УДК 004.43
ББК 32.973.2
С34

Книга «CPlusPlusProgramming» на английском языке доступна в pdf-формате по ссылке https://en.wikibooks.org/wiki/File:CPlusPlusProgramming.pdf.
Этот pdf-файл был создан программой, написанной Дирком Хюннигером, которая находится в свободном доступе под лицензией с открытым исходным кодом по адресу http://de.wikibooks.org/wiki/benutzer:dirk_huenniger/wb2pdf.
Авторами этой книги являются пользователи Panic (https://en.wikibooks.org/wiki/User%3APanic2k4) и Thenub314 (https://en.wikibooks.org/wiki/User%3AThenub314).
Авторы выражают благодарность за использование некоторых материалов из других источников, таких как Wikipedia, книги «Java Programming», «C Programming» и «C++ Reference» на Wikibooks, а также авторам Scott Wheeler, Stephen Ferg и Ivor Horton.

С++ с нуля до профессионального уровня. — Москва : Издательство АСТ, 2026. — 576 с. : ил. — (Быстрый старт в программирование).
ISBN 978-5-17-189077-3.

Перед вами подробное и тщательно структурированное руководство по языку программирования C++, охватывающее как фундаментальные основы разработки, так и современные возможности языка. Здесь C++ рассматривается не только как средство написания программ, но и как мощный инструмент для решения широкого круга задач с использованием различных подходов и парадигм программирования.

Материал построен по принципу последовательного обучения: от устройства программ и базовых конструкций C++ до объектно-ориентированного программирования, шаблонов, стандартной библиотеки STL, обработки исключений, управления памятью и продвинутых механизмов языка. Особое внимание уделяется пониманию принципов проектирования программного обеспечения и практическому применению изучаемых концепций.

В издании подробно рассматриваются классы, структуры, перегрузка операторов, шаблоны, умные указатели, семантика объектов, паттерны программирования, библиотека Boost, вопросы кроссплатформенной разработки, оптимизации и повышения качества программного кода. Теоретические выкладки сопровождаются многочисленными примерами, позволяющими лучше понять особенности языка и обрести практические навыки программирования.

Книга будет полезна как начинающим разработчикам, которые лишь делают первые шаги в изучении C++, так и опытным программистам, уже знакомым с другими языками и желающим глубже понять особенности современного C++. Благодаря сочетанию фундаментальных знаний и практических рекомендаций данное издание может использоваться и как учебное пособие для последовательного изучения языка, и в качестве справочного руководства при разработке собственных проектов.

УДК 004.43
ББК 32.973.2

ISBN 978-5-17-189077-3

Перевод на русский язык: ООО «Интеджер».
Издание на русском языке: ООО «Издательство АСТ».

Содержание

1. О книге

Эта книга посвящена языку программирования C++, его взаимодействию с проектированием программного обеспечения и практическому применению. Она представлена как курс от вводного до продвинутого уровня, но также может использоваться в качестве справочника.

Если вы уже знакомы с программированием на других языках, вы можете лишь бегло просмотреть главу **«Начало работы»**. Однако не следует пропускать раздел **«Парадигмы программирования»**, поскольку C++ имеет некоторые особенности, которые могут быть полезны даже в том случае, если вы уже знаете другой язык объектно-ориентированного программирования.

Раздел **«Сравнение языков»** содержит сравнения с некоторыми языками, которые вы, возможно, уже знаете, что может быть полезно для опытных программистов.

Если это ваше первое знакомство с программированием, читайте книгу с самого начала. Имейте в виду, что раздел **«Парадигмы программирования»** может быть сложным для понимания, если у вас пока мало опыта. Не отчаивайтесь — соответствующие идеи будут дополнительно объясняться по мере введения других концепций. Этот раздел предназначен для того, чтобы дать вам концептуальную основу — не только для понимания C++, но и для того, чтобы вы могли легче адаптироваться к другим языкам (и переходить с них), которые могут разделять схожие концепции.

2. C++ — мультипарадигменный язык

2.1. Введение в C++

C++ (произносится как «си плюс плюс») — это язык программирования *общего назначения со статической типизацией, свободным форматом записи и поддержкой нескольких парадигм программирования,* включая процедурное программирование, абстракцию данных и обобщенное программирование. В течение 1990-х годов **C++** стал одним из самых популярных языков программирования во всем мире.

2.1.1. История и стандартизация

Бьерн Страуструп, специалист в области компьютерных наук из Bell Labs, является разработчиком и первоначальным реализатором языка **C++** (изначально называвшегося «C с классами») в 1980-е годы как расширения языка программирования C. Развитие языка началось с добавления концепций объектно-ориентированного программирования, таких как *классы*, а затем, среди прочего, были добавлены *виртуальные функции, перегрузка операторов, множественное наследование, шаблоны* и *обработка исключений*. Эти и другие возможности подробно рассматриваются в данной книге.

Язык программирования C++ является стандартом, признанным ANSI (Американский национальный институт стандартов), BSI (Британский институт стандартов), DIN (немецкий институт стандартизации) и рядом других национальных организаций по стандартизации. В 1998 году он был утвержден ISO (Международная организация по стандартизации) как стандарт ISO/IEC 14882:1998, который состоит из двух частей: ядра языка и стандартной библиотеки. Последняя включает стандартную библиотеку шаблонов (Standard Template Library, STL) и стандартную библиотеку C (ANSI C 89).

К возможностям, появившимся в **C++**, относятся: объявления как выражения, приведения типов в стиле функций, операторы `new`/`delete`, тип `bool`, ссылочные типы, модификатор `const`, встроенные функции (inline functions), значения аргументов по умолчанию, перегрузка функций (overloading), пространства имен (namespace), классы (включая такие связанные возможности, как наследование, функции-члены, виртуальные функции, абстрактные классы и конструкторы), перегрузка операторов, шаблоны, оператор `::`, обработка исключений, идентификация типов во время выполнения и более строгая проверка типов в ряде случаев. Комментарии, начинающиеся с двух косых черт («`//`»), изначально были частью языка BCPL и были вновь введены в **C++**. Некоторые возможности **C++** впоследствии были заимствованы языком

C, включая `const`, `inline`, объявления в циклах `for` и комментарии в стиле **C++** (с использованием символа `//`).

Текущая версия стандарта — версия 2003 года, *ISO/IEC 14882:2003* — определяет язык как единое целое. STL, существовавшая еще до стандартизации **C++** и первоначально реализованная на языке Ada, теперь является неотъемлемой частью стандарта и обязательна для соответствующей реализации. Существует множество других библиотек **C++**, не входящих в стандарт, например BOOST. Также библиотеки, написанные на языке C, как правило, могут использоваться в программах на **C++**.

Начиная с 2004 года комитет по стандартизации (в который входит Бьерн Страуструп) активно работал над новой редакцией стандарта, временно обозначенной как C++0x. Некоторые реализации уже поддерживают часть предложенных изменений.

Пример исходного кода на C++

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

Традиционно первая программа, которую пишут при изучении нового языка, называется «Hello World», поскольку она лишь выводит на экран строку `Hello World` («Привет, Мир»). В разделе «Разбор программы `Hello World`» (https://en.wikibooks.org/wiki/C%2B%2B_Programming/Examples) приводится подробное объяснение этого кода; приведенный здесь исходный код предназначен для того, чтобы дать вам общее представление о простой программе на **C++**.

2.1.2. Обзор

Прежде чем приступить к изучению написания программ на **C++**, важно понять несколько ключевых концепций, с которыми вы можете столкнуться. Эти понятия не являются уникальными для **C++**, но помогают лучше понять программирование в целом. Читатели, уже имеющие опыт работы с другими языками программирования, могут лишь бегло ознакомиться с этим разделом.

Сегодня существует множество различных типов программ. От операционной системы, которая обеспечивает корректную работу компьютера, до видеоигр и музыкальных приложений, используемых для развлечения, — программы выполняют самые разные задачи. Общее для всех **программ** (также называемых **программным обеспечением** или **приложениями**) заключа-

ется в том, что они состоят из последовательности инструкций, записанных на некотором языке программирования. Эти инструкции указывают компьютеру, что нужно делать и, как правило, каким образом это делать. Программы могут содержать как инструкции для решения математических задач или отправки электронных писем, так и описание поведения персонажа видеоигры при получении урона. Компьютер выполняет инструкции программы последовательно — одну за другой, от начала и до конца.

2.1.3. Почему стоит изучать C++?

Почему бы и нет? Это самый простой и наглядный подход к принятию решения об изучении чего-либо. Хотя обучение само по себе всегда полезно, еще важнее — правильно выбрать, чему именно учиться, поскольку от этого зависит приоритизация ваших задач. С другой стороны, освоение нового навыка требует времени, и вам необходимо понять, какую пользу это вам принесет. Оцените свои цели, сравните схожие проекты или изучите потребности рынка разработки. В любом случае, чем больше языков программирования вы знаете, тем лучше.

Если вы подходите к обучению лишь с целью «пополнить коллекцию навыков», то есть готовы уделить этому ровно столько времени и усилий, чтобы понять основные особенности языка и ознакомиться с его менее очевидными сторонами, то лучше сначала изучить два других языка. Это поможет вам яснее понять, в чем заключается особенность подхода **C++** к решению задач программирования. Рекомендуется выбрать один императивный язык и один объектно-ориентированный. Для первого лучше всего подойдет C — он востребован на рынке и тесно связан с **C++**, хотя достойной альтернативой может быть и ASM. Для второго хорошим выбором будет Java — по тем же причинам.

Если же вы готовы уделить **C++** больше внимания, чем поверхностное изучение, вы можете выбрать его в качестве первого языка. В этом случае обязательно уделите время пониманию различных парадигм программирования и причин, по которым **C++** считается мультипарадигменным, или гибридным, языком.

Хотя знание C не является обязательным для понимания **C++**, вам необходимо владеть императивным языком программирования. **C++** не всегда облегчает понимание и различение более глубоких концепций, поскольку он предоставляет большую свободу в выборе способов реализации решений. Понимание того, какие подходы выбирать, станет основой для овладения языком.

Не стоит изучать **C++**, если ваша единственная цель — освоить объектно-ориентированное программирование, поскольку используемая терминология и некоторые подходы к решению задач могут усложнить понимание и освоение этих концепций. Если вас действительно интересует объектно-ориентированное программирование, лучше начать с Smalltalk.

Как и любой другой язык, **C++** имеет свою область применения, в которой он раскрывает свои сильные стороны. Он сложнее в освоении, чем C и Java, но при этом мощнее их обоих. **C++** позволяет абстрагироваться от низкоуровневых деталей, с которыми приходится работать в C и других языках низкого уровня, при этом предоставляя больше контроля и ответственности, чем Java. Поскольку он не предоставляет многие возможности «из коробки», доступные в языках более высокого уровня, вам придется самостоятельно искать и анализировать сторонние реализации этих возможностей и выбирать наиболее подходящие (или реализовывать собственные решения).

2.2. Что такое язык программирования?

В самом общем смысле язык программирования — это средство общения между человеком (программистом) и компьютером. Программист использует это средство для передачи компьютеру инструкций. Эти инструкции называются программами.

Подобно множеству языков, которые люди используют для общения друг с другом, существует множество языков, с помощью которых программист может взаимодействовать с компьютером. Каждый язык имеет свой набор слов и правил, называемых семантикой. При написании программы необходимо соблюдать семантику выбранного языка — в противном случае компьютер вас «не поймет».

Языки программирования в целом можно разделить на две категории: *низкоуровневые* и *высокоуровневые*. Далее мы рассмотрим эти понятия и их значение для C++.

2.2.1. Низкоуровневые языки

Языки более низкого уровня включают:

- **Машинный код** (также называемый двоичным кодом) — это самый низкий уровень среди языков низкого уровня. Он представляет собой последовательность нулей и единиц, которые формируют осмысленные инструкции, выполняемые компьютером. Достаточно взглянуть на страницу двоичного кода, чтобы понять, почему он практически не используется для написания программ — едва ли найдется человек, способный запомнить, что означают длинные последовательности из 0 и 1.
- **Язык ассемблера** (также называемый ASM) находится на уровень выше машинного кода. Это человекочитаемое представление инструкций машинного языка, выполняемых компьютером. Например, вместо использования двоичных кодов (0 и 1) программист применяет более удобные для запоминания мнемонические обозначения. Обычно это короткие последовательности букв, отражающие действие инструкции, например «ADD» для сложения или «MOV» для перемещения данных.

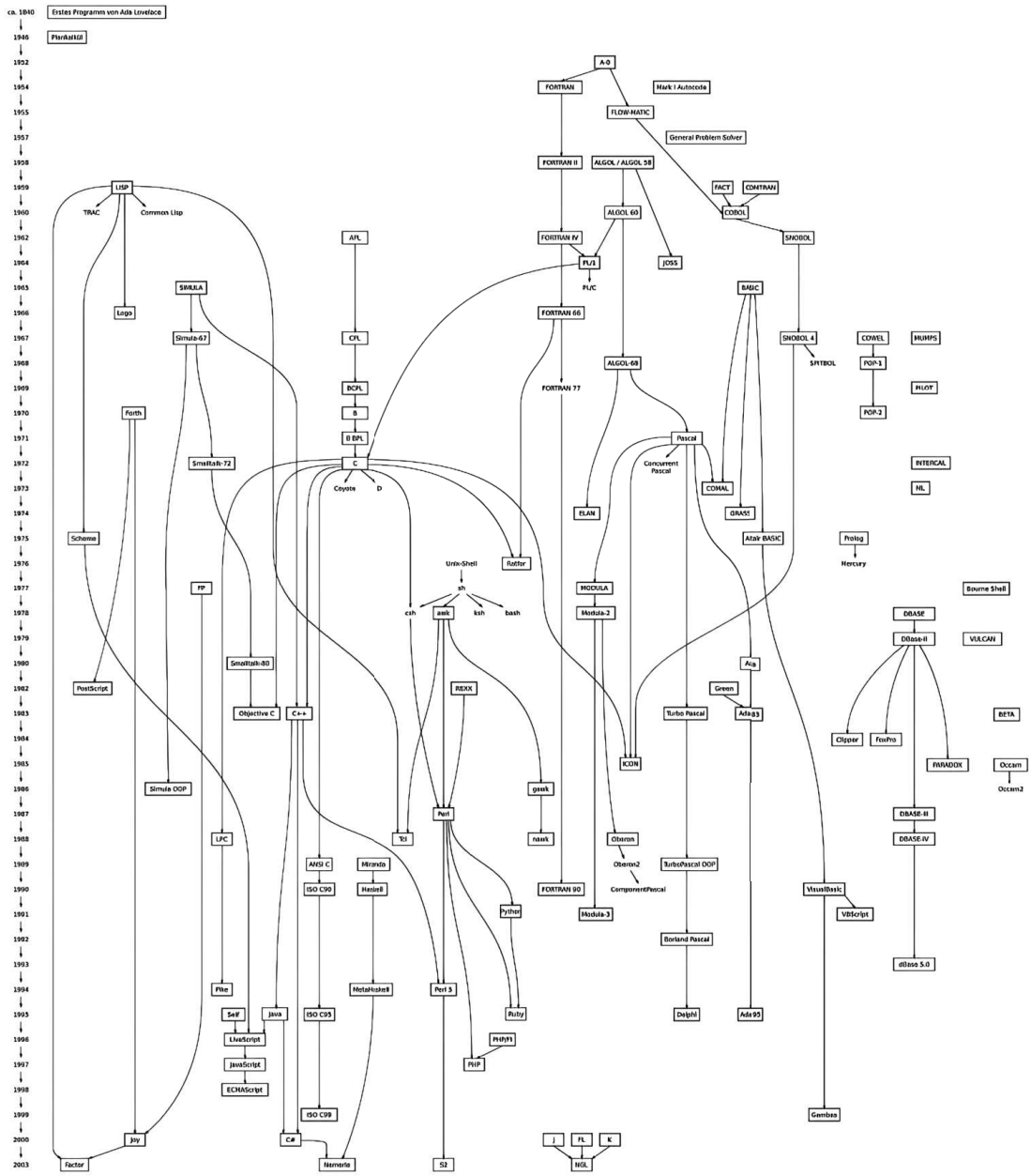

Рисунок 1. Изображение показывает большинство языков программирования и их взаимосвязи с середины XIX века до 2003 года.

> **Примечание**
>
> Язык ассемблера зависит от конкретного процессора. Это означает, что программа, написанная на ассемблере, не будет работать на компьютерах с другой архитектурой процессора.
>
> Использование ASM для оптимизации отдельных задач является распространенной практикой среди разработчиков на C++, однако требует особого подхода, поскольку ASM плохо переносим между платформами.

Для программирования на C++ не требуется знание ассемблера, однако общее понимание происходящего «за кулисами» может быть полезным. Изучение ассемблера дает больше контроля над программой, а также помогает при отладке и понимании кода.

Преимущества использования языков высокого уровня значительно перевешивают их недостатки, особенно с учетом размера и сложности современных программных задач. К таким преимуществам относятся:

- Развитая структура программ: циклы, функции и объекты имеют ограниченное применение в языках низкого уровня, поскольку считаются высокоуровневыми абстракциями и требуют дополнительного преобразования в низкоуровневые инструкции.
- Переносимость: программы, написанные на языках высокого уровня, могут работать на различных типах компьютеров с минимальными изменениями или вовсе без них. Низкоуровневые программы часто используют специфические возможности конкретных процессоров и требуют переписывания для запуска на другой системе.
- Простота использования: многие задачи, требующие большого количества строк кода на ассемблере, в языках высокого уровня могут быть реализованы с помощью нескольких вызовов функций из библиотек. Например, Java позволяет создать рабочее окно примерно в пяти строках кода, тогда как аналогичная программа на ассемблере потребует как минимум в четыре раза больше кода.

2.2.2. Высокоуровневые языки

Высокоуровневые языки позволяют делать больше с меньшим количеством кода, хотя иногда это сопровождается снижением производительности и уменьшением степени контроля со стороны программиста. Они также стремятся использовать слова, близкие к английскому языку, в форме, которую может прочитать и в целом понять человек с небольшим опытом. Программа, написанная на таком языке, иногда называется человекочитаемым кодом. В целом, чем выше уровень абстракции, тем проще язык для изучения.

Тем не менее ни один язык программирования не является тем, что можно было бы назвать обычным английским языком (хотя BASIC близок к этому). По этой причине текст программы часто называют кодом, или, более точно, исходным кодом. Это более подробно рассматривается в разделе «Код» данной книги.

Высокоуровневые языки частично решают проблему абстрагирования от аппаратного обеспечения (CPU, сопроцессоры, количество регистров и т. д.), обеспечивая переносимость кода.

При этом стоит помнить, что такая классификация со временем совершенствуется. C++ по-прежнему считается языком высокого уровня, однако с появлением более новых языков (Java, C#, Ruby и т. д.) его все чаще относят к более низкому уровню, наряду с языком C.

2.2.3. Трансляция языков программирования

Поскольку компьютер способен понимать только машинный код, человекочитаемый код должен быть либо интерпретирован, либо переведен в машинный код.

Интерпретатор — это программа (часто написанная на языке более низкого уровня), которая последовательно обрабатывает инструкции программы, преобразуя их в команды, выполняемые по мере их поступления. Обычно каждая инструкция представляет собой одну строку текста или имеет другой четкий способ отделения от остальных, и программа должна интерпретироваться заново при каждом запуске.

Компилятор — это программа, предназначенная для перевода исходного кода, инструкция за инструкцией, в машинный код. При этом одна инструкция, понятная компилятору, может преобразовываться в несколько машинных инструкций. Перевод выполняется только один раз, после чего компьютер может напрямую выполнять полученные инструкции при каждом запуске программы. Подробное рассмотрение компилятора C++ приводится в разделе «Компилятор» данной книги.

Слова и конструкции, используемые для указания компьютеру, что делать, могут различаться, однако практически любой язык программирования включает средства для выполнения следующих задач:

- *Ввод*
  Ввод — это получение информации от устройства, такого как клавиатура или мышь, либо от другой программы.
- *Вывод*
  Вывод — это противоположность ввода; он заключается в передаче информации на экран, другое устройство или в другую программу.
- *Математика / алгоритмы*
  Все процессоры («мозг» компьютера) способны выполнять базовые математические операции, и каждый язык программирования предоставляет средства для их выполнения.
- *Проверка условий*
  Проверка условий заключается в том, чтобы указать компьютеру проверить некоторое условие и выполнить определенные действия в зависимости от того, истинно оно или ложно. Условные конструкции являются одной из важнейших концепций программирования, и во всех языках существуют средства для работы с условиями.
- *Повторение*
  Выполнение некоторого действия многократно, как правило, с определенными изменениями.

Более подробное рассмотрение приведено в разделе «Инструкции» данной книги.

C++ в основном *компилируется*, а не *интерпретируется* (хотя существуют и интерпретаторы C++), и затем выполняется. Несмотря на кажущуюся сложность, далее вы увидите, что это довольно просто.

Как мы уже отмечали в разделе «Введение в C++», язык C++ развился из языка C за счет добавления дополнительных уровней абстракции (поэтому можно сказать, что C++ является языком более высокого уровня по сравнению с C). Особенности этих различий будут рассмотрены в разделе «Парадигмы программирования» данной книги, а тем, кто уже знаком с другими языками, стоит также обратить внимание на раздел «Сравнение языков программирования».

2.3. Парадигмы программирования

Парадигма программирования — это модель программирования, основанная на определенных концепциях, которая определяет, как программисты проектируют, организуют и пишут программы. Мультипарадигменный язык программирования позволяет программистам выбирать один конкретный подход или комбинировать элементы различных парадигм. C++ как мультипарадигменный язык поддерживает как отдельное использование, так и смешение процедурного и объектно-ориентированного программирования, а также включает элементы обобщенного и даже функционального программирования.

2.3.1. Процедурное программирование

Процедурное программирование можно определить как подвид императивного программирования — парадигмы, основанной на концепции вызова процедур, при которой инструкции организуются в процедуры (также называемые подпрограммами или функциями). Вызовы процедур обладают модульностью и ограничены областью видимости. Процедурная программа состоит из одного или нескольких модулей. Каждый модуль, в свою очередь, включает одну или несколько подпрограмм. В зависимости от языка программирования модули могут содержать процедуры, функции, подпрограммы или методы. Процедурные программы могут иметь несколько уровней или областей видимости, при этом подпрограммы могут быть определены внутри других подпрограмм. Каждая область видимости может содержать имена, недоступные за ее пределами.

Процедурное программирование обладает рядом преимуществ по сравнению с простым последовательным программированием, поскольку процедурный код:

- легче читается и проще в сопровождении;
- более гибкий;

- облегчает применение принципов правильного проектирования программ;
- позволяет повторно использовать модули в виде библиотек кода.

> **Примечание**
>
> В настоящее время крайне редко можно встретить использование C++ строго в рамках процедурной парадигмы — обычно она применяется лишь в небольших демонстрационных или тестовых программах.

2.3.2. Статическая типизация

Типизация определяет, каким образом язык программирования работает с переменными и как они различаются по типу. Переменные — это значения, используемые программой во время выполнения. Эти значения могут изменяться — отсюда и их название. *Статическая типизация*, как правило, приводит к созданию скомпилированного кода, который выполняется быстрее. Когда компилятор точно знает используемые типы, он может проще сгенерировать корректный машинный код. В C++ переменные должны быть объявлены до их использования, чтобы компилятор знал их тип, поэтому язык относится к статически типизированным. Языки, не обладающие этим свойством, называются *динамически типизированными*.

Статическая типизация позволяет более надежно обнаруживать ошибки типов на этапе компиляции, повышая надежность программ. Проще говоря, это означает, что «круглый колышек не войдет в квадратное отверстие» — компилятор сообщит об ошибке, если возникает неоднозначность или несовместимость типов. Тем не менее среди программистов существуют разногласия относительно того, насколько часто встречаются ошибки типов и какую долю всех ошибок они составляют. Сторонники статической типизации считают, что программы становятся надежнее благодаря проверке типов, тогда как сторонники динамической типизации указывают на надежность динамических языков за счет небольшого количества ошибок в реальных проектах. Таким образом, предполагается, что ценность статической типизации возрастает по мере усиления системы типов.

Статическая типизация сильнее ограничивает использование мощных языковых конструкций по сравнению с более простыми. Это делает такие конструкции сложнее в применении и возлагает на программиста ответственность за выбор «правильного инструмента для задачи», тогда как он мог бы предпочесть самый мощный доступный инструмент. Использование чрезмерно мощных средств может привести к проблемам с производительностью, надежностью или корректностью, поскольку существуют теоретические ограничения на свойства таких конструкций.