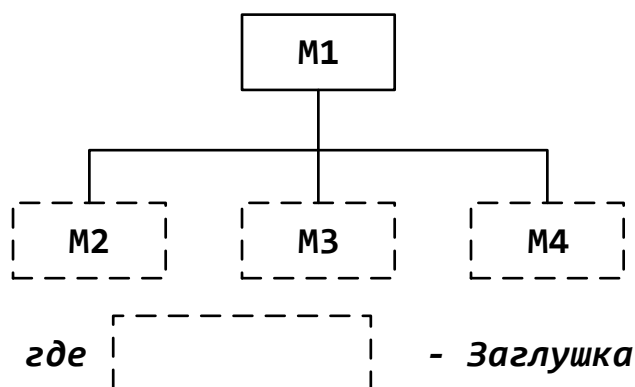
**Программный модуль** является самостоятельным программным продуктом. Это означает, что каждый программный модуль разрабатывается, компилируется и отлаживается отдельно от других модулей программы.

Роль модулей могут играть структуры данных, библиотеки функций, классы, сервисы и другие программные единицы, реализующие некоторую функциональность и предоставляющие интерфейс к ней.

*Технология модульного программирования* базируется на следующих методах:

- методы нисходящего проектирования (назначение – декомпозиция большой задачи на меньшие так, чтобы каждую подзадачу можно было рассматривать независимо.);
- методы восходящего проектирования.

*Нисходящее проектирование программ и его стратегии.*



На первом этапе разработки кодируется, тестируется и отлаживается головной модуль, который отвечает за логику работы всего программного комплекса. Остальные модули заменяются заглушками, имитирующими работу этих модулей.

На последних этапах проектирования все заглушки постепенно заменяются рабочими модулями.

### ***Стратегии, на которой основана реализация:***

- пошаговое уточнение (данная стратегия разработана Е. Дейкстрой);
- анализ сообщений (данная стратегия базируется на работах группы авторов: Иодана, Константайна, Мейерса).

***Пример.*** Стратегия, основанная на использовании псевдокода.

Способы реализации пошагового уточнения.

1. Кодирование программы с помощью псевдокода и управляющих конструкций структурного программирования;
2. Использование комментариев для описания обработки данных. Пошаговое уточнение требует, чтобы взаимное расположение строк программы обеспечивало читабельность всей программы.

### ***Преимущества*** метода пошагового уточнения:

- основное внимание при его использовании обращается на проектирование корректной структуры программ, а не на ее детализацию;
- так как каждый последующий этап является уточнением предыдущего лишь с небольшими изменениями, то легко может быть выполнена проверка корректности процесса разработки на всех этапах.

### ***Недостаток*** метода пошагового уточнения:

- на поздних этапах проектирования может возникнуть необходимость в структурных изменениях, которые повлекут за собой пересмотр более ранних решений.

### ***Использование псевдокода.***

#### ***Пример.***

Пусть программа обрабатывает файл с датами.

Необходимо отделить правильные даты от неправильных, отсортировать правильные даты, определить летние даты и вывести их в выходной файл.

***Первый этап*** пошагового уточнения: ***задается*** заголовок программы, соответствующий ее основной функции.

## **Program** Обработка\_дат

**Второй этап** пошагового уточнения: *определяются* основные действия.

```
Program Обработка_дат;  
    Отделить_правильные_даты_от_неправильных {*}  
    Обработать_неправильные_даты;  
    Сортировать_правильные_даты;  
    Выделить_летние_даты;  
    Вывести_летние_даты_в_файл;  
EndProgram.
```

**Третий этап** пошагового уточнения: *детализация* фрагмента \*.

```
Program Обработка_дат;  
    While не_конец_входного_файла  
    Do  
        Begin  
            Прочитать_дату;  
            Проанализировать_правильные|неправильные_даты;  
        End  
        Обработать_неправильные_даты;  
        Сортировать_правильные_даты;  
        Выделить_летние_даты;  
        Вывести_летние_даты_в_файл;  
    EndProgram.
```

и так далее.

### **Метод восходящей разработки**

При восходящем проектировании разработка идет снизу-вверх.

На первом этапе разрабатываются модули самого низкого уровня.

На следующем этапе к ним подключаются модули более высокого уровня и проверяется их работоспособность.

На завершающем этапе проектирования разрабатывается головной модуль, отвечающий за логику работы всего программного комплекса.

Методы нисходящего и восходящего программирования имеют свои преимущества и недостатки.

### **Недостатки нисходящего проектирования:**

- необходимость заглушек;
- до самого последнего этапа проектирования неясен размер всего программного комплекса и его характеристики, которые определяются только после реализации модулей самого низкого уровня.

**Преимущество нисходящего проектирования** – на самом начальном этапе проектирования отлаживается головной модуль (логика программы).

**Недостаток восходящего программирования** – головной модуль разрабатывается на завершающем этапе проектирования, что порой приводит к необходимости дорабатывать модули более низких уровней.

**Преимущество восходящего программирования** – не нужно писать заглушки.

На практике применяются оба метода. Метод нисходящего проектирования чаще всего применяется при разработке нового программного комплекса, а метод восходящего проектирования – при модификации уже существующего комплекса.

## 11. Стандарт оформления кода

**Стандарт оформления кода** – набор правил и соглашений, используемых при написании исходного кода на некотором языке программирования.

Стандарт оформления кода (или стандарт кодирования, или стиль программирования) (англ. coding standards, coding convention или programming style).

Это набор соглашений, который принимается и используется некоторой группой разработчиков программного обеспечения для единообразного оформления совместно используемого кода.

**Целью** принятия и использования стандарта является упрощение восприятия программного кода человеком, минимизация нагрузки на память и зрение при чтении программы.

**Стандарт оформления кода описывает:**

- способы выбора названий и используемый регистр символов для имен переменных и других идентификаторов:
  - запись типа переменной в ее идентификаторе;
  - регистр символов (нижний, верхний, «верблюжий», «верблюжий» с малой буквы), использование знаков подчёркивания для разделения слов;
- стиль отступов при оформлении логических блоков – используются ли символы табуляции, ширина отступа;
- способ расстановки скобок, ограничивающих логические блоки;
- использование пробелов при оформлении логических и арифметических выражений;
- стиль комментариев и использование документирующих комментариев.

**Вне стандарта подразумевается:**

- отсутствие магических чисел;
- ограничение размера кода по горизонтали (чтобы помещался на экране) и вертикали (чтобы весь код файла держался в памяти);

- ограничение размера функции или метода – в размер одного экрана.

## Рекомендации по стилю оформления кода в C++

### Пробелы и отступы

После зарезервированных ключевых слов языка C++ следует ставить пробел. Ставьте пробелы между операторами и операндами.

Отделяйте пробелами фигурные скобки, запятые и другие специальные символы.

Оставляйте пустые строки между функциями и между группами выражений.

### Именованье

Основные правила стиля кодирования приходятся на именованье. Вид имени сразу же (без поиска объявления) говорит нам что это: тип, переменная, функция, константа, макрос и т.д. Правила именования могут быть произвольными, однако важна их согласованность, и правилам нужно следовать.

#### Общие принципы именования

- используйте имена, который будут понятны даже людям из другой команды;
- имя должно говорить о цели или применении объекта;
- не экономьте на длине имени;
- не используйте аббревиатур; исключение: допускаются только известные аббревиатуры;
- не сокращайте слова.

Отметим, что типовые имена также допустимы: `i` для итератора или счётчика, `T` для параметра шаблона.

Примеры:

`firstName, homeworkScore`

Выбирайте подходящий тип данных для переменных. Если переменная содержит лишь целые числа, то определяйте её как `int`, а не `double`.

```
int count;           // Глобальная переменная
```

```
void func1() {        // Плохая практика
    count = 42;
}
```

```
int func1() {         // Хорошая практика!
    return 42;
}
int main() {
    int count = func1();}
```



### **Имена файлов**

Имена файлов должны быть записаны только строчными буквами, для разделения можно использовать подчёркивание ( `_` ) или дефис ( `-` ). Используйте тот разделитель, который используется в проекте.

### **Имена типов**

Имена пользовательских типов начинаются с прописной буквы, каждое новое слово также начинается с прописной буквы. Подчёркивания не используются:

```
MyExcitingEnum.
```

### **Имена переменных**

Имена переменных (включая параметры функций) пишутся строчными буквами, возможно с подчёркиванием между словами (в одном стиле):

```
line, lineAccount
```

! Следует инициализировать переменные в месте их объявления.

Префикс `n` следует использовать для представления количества объектов:

```
nPoints, nLines // Обозначение взято из математики
```

Переменным-итераторам следует давать имена

```
i, j, k // и т.д.
```

### **Имена констант**

Объекты объявляются как `const` или `constexpr`, чтобы значение не менялось в процессе выполнения. Имена констант константы пишутся в верхнем регистре. Подчёркивание может быть использовано в качестве разделителя.

```
OK, OUT_OF_MEMORY
```

Именованные константы (включая значения перечислений) должны быть записаны в верхнем регистре с нижним подчёркиванием в качестве разделителя.

### **Имена функций**

Названия методов и функций должны быть глаголами, быть записанными в смешанном регистре, начинаться с прописной буквы (в нижнем регистре) и каждое слово в имени пишется с прописной буквы:

```
getName();  
computeAverage();  
findNearestVertex();
```

## *Именованное пространство имён (namespace)*

Пространство имен называется строчными буквами.

Пространство имён верхнего уровня – это обычно название проекта.

## *Имена перечислений*

Перечисления должны именоваться либо как константы, либо как макросы:

```
enum TableErrors {  
    OK = 0,  
    OUT_OF_MEMORY = 1,  
    SURPRISE = 2  
};
```

## *Чрезмерность*

Если вы используете один и тот же код дважды или более раз, то найдите способ удалить излишний код, чтобы он не повторялся.

## *Комментарии*

**Сложный код, написанный с использованием хитрых ходов, следует не комментировать, а переписывать!**

Следует **делать** как можно меньше комментариев, делая код самодокументируемым путем выбора правильных имен и создания ясной логической структуры.

## *Эффективность*

Вызывая большую функцию и используя результат несколько раз, сохраните результат в переменной вместо того, чтобы постоянно вызывать данную функцию.

```
int index = lineAccount;    // Хорошая практика  
if (index >= 0) {  
    return index;  
}
```

```
if (lineAccount >= 0) {      // Плохая практика  
    return lineAccount;  
}
```

## Оформление

Основной отступ следует делать в два пробела

```
for (i = 0; i < nElements; i++)  
    a[i] = 0;  
....
```

```
for (initialization; condition; update) {  
    statements;  
}
```

```
while (!done) {  
    doSomething();  
    done = moreToDo();  
}
```

```
do {  
    statements;  
} while (condition);
```

## Проектирование функции

Хорошо спроектированная функция имеет следующие характеристики:

- полностью выполняет четко поставленную задачу;
- не берет на себя слишком много работы;
- не связана с другими функциями бесцельно;
- хранит данные максимально сжато;
- помогает распознать и разделить структуру программы;
- помогает избавиться от лишней работы, которая иначе присутствовала бы в программе.

## Рекомендации

При использовании операторов управления (if / else, for, while, других) всегда используйте {} и соответствующие отступы, даже если тело оператора состоит лишь из одной инструкции:

```
if (size == 0) {           // Хорошая практика  
    return;  
}
```

Возвращайте результаты проверки логического выражения (условия) напрямую:

```
return score1 == score2; // Хорошая практика
```

```
if (score1 == score2) { // Плохая практика
    return true;
} else {
    return false;
}
```

Проверка значения логического типа:

```
if (x) { // Хорошая практика
    ...
} else {
    ...
}
```

```
if (x == true) { // Плохая практика
    ...
} else if (x != true) {
    ...
}
```