Низкий уровень

Среди характеристик часто встречаются: ограничения на [абстракции данных](https://ru.wikipedia.org/wiki/Абстракция_данных), сильная статическая [типизация](https://ru.wikipedia.org/wiki/Система_типов), отсутствие промежуточной [среды выполнения](https://ru.wikipedia.org/wiki/Среда_выполнения), [прямой доступ к памяти](https://ru.wikipedia.org/wiki/Прямой_доступ_к_памяти).

Примеры: C, C++, Assembler.

**+** Полный контроль практически надо всем; вы используете только то, что вам нужно.

**–** Дополнительный контроль означает дополнительные сложности, которые могут сделать вроде бы простые задачи более трудными в реализации.

**+** Больший контроль над памятью; вы можете сделать то, что практически невозможно в других языках.

**–** Управлять памятью может быстро стать очень сложным.

**+** Позволит вам лучше понять, что происходит за кулисами в высокоуровневых языках и научит ценить абстракции.

**–** Легко закопаться в синтаксисе и мелких деталях вместо того, чтобы понимать концепцию и общую картину.

**+** Мотивирует думать об эффективности.

**–** Требуется предварительная оптимизация.

**+** Мотивирует думать об архитектуре наперёд.

**–** Изменения в плохой архитектуре могут быть болезненными. А хорошую архитектуру тяжело придумать.

**–** Относительно бедная [стандартная библиотека](https://ru.wikipedia.org/wiki/Стандартная_библиотека) означает, что вы должны часто полагаться на третьих лиц или изобретать колесо.

**–** Необходимо часто вставлять вспомогательные куски кода ([boilerplate](http://en.wikipedia.org/wiki/Boilerplate_code)), что увеличивает время на разработку.

Средний уровень

Среди характеристик часто встречаются: фокус на абстракциях, сильная статическая типизация, среда выполнения, ограничения на прямой доступ к памяти.

Примеры: Java, C#.

**+** Управлять памятью необязательно, но при желании вы можете это делать самостоятельно.

**–** До сих пор нужно понимать, как устроена память и как работает [сборка мусора](https://ru.wikipedia.org/wiki/Сборка_мусора), но язык этому обучению не способствует.

**+** Богатые стандартные библиотеки.

**–** Многие абстракции мешают новичку в освоении концепций, т.к. непонятно, почему они созданы именно таким образом.

**+** Компилируется в [байт-код](https://ru.wikipedia.org/wiki/Байт-код), упрощающий взаимодействие с другими языками.

**–** Байт-код требует установленной среды выполнения.

**–** До сих пор нужно часто вставлять стандартные куски кода (boilerplate), несмотря на наличие абстракций.

Высокий уровень

Среди характеристик часто встречаются: сильное абстрагирование, динамическая и/или слабая типизация, полностью независимое управление памятью и/или наличие среды выполнения.

Примеры: Python, Ruby, JavaScript, Common Lisp.

**+** Абстракции делают сложные задачи простыми.

**–** Надстройки для реализации абстракций снижают производительность.

**+** В целом всё просто и интуитивно, даже при внесении изменений.

**–** Архитектура может страдать, т.к. довольно просто вносить изменения почти в любом месте вместо того, чтобы вносить их там, где действительно нужно.

**+** Сравнительно большие стандартные библиотеки означают, что то, что вы хотите сделать, скорее всего уже реализовано и доступно.

**–** Из-за скрытых деталей сложно выяснять причины возникновения проблем, когда они появляются.

**+** Меньше вставок стандартного кода (boilerplate) – синтаксис значительно проще.

**–** Динамическая типизация усложняет поиск ошибок без запуска кода.

ЕЩЕ ВОТ ЭТО ПОЧИТАТЬ  
<http://bourabai.ru/alg/classification.htm>

# [**Структурное программирование**](https://ru.wikipedia.org/wiki/Структурное_программирование)

[Basic](https://ru.wikipedia.org/wiki/Basic)[Pascal](https://ru.wikipedia.org/wiki/Pascal)[QBASIC](https://ru.wikipedia.org/wiki/QBASIC)….

**Структу́рное программи́рование** — методология разработки [программного обеспечения](https://ru.wikipedia.org/wiki/Программное_обеспечение), в основе которой лежит представление программы в виде иерархической структуры [блоков](https://ru.wikipedia.org/wiki/Блок_(программирование)).

В соответствии с данной методологией любая программа строится без использования оператора goto из трёх базовых управляющих структур: последовательность, ветвление, цикл; кроме того, используются [подпрограммы](https://ru.wikipedia.org/wiki/Подпрограмма). При этом разработка программы ведётся пошагово, методом «сверху вниз».

# **[Процедурное программирование](https://ru.wikipedia.org/wiki/Процедурное_программирование)**

С, C++, Паскаль, Lua

**Процедурный язык программирования** предоставляет возможность программисту определять каждый шаг в процессе решения задачи. Особенность таких [языков программирования](http://dic.academic.ru/dic.nsf/ruwiki/1465) состоит в том, что задачи разбиваются на шаги и решаются шаг за шагом. Используя процедурный язык, программист определяет языковые конструкции для выполнения последовательности алгоритмических шагов.

Программа на процедурном языке программирования состоит из последовательности операторов (инструкций), задающих процедуру решения задачи.Выполнение программы сводится к последовательному выполнению операторов с целью преобразования исходного состояния памяти, то есть значений исходных данных, в заключительное, то есть в результаты.

# [**Объектно-ориентированное программирование**](https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование)

C++, Delphi, JS, Java, Objective-C, Perl, PHP, Python, Ruby

**Объе́ктно-ориенти́рованное программи́рование (ООП)** — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса[[1]](https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование#cite_note-1), а классы образуют иерархию наследования.

# [**Мультипарадигмальный язык программирования**](https://ru.wikipedia.org/wiki/Мультипарадигмальный_язык_программирования)

C++, C#

# **Мультипарадигмальное программирование** — это [программирование](http://dic.academic.ru/dic.nsf/ruwiki/977054) с одновременным использованием множества [парадигм программирования](http://dic.academic.ru/dic.nsf/ruwiki/8333).

**Паради́гма программи́рования** — это совокупность идей и понятий, определяющих стиль написания [компьютерных программ](https://ru.wikipedia.org/wiki/Компьютерная_программа) (подход к программированию). Это способ концептуализации, определяющий организацию вычислений и структурирование работы, выполняемой компьютером[[1]](https://ru.wikipedia.org/wiki/Парадигма_программирования#cite_note-.D0.A0.D0.BE.D0.B3.D0.B0.D0.BD.D0.BE.D0.B2.E2.80.942001.E2.80.94.D0.BF.D0.BE.D0.B4.D1.80.D0.B0.D0.B7.D0.B4.D0.B5.D0.BB_.C2.AB.D0.9F.D0.B0.D1.80.D0.B0.D0.B4.D0.B8.D0.B3.D0.BC.D1.8B_.D0.BF.D1.80.D0.BE.D0.B3.D1.80.D0.B0.D0.BC.D0.BC.D0.B8.D1.80.D0.BE.D0.B2.D0.B0.D0.BD.D0.B8.D1.8F.C2.BB.E2.80.94-1).

Важно отметить, что парадигма программирования не определяется однозначно языком программирования; практически все современные языки программирования в той или иной мере допускают использование различных парадигм ([мультипарадигмальное программирование](https://ru.wikipedia.org/wiki/Мультипарадигмальное_программирование)). Так, на [языке Си](https://ru.wikipedia.org/wiki/Си_(язык_программирования)), который не является объектно-ориентированным, можно работать в соответствии с принципами объектно-ориентированного программирования, хотя это и сопряжено с определёнными сложностями; функциональное программирование можно применять при работе на любом императивном языке, в котором имеются функции, и т. д.