ВикипедиЯ

Сравнение языков программирования

Материал из Википедии — свободной энциклопедии

В приведённой ниже таблице отмечено наличие или отсутствие тех или иных возможностей в некоторых популярных сегодня языках программирования. Столбцы упорядочены по алфавиту. Если возможность в языке недоступна напрямую, но может быть эмулирована с помощью других средств, то в таблице отмечено, что её нет.

При заполнении таблицы учтены только фактические данные, при том, что наличие возможности не обязательно является преимуществом языка, а отсутствие — недостатком.

Условные обозначения	
+	Указанная возможность присутствует
−	Указанная возможность отсутствует
+/−	Возможность поддерживается не полностью
−/+	Возможность поддерживается очень ограниченно
?	Нет данных
N/A	Постановка вопроса не применима к языку

Сравнения языков программирования

Общее сравнение

Основной синтаксис

Основные инструкции

Массивы

Ассоциативные массивы

Операции со строками

Строковые функции

List comprehension

Объектно-ориентированное программирование

Объектно-ориентированные конструкторы

Доступ к базам данных

СУБД баз данных

Оценка стратегии

Список программ «hello world»

Влияние ALGOL 58 на ALGOL 60

ALGOL 60: Сравнения с другими языками

Сравнение ALGOL 68 и C++

ALGOL 68: Сравнения с другими языками

Совместимость C и C++

Сравнение Pascal и Borland Delphi

Сравнение Object Pascal и C

Сравнение Pascal и C

Сравнение Java и C++

Сравнение C# и Java

Содержание

Сравнение C# и Visual Basic .NET

- Вывод типов переменных из использования
- Вывод типов-аргументов при вызове метода
- Вывод сигнатуры для локальных функций
- Параметрический полиморфизм
- Параметрический полиморфизм с ковариантностью
- Параметрический полиморфизм высших порядков
- Информация о типах в runtime
- Информация о типах-параметрах в runtime

Компилятор/интерпретатор

- Open-source компилятор (интерпретатор)
- Возможность компиляции
- Bootstrapping
- Многопоточная компиляция
- Интерпретатор командной строки
- Условная компиляция

Управление памятью

- Объекты на стеке
- Неуправляемые указатели
- Ручное управление памятью
- Сборка мусора

Управление потоком вычислений

- Инструкция goto
- Инструкция break без метки
- Инструкция break с меткой
- Поддержка try/catch
- Блок finally
- Блок else (исключения)
- Перезапуски
- Легковесные процессы

Типы и структуры данных

- Многомерные массивы
- Динамические массивы
- Ассоциативные массивы
- Цикл foreach
- Списковые включения
- Кортежи
- Целые числа произвольной длины
- Целые числа с контролем границ

Объектно-ориентированные возможности

- Интерфейсы
- Множественное наследование
- Мультиметоды
- Переименование членов при наследовании
- Решение конфликта имен при множественном наследовании

Функциональные возможности

- First class functions
- Лексические замыкания
- Частичное применение

Разное

- Макросы
- Шаблоны/Generics
- Поддержка Unicode в идентификаторах
- Перегрузка функций
- Динамические переменные
- Именованные параметры
- Значения параметров по умолчанию
- Локальные функции
- Сопоставление с образцом
- Контрактное программирование

Ссылки

Парадигмы

Возможность	Языки											
	Active Oberon	Ada	C	C++	C#	Go	Java	JavaScript	Common Lisp	Modula-3	Python	Rust
Императивная	+	+	+	+	+	+	+	+	+	+	+	+
Объектно-ориентированная	+	+	−/+[2]	+	+	+/−	+	+ [3]	+	+	+	−/+ [5]
Функциональная	−	−	+/− [6]	−/+ [7]	+/−	+/−	−/+	+/−	+/−[8]	−	+	+
Рефлексивная	−	−	−	−/+ [9]	−/+	−	−/+	+	+	−	+	−/+ [10]
Обобщенное программирование	−	+	+ [11]	−/+ [12]	+	−	+	+	+	+	+	+
Логическая	−	−	−	−	−	−	−	−	+/− [14]	+	−	−
Декларативная	−	−	−	−	−/+[15]	−	−	+/−	+[16]	−	+	−
Распределенная	−	+ [18]	+/− [19]	+/− [19]	−/+ [20]	+	+	−	+/−	−/+	−/+	+

Типизация

Возможность	Язык											
	Active Oberon	Ada	C	C++	C#	Go	Java	JavaScript	Common Lisp	Modula-3	Python	Rust
Статическая типизация	+	+	+	+	+	+	+	–	+/–[23]	+	–	+
Динамическая типизация	–	–	–	–	+ [26]	–	–	+	+	+	+	–/+ [29]
Явная типизация	+	+	+	+	+	+	+	–	+/–[23]	+	+/– [34]	+
Неявная типизация	–	–	–	–/+	–/+[36]	–	–	+	+	+	+	+
Неявное приведение типов без потери данных	–/+	–/+ [37]	+	+	+	–	+ [38]	+	+	–/+	+	–/+ [39]
Неявное приведение типов с потерей данных	–	–	+	+	–	–	–	?	–	–	–	–
Неявное приведение типов в неоднозначных ситуациях	–	–	+	+	+	–	–	+	–	–	–	–
Алиасы типов	+	+	+	+	+	?	–	N/A	+[42]	+	N/A	+
Вывод типов переменных из инициализатора	–	–	–	+/– [44]	+	?	–	N/A	+/– [45]	+	N/A	+
Вывод типов переменных из использования	–	–	–	+/– [44]	–	?	–	N/A	+/– [45]	+	N/A	+
Вывод типов-аргументов при вызове метода	–	–	–	+	+	?	+	N/A	+/– [45]	–	N/A	+
Вывод сигнатуры для локальных функций	–	–	–	+/– [46]	–	?	–	N/A	+/–[45]	–	N/A	+
Параметрический полиморфизм	–	–	N/A	–	+	–	+	–	+	–	N/A	+
Параметрический полиморфизм с ковариантностью	–	–	N/A	–	+/– [47]	–	+	–	+	–	N/A	+
Параметрический полиморфизм высших порядков	–	–	N/A	–	–	–	–	–	+	–	N/A	– [48]
Информация о типах в runtime	+	–/+ [49]	–	–/+ [50]	+	?	+	–/+	+	+	+	–
Информация о типах-параметрах в runtime	–	–	–	–/+	+	?	–	–/+	+	+	+	–

Компилятор/интерпретатор

Возможность	Язык											
	Active Oberon	Ada	C	C++	C#	Go	Java	JavaScript	Common Lisp	Modula-3	Python	Rust
Open-source компилятор (интерпретатор)	+	+	+	+	+	?	+	+	+	+	+	+
Возможность компиляции	+	+	+	+	+	?	+	+	+	+	+	+
Bootstrapping	+	+	+	+	+	?	+ [58]	+ [59]	+[60]	–	+ [61]	+
Многопоточная компиляция	+	+	+	+	–	?	+	?	+	–	N/A	+
Интерпретатор командной строки	+	+/– [63]	–/+ [64]	+/– [64]	+	?	–	+ [66]	+	–	+	–
Условная компиляция	–	+/– [69]	+	+	+	?	–/+ [70]	–/+ [71]	+[72]	+	N/A	+

Управление памятью

Возможность	Язык											
	Active Oberon	Ada	C	C++	C#	Go	Java	JavaScript	Common Lisp	Modula-3	Python	Rust
Создание объектов на стеке	+	+	+	+	+	?	–	–	+/– [75]	–	–	+
Неуправляемые указатели	+	+	+	+	+	?	–[77]	–	–[77]	+	– [78]	+ [79]
Ручное управление памятью	+/–	+	+	+	+[81]	?	–[77]	–	–[77]	+	–[77]	+ [79]
Сборка мусора	+	–/+ [83]	– [84]	–/+ [85]	+	+	+	+	+	+	+	+/– [86]

Управление потоком вычислений

Возможность	Язык											
	Active Oberon	Ada	C	C++	C#	Go	Java	JavaScript	Common Lisp	Modula-3	Python	Rust
Инструкция goto	–	+	+	+	+	?	– [88]	–	+[89]	–	–	–
Инструкции break без метки	+	+	+	+	+	?	+	+	+[93]	–	+	+
Инструкция break с меткой	–	+	–	–	–	?	+	+	+[95]	–	–	+
Поддержка try/catch	–	+	–	+	+	–	+	+	+ [99]	+	+	– [101]
Блок finally	+	–/+ [103]	–	–	+	?	+	+ [104]	+ [105]	+	+	– [101]
Блок else (исключения)	–	–	–	–	+	?	+ [108]	?	+ [109]	+	+	– [101]
Перезапуски	+[111]	?	–	?	–	?	?	?	+	–	?	– [101]
Ленивые вычисления	–	?	–	–/+	–/+ [114]	–	–	–	– [116]	–	+	+
Continuations	–	?	–/+[122]	?	+ [123]	?	?	?	–[124]	?	–	–
Легковесные процессы (Coroutines)	–	–	–	–	–	+	+/– [127]	–	+/–[128]	+	+/– [130]	–/+ [132]

Типы и структуры данных

Возможность	Язык											
	Active Oberon	Ada	C	C++	C#	Go	Java	JavaScript	Common Lisp	Modula-3	Python	Rust
Кортежи	–	–	–	+/–	+	+	–	–	+	–	+	+
Алгебраические типы данных	–	–/+ [135]	–	–	–	?	–	N/A [136]	N/A [136]	–	N/A [136]	+
Многомерные массивы	+	?	+	+	+	+	+/–	+/–	+	+	+/–	+
Динамические массивы	+	?	–[139]	+	+/–	+	+/ –[140]	+/–	+	+	+/–	+/– [141]
Ассоциативные массивы	–	?	–	+ [142]	+	?	+/ –[143]	+	+	–	+	+ [145]
Контроль границ массивов	+	?	–	+/– [146]	+	+	+	N/A [147]	+	+	+	+
Цикл foreach	–	+/– [150]	–	+ [151]	+	+	+	+ [153]	+ [154]	–	+	+
Списковые включения	–	–	–	–	–/+ [155]	?	–	–	+[156]	–	+	–
Целые числа произвольной длины	–	–	–	–	+ [157]	?	+ [158]	–	+	–	+	+/– [160]
Целые числа с контролем границ	–/+	+	–	–	–	?	–	–	+[163]	+	–	–

Объектно-ориентированные возможности

Возможность	Язык											
	Active Oberon	Ada	C	C++	C#	Go	Java	JavaScript	Common Lisp	Modula-3	Python	Rust
Интерфейсы	+	?	+/− [165]	+ [166]	+	+	+	?	N/A[167]	−	+	+ [169]
Мультиметоды	−	−	−	−/+[170]	−/+[171]	−	− [172]	−	+	−	− [172]	−/+ [174]
Mixins	−	?	−	−/+[175]	−	?	+	?	+	−	+ [178]	?
Переименование членов при наследовании	−	?	N/A	−/+[180]	−	?	−	?	−	−	−	N/A [181]
Множественное наследование	−	?	N/A	+	−	?	−	?	+	−	+	N/A [181]
Решение конфликта имен при множественном наследовании	N/A	?	N/A	−/+ [182]	N/A	?	N/A	?	+[184]	N/A	+	N/A [181]

Функциональные возможности

Возможность	Язык											
	Active Oberon	Ada	C	C++	C#	Go	Java	JavaScript	Common Lisp	Modula-3	Python	Rust
Декларации чистоты функций	−	−	−	−	−	?	−	−	−	−	−	−
First class functions	+	?	−/+[186]	+[187]	+	?	−	+	+	+	+	+
Анонимные функции	−	?	−	+[189]	+[190]	?	+	+	+[191]	−	+/−[192]	+
Лексические замыкания	−	−	−	+ [193]	+	?	+[194]	+	+	−	+	+
Частичное применение	−	?	−	+/− [197]	?	?	−	+[199]	−	−	+ [201]	−/+ [202]
Каррирование	−	−	−	+/− [203]	+	?	−	+	−	−	+	−/+ [202]

Разное

Возможность	Язык											
	Active Oberon	Ada	C	C++	C#	Go	Java	JavaScript	Common Lisp	Modula-3	Python	Rust
Макросы	−	−/+	+ [206]	+ [206]	+/−	?	−	−	+	−	−	+
Шаблоны/Generics	−	+	−	+	+	−	+	N/A [211]	N/A [211]	+	N/A [211]	+
Поддержка Unicode в идентификаторах	−	+	+ [214]	+ [215]	+	?	+	+	+[216]	−	+ [217]	−/+ [219]
Перегрузка функций	−	+	−	+	+	−	+	−/+ [220]	+[221]	−	− [223]	−
Динамические переменные	−	?	−	−	+	−	?	?	+[225]	−	−	−
Именованные параметры	−	+	−	−	+ [226]	−	−	−/+ [227]	+[228]	+	+	−
Значения параметров по умолчанию	+	+	−	+	+ [226]	+	−	+	+[232]	+	+	−
Локальные функции	+	+	−/+ [234]	+ [235]	+	+	+/− [236]	+	+[237]	+	+	+
Сопоставление с образцом	−	−	−	−	+	−	−	−	+/−[238]	−	− [223]	+
Контрактное программирование	−	−	−	−	+ [239]	?	+/− [240]	?	+	−	+/−	−
Наличие библиотек для работы с графикой и мультимедиа (OpenGL/WebGL/OpenML, OpenAL, DirectX)	+	?	+	+	+[242]	?	+	+	+[244]	+	+	+

Примечания

1. ↑ Императивный/Haskell. Монады позволяют выполнять императивные действия.
2. ↑ Несмотря на отсутствие встроенных средств поддержки ООП, реализация объектно-ориентированного подхода возможна. В качестве наиболее ярких примеров можно привести библиотеки OpenGL, OpenCL, OpenMAX AL и т. п., которые реализуют именно ООП средствами языка C.
3. ↑ ООП/Javascript. Прототипная модель ООП.
4. ↑ ООП/Haskell. Классы типов и семейства типов перекрывают возможности ООП.
5. ↑ ООП/Rust. Объектно-ориентированное программирование как таковое на уровне языка не поддерживается, но язык позволяет реализовать большинство понятий ООП при помощи других абстракций; см. Rust Frequently Asked Questions // Design Patterns (https://www.rust-lang.org/en-US/faq.html#design-patterns) (англ.). — Официальный FAQ о языке Rust. — «Many things you can do in OO languages you can do in Rust, but not everything, and not always using the same abstraction you're accustomed to. […] There *are* ways of translating object-oriented concepts like multiple inheritance to Rust, but as Rust is not object-oriented the result of the translation may look substantially different from its appearance in an OO language.». Проверено 24 июля 2017.
6. ↑ присваивание как типизируемое выражение (не оператор), тернарная операция, функции с неограниченным числом параметров, void* как небезопасная форма параметрического полиморфизма
7. ↑ Робинсон определил шаблоны C++ как полный по Тьюрингу функциональный язык программирования. Саймон Пейтон Джонс и Тим Шерд назвали работу Робинсона провокационной, природу шаблонного метапрограммирования — нелепой и причудливой, и отметили, что функциональные программисты не спешат использовать C++: *Sheard T., Jones S.P.* Template Metaprogramming in Haskell // Haskell Workshop. — Pittsburgh: ACM 1-58113-415-0/01/0009, 2002.
8. ↑ Пространства имен функций и данных разделены, для работы с функциями высших порядков используется специальный синтаксис
9. ↑ Гомоиконность отсутствует, но RTTI есть
10. ↑ Рефлексивный/Rust. В процедурных макроопределениях (*procedural macros*), см. Procedural Macros (and custom Derive) (https://doc.rust-lang.org/beta/book/first-edition/procedural-macros.html) (англ.). — The Rust Programming Language, 1st Edition. Проверено 24 июля 2017.
11. ↑ void*
12. ↑ шаблоны являются отдельным мини-языком
13. ↑ Логический/Haskell. Изначально инструментов для логического программирования не встроено, но есть сторонние библиотеки. Существует академический функционально-логический язык Curry, берущий Haskell за основу.
14. ↑ Логический/Common Lisp. Логическая парадигма изначально в язык не встроена, но реализуется средствами языка.
15. ↑ LINQ

16. ↑ В языке существует множество декларативных конструкций, и, более того, возможность создавать свои, с помощью макросов.
17. ↑ Декларативный/Perl. Только регулярные выражения.
18. ↑ Распределённый/Ada. См. Annex E. Distributed Systems (http://www.adaic.com/standards/05rm/html/RM-E.html).
19. Распределённый/C и C++. Многие распространённые компиляторы поддерживают директивы для распараллеливания в рамках технологий MPI и OpenMP.
20. ↑ Распределённый/C#. Существуют проекты распределённых модификаций языка, например Parallel C# (http://www.parallelcsharp.com/).
21. ↑ std.parallelism
22. ↑ Распределённый/Haskell. Модель языка подразумевает распределённое использование, при этом не требуя от программиста усилий на реализацию распределённости. Один из поддерживающих эту возможность компиляторов — Glasgow Distributed Haskell.
23. ANSI стандарт языка предусматривает опциональные декларации типов, которые какие-либо конкретные реализации могут использовать по своему усмотрению. Большинство современных реализаций CL принимают декларации типов в расчет, и используют для статической проверки типов и в целях оптимизации.
24. ↑ Статическая типизация/Perl. С версии 5.6. Только для не встроенных типов.
25. ↑ Статическая типизация/Smalltalk. Возможность статической типизации есть в диалекте Smalltalk — Strongtalk'е.
26. ↑ Динамическая типизация/C#. Посредством специального псевдо-типа `dynamic` с версии 4.0.
27. ↑ Динамическая типизация/F#. Компилятор поддерживает синтаксический сахар в виде преобразования использования оператора (?) `xml?name` в вызов `xml.op_Dynamic("name")`, на базе чего может быть реализована имитация динамической типизации.
28. ↑ Динамическая типизация/Haskell. Обеспечивается модулем `Data.Dynamic`.
29. ↑ Динамическая типизация/Rust. С помощью типажа `Any`, см. The Rust Standard Library // std::any (https://doc.rust-lang.org/std/any/) (англ.). Проверено 25 июля 2017.
30. ↑ Динамическая типизация/VB.NET. Контролируемо с помощью Option Strict.
31. ↑ Динамическая типизация/Delphi. Посредством специального типа Variant.
32. ↑ Явная типизация/Erlang. Можно использовать т. н. type test BIFs. См (http://erlang.org/doc/reference_manual/expressions.html#6.24)
33. ↑ Явная типизация/Perl. См. Prototypes в *man perlsub*.
34. ↑ Явная типизация/Python. Частично в Python 3.0.
35. ↑ Явная типизация/Smalltalk. Есть в Strongtalk.
36. ↑ var, dynamic etc.
37. ↑ Неявное приведение типов/Ada. См. 4.6 Type Conversions (http://www.adaic.com/standards/05rm/html/RM-4-6.html#I2822).
38. ↑ Расширение для примитивных типов, приведение к супертипу для классов.
39. ↑ Неявное приведение типов без потери данных/Rust. В *очень* небольшом наборе ситуаций, в частности: приведение ссылки к указателю; изменяемой ссылки (указателя) к неизменяемой ссылке (указателю); объекта определённого типа к объекту с типажом, реализованным этим типом. См. Coercions (https://doc.rust-lang.org/nomicon/coercions.html) (англ.). — The Rustonomicon. Проверено 24 июля 2017.
40. ↑ Неявное приведение с потерей данных/Perl. При сложении строки с числом: `$a = '5aa'; print $a + 0;` Напечатает: 5
41. ↑ Неявное приведение в неоднозначных ситуациях/Perl. Не совсем корректно, так как в Perl эти ситуации однозначны: `1 + "2" # 3` и `1 . "2" # "12"`
42. ↑ Макрос DEFTYPE (http://www.lispworks.com/documentation/HyperSpec/Body/m_deftp.htm)
43. ↑ Abstract types (http://docs.scala-lang.org/tutorials/tour/abstract-types.html)
44. Вывод типов/C++. Поддержка вывода типов имплементируется в C++11 с использованием ключевых слов `auto` и `decltype`.
45. Вывод типов/Common Lisp. Некоторые компиляторы Common Lisp, такие как SBCL, поддерживают частичный вывод типов.
46. ↑ auto function = [&](int a){} в c++11
47. ↑ Параметрический полиморфизм с ковариантностью/C#. Доступно начиная с C# 4.0 для типов интерфейсов и делегатов.
48. ↑ Параметрический полиморфизм высших порядков/Rust. См. RFC 324 (https://github.com/rust-lang/rfcs/issues/324).
49. ↑ Информация о типах в runtime/Ada. Точный тип узнать можно (Ada.Tags (http://www.adaic.com/standards/05rm/html/RM-3-9.html#I2032)), но полной поддержки отражения в языке нет. Можно узнать имя, предков, интерфейсы, сериализовать объект, но нельзя запросить список методов.
50. ↑ Информация о типах в runtime/C++. Можно сравнить типы на точное совпадение, узнать имя типа (typeid (http://www.hep.wisc.edu/~pinghc/isocppstd/expr.html#expr.typeid)), приводить типы вниз по иерархии наследования.
51. ↑ См. встроенную функцию ref и метод isa
52. ↑ см. TypeTags
53. ↑ Open-source компилятор (интерпретатор)/Smalltalk. В любом диалекте Smalltalk исходники всего, кроме виртуальной машины, (то есть библиотека классов, компилятор в байткод, среда разработки, сторонние библиотеки и пр.) принципиально открыты — это свойство языка. Из основных диалектов исходники виртуальной машины открыты у GNU Smalltalk, Squeak и Strongtalk.
54. ↑ Open-source компилятор (интерпретатор)/Delphi. FreePascal и Lazarus.
55. ↑ Возможность компиляции/Erlang. HiPE — High Performance Erlang (http://www.it.uu.se/research/group/hipe/). Доступен только для *nix-систем.
56. ↑ Существуют PHP-компиляторы, вполне корректно комилирующие любые PHP-скрипты. Например, Roadsend PHP Compiler (http://www.roadsend.com/home/index.php).
57. ↑ Возможность компиляции/Smalltalk. Стандартная реализация в Smalltalk — это прозрачная компиляция в байт-код (в момент сохранения изменённого исходного кода) с последующим исполнением на виртуальной машине, часто с использованием JIT-компилятора. Однако некоторые диалекты поддерживают прямую компиляцию в машинные коды. В частности, к таким диалектам относятся Smalltalk MT и Smalltalk/X.
58. ↑ Bootstrapping-компилятор/Java. Java Compiler API появилось в версии 6.0.
59. ↑ Narcissus.
60. ↑ Например, SBCL
61. ↑ Bootstrapping-компилятор/Python. Проект PyPy (http://codespeak.net/pypy/dist/pypy/doc/news.html).
62. ↑ Bootstrapping-компилятор/Smalltalk. Компилятор в байт-коды изначально написан на самом Smalltalk и исполняется внутри виртуальной машины. Кроме этого также есть примеры *виртуальных машин* Smalltalk, написанных на самом Smalltalk — к ним, в частности, относится виртуальная машина Squeak, написанная на подмножестве Smalltalk, которое потом транслируется в C и компилируется в машинные коды. При этом собственно разработка и отладка виртуальной машины Squeak осуществляется внутри работающей системы Squeak.
63. ↑ Интерпретатор командной строки/Ada. Business Shell (BUSH) (http://www.pegasoft.ca/bush.html).
64. Интерпретатор командной строки/C++. C++ интерпретатор CINT (https://root.cern.ch/cint).

65. ↑ компиляция на лету с помощью rdmd
66. ↑ Rhino Shell (http://developer.mozilla.org/en/docs/Rhino_Shell).
67. ↑ В диалекте GNU Smalltalk реализована поддержка командной строки.
68. ↑ Интерпретатор командной строки/Delphi. PascalScript. (http://wiki.freepascal.org/Pascal_Script)
69. ↑ Условная компиляция/Ada. Поскольку использование препроцессора существенно осложняет работу утилит, отличных от компилятора, работающих с исходными текстами, в стандарт эта возможность не входит. Здесь: Conditional Compilation (http://www.adacore.com/wp-content/files/auto_update/gnat-unw-docs/html/gnat_ugn_32.html) описывается, как можно организовать условно компилируемый код. В качестве резервного варианта предоставляется препроцессор gnatprep.
70. ↑ Условная компиляция/Java. Утверждения (операторы assert) всегда включаются компилятором в байт-код и могут быть разрешены (по умолчанию запрещены, то есть игнорируются) при запуске виртуальной машины ключом -ea/-enableassertion.
71. ↑ [1] (http://wdh.suncloud.ru/js13.htm#ref3134).
72. ↑ Макросы лиспа позволяют при компиляции вычислять произвольные выражения, включая, естественно, конструкции ветвлений. Кроме того, имеется также примерный аналог #ifdef из Си.[2] (http://www.lispworks.com/documentation/HyperSpec/Body/02_dhq.htm)[3] (http://www.lispworks.com/documentation/HyperSpec/Body/02_dhr.htm)
73. ↑ Компилятор должен решать, какие классы будут представлены «простыми» типами и будут, в том числе, размещаться в стеке.
74. ↑ Создание объектов на стеке/Haskell. В GHC при помощи Unboxed Types / Unboxed Arrays.
75. ↑ Стандарт языка предусматривает декларацию DYNAMIC-EXTENT (http://www.lispworks.com/documentation/HyperSpec/Body/d_dynami.htm), которая может трактоваться компилятором как указание выделить место под объект на стеке.
76. ↑ Создание объектов на стеке/Delphi. В Delphi имеется 2 объектных модели — старая (унаследована из Turbo Pascal) и новая. Создание объектов на стеке возможно только в старой объектной модели.
77. Через FFI (foreign function interface)
78. ↑ Можно с помощью модуля стандартной библиотеки — ctypes.
79. Позволяется в `unsafe`-блоках (участках кода, помеченных как небезопасные).
80. ↑ Неуправляемые указатели/Smalltalk. В Smalltalk есть возможность низкоуровневой работы с памятью, но только в адресном пространстве, предоставляемом виртуальной машиной.
81. ↑ unsafe + System.Runtime.InteropServices
82. ↑ Ручное управление памятью/Smalltalk. При низкоуровневой работе в пространстве памяти, предоставляемом виртуальной машиной, можно вручную создавать и удалять объекты, записывая данные в соответствующие адреса памяти. Аналогично можно вручную управлять размещением объектов в памяти.
83. ↑ Сборка мусора/Ada. Только на некоторых платформах (.NET и JVM) или при помощи библиотек (AdaCL:GC (http://adacl.sourceforge.net/pmwiki.php/Main/GarbageCollector)). Тем не менее, практически все программы на Ada могут работать как с ним, так и без него. В этом смысле к сборке мусора применительно к Аде следует относиться не как к инженерному решению, а как к оптимизации управления памятью.
84. ↑ Сборка мусора/C. В стандарте языка и в стандартных библиотеках нет сборки мусора. Однако существуют сборщики мусора для C и C++ в виде библиотек. Например, BoehmGC (англоязычный раздел).
85. ↑ В новом стандарте C++0x предполагается сборка мусора для интеллектуальных указателей
86. ↑ Сборка мусора/Rust. Через `Rc` или `Arc` — умные указатели со счётчиком ссылок, часть стандартной библиотеки. Следует отметить, что синтаксис языка позволяет вывести время жизни объекта статически, во время компиляции, что делает динамическую сборку мусора в большинстве случаев ненужной.
87. ↑ Сборка мусора/Delphi. Если не считать Delphi.NET.
88. ↑ Инструкция goto/Java. Является зарезервированным словом.
89. ↑ Специальный оператор GO (http://www.lispworks.com/documentation/HyperSpec/Body/s_go.htm). Все конструкции циклов в CL, фактически, являются макросами-надстройками над этой инструкцией.
90. ↑ Целевая метка должна находиться в том же файле, в том же контексте. Имеется ввиду, что вы не можете ни перейти за границы функции или метода, ни перейти внутрь одной из них [4] (http://php.net/manual/ru/control-structures.goto.php).
91. ↑ Инструкция goto/Ruby. В языке goto нет, но есть библиотека (http://raa.ruby-lang.org/project/ruby-goto/) реализующая его.
92. ↑ Инструкция goto/Smalltalk. В стандарте языка goto нет, но существуют библиотеки, реализующие функциональность goto через управление стеком исполнения. Используются крайне редко, это скорее proof of concept (англ.).
93. ↑ Макрос RETURN (http://www.lispworks.com/documentation/HyperSpec/Body/m_return.htm). Фактически, является частным случаем RETURN-FROM (http://www.lispworks.com/documentation/HyperSpec/Body/s_ret_fr.htm).
94. ↑ заменяется исключениями, также реализуется с помощью Camlp4 http://code.google.com/p/ocaml-break-continue/
95. ↑ Специальный оператор RETURN-FROM (http://www.lispworks.com/documentation/HyperSpec/Body/s_ret_fr.htm)
96. ↑ Принимает необязательный числовой аргумент, который сообщает ему выполнение какого количества вложенных структур необходимо прервать [5] (http://php.net/manual/ru/control-structures.break.php).
97. ↑ Есть возможность указать число вложенных циклов, которые нужно прервать
98. ↑ Можно либо повторить выполнение метода, либо пробросить исключение далее
99. ↑ Java-style try-catch блок реализуется макросом handler-case. Кроме того, в возможности системы обработки исключений Common Lisp входит система т. н. перезапусков(restarts), которые позволяют обрабатывать исключения «изнутри» без раскрутки стека вызовов функций
100. ↑ При помощи оператора eval
101. Вместо механизма исключений, в Rust применяется сопоставление опционального значения с образцом, то есть обязательная проверка наличия значения или ошибки.
102. При использовании библиотеки PBOSL (http://pbosl.purearea.net)
103. ↑ Блок finally/Ada. В стандарте языка finally нет, но существуют библиотеки, реализующие функциональность finally (http://www.ada-ru.org/smpl-final). Используются крайне редко, это скорее proof of concept (англ.).
104. ↑ MDN — MDC (http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference:Statements:try...catch)
105. ↑ Специальный оператор UNWIND-PROTECT (http://www.lispworks.com/documentation/HyperSpec/Body/s_unwind.htm)
106. Начиная с версии 5.5
107. ↑ реализуется на camlp4 http://bluestorm.info/camlp4/dev/try/pa_tryfinally.ml.html
108. ↑ При помощи нескольких последовательных catch
109. ↑ Java-style try-catch блок реализуется макросом handler-case. Кроме того, в возможности системы обработки исключений Common Lisp входит система т. н. перезапусков(restarts), которые позволяют обрабатывать исключения «сверху» без раскрутки стека вызовов функций
110. ↑ При помощи eval or {…}

111. ↑ Реализован перезапуск тела активного объекта (активности), для этого тело объекта помечается модификатором {SAFE}
112. ↑ Частично реализуются нестандартным модулем Runops::Resume (http://use.perl.org/~chromatic/journal/27491)
113. ↑ Ключевое слово retry (http://www.ruby-doc.org/docs/ProgrammingRuby/html/tut_exceptions.html)
114. ↑ Конструкции yield return, запросы LINQ, в FCL 4.0 войдёт тип Lazy.
115. ↑ Seq-генераторы, модуль Lazy стандартной библиотеки F#.
116. ↑ Однако, данную возможность можно реализовать на макросах
117. ↑ Данная возможность реализована на макросах
118. ↑ Spl-интерфейсы итераторов и конструкция yield, начиная с версии 5.5.
119. ↑ Ленивые вычисления/Ruby. В языке ленивых вычислений нет, но есть библиотека (http://moonbase.rydia.net/software/lazy.rb/) реализующая их.
120. ↑ Конструкции Linq.
121. ↑ модуль Lazy стандартной библиотеки Ocaml.
122. ↑ setcontext et al. (UNIX System V and GNU libc)
123. ↑ Конструкции yield return и await
124. ↑ Реазилуется сторонними библиотеками, например cl-cont (http://common-lisp.net/project/cl-cont/)
125. ↑ Только для байт-кода http://okmij.org/ftp/Computation/Continuations.html#caml-shift
126. ↑ Fibers (http://ddili.org/ders/d.en/fibers.html)
127. ↑ Легковесные процессы/Java. Вплоть до Java 1.1.
128. ↑ Только в некоторых реализациях.
129. ↑ Следует заметить что это не стандартные легкие процессы (http://perldoc.perl.org/threads.html)
130. ↑ Легковесные процессы/Python. Используя Stackless Python (http://www.stackless.com/).
131. ↑ Класс Fiber в Руби 1.9+
132. ↑ Легковесные процессы/Rust. В библиотеке futures (https://docs.rs/futures).
133. ↑ Монадические потоки выполнения, реализованы в библиотеке Lwt
134. ↑ PHP позволяет возвращать из функции/метода массив и разворачивать его конструкцией list что работает также как кортежи, с версии 5.4 появилась возможность разворачивать возвращаемые массивы сразу(array dereferencing)
135. Алгебраические типы данных/Ada и Delphi. Через механизм вариантных записей.
136. В динамических языках механизм алгебраических типов данных не имеет смысла.
137. ↑ Через механизм case-классов
138. Массивы/Haskell. С помощью Data.Array.
139. ↑ Динамические массивы/C. «Из коробки» данной возможности нет, однако похожий функциональность можно реализовать, используя функцию realloc.
140. ↑ Динамические массивы/Java. С помощью java.util.Vector (в стандартной библиотеке).
141. ↑ Динамические массивы/Rust. `Vec` в стандартной библиотеке.
142. ↑ map и unordered_map в стандартной библиотеке
143. ↑ Ассоциативные массивы/Java. С помощью java.util.HashMap (в стандартной библиотеке).
144. ↑ Ассоциативные массивы/Haskell. С помощью Data.Map
145. ↑ Ассоциативные массивы/Rust. `HashMap` в стандартной библиотеке.
146. ↑ Контроль границы массивов/C++. Для массивов контроля нет, однако в контейнерах STL, таких как std::vector, std::array есть метод at с контролем границ.
147. Контроль границ массивов/Perl, PHP и JavaScript. В языке нет массивов со статическими границами, присваивание элементу за текущими границами массива просто расширяет границы массива.
148. ↑ Существует только в виде SPL структуры данных. Стандартные массивы не предоставляют контроля границ — присваивание элементу за текущими границами массива — просто расширяет границы массива.
149. ↑ Контроль границ массивов/Ocaml. Можно отключить на этапе компиляции с помощью ключа -unsafe
150. ↑ Цикл foreach/Ada. Методы Iterate и Reverse_Iterate различных контейнеров, входящих в библиотеку Ada.Containers.
151. ↑ Цикл foreach/C++. В C++11 `for(auto x : some_array) {}` — не может изменять элементы, `for(auto& x : some_array){}` — может изменять элементы.
152. ↑ Цикл foreach/Erlang. В виде функции `foreach/3` из модуля `lists`.
153. ↑ Цикл foreach/JavaScript. С версии 1.6 [6] (http://developer.mozilla.org/en/docs/New_in_JavaScript_1.6).
154. ↑ Цикл foreach/Lisp. Макрос LOOP (http://www.lispworks.com/documentation/HyperSpec/Body/m_loop.htm) в составе стандартной библиотеки. Представляет собой «язык в языке» с большим количеством возможностей.
155. ↑ List comprehensions/C#. «Query Comprehension» можно считать за List Comprehension только с большой натяжкой.
156. ↑ LOOP et al.
157. Целые числа произвольной длины/.NET. Посредством типа `System.Numerics.BigInteger`, включенного в FCL версии 4.0.
158. ↑ Целые числа произвольной длины/Java. С помощью классов `BigInteger` и `BigDecimal`.
159. ↑ Для вычислений с произвольной точностью PHP предоставляет Двоичный калькулятор, который поддерживает числа любого размера и точности, представленные в виде строк [7] (http://php.net/manual/ru/book.bc.php) .
160. ↑ Целые числа произвольной длины/Rust. `BigInt` в библиотеке num (https://docs.rs/num/).
161. ↑ Целые числа произвольной длины/Scala. С помощью классов `BigInteger` и `BigDecimal`.
162. ↑ Целые числа произвольной длины/OCaml. В помощью модуля `Num` и `Big_int`.
163. ↑ Пример: Тип (INTEGER 0 9) включает в себя все цифры от 0 до 9
164. ↑ Целые числа произвольной длины/Perl. С помощью модуля `Tie::Scalar`.
165. ↑ Интерфейсы традиционно реализуются структурами с указателями на функции, входящие в интерфейс. Пример реализации и использования — библиотеки OpenGL, OpenMAX AL и т. п..
166. ↑ Множественное наследование абстрактных классов
167. ↑ Похожая функциональность реализуется макросами и средствами CLOS.
168. ↑ Через множественное наследование от классов с методами-заготовками. См (http://perldesignpatterns.com/?AbstractClass)
169. ↑ Типажи (*traits*).
170. ↑ Могут быть реализованы с помощью паттерна Visitor(Посетитель)
171. ↑ Эмуляция через dynamic
172. Реализуется сторонними библиотеками
173. ↑ появятся(?) в Perl 6
174. ↑ Могут быть реализованы с помощью паттерна Visitor (Посетитель) (https://github.com/rust-unofficial/patterns/issues/55).
175. ↑ Могут быть реализованы с помощью наследования шаблонов Примесь (программирование)#.D0.AD.D0.BC.D1.83.D0.BB.D1.8F.D1.86.D0

176. ↑ Groovy — Category and Mixin transformations (http://groovy.codehaus.org/Category+and+Mixin+transformations)
177. ↑ Начиная с PHP версии 5.4 присутствует в виде trait
178. ↑ Через множественное наследование и/или изменение атрибутов произвольного объекта во время выполнения
179. ↑ Подмешивание реализации интерфейсов через ключевое слово implements. См. страницы 10-7 и 10-8 в Object Pascal Guide (http://docs.embarcadero.com/products/rad_studio/cbuilder6/EN/CB6_ObjPascalLangGuide_EN.pdf) .
180. ↑ Переименование членов при наследовании не поддерживается c++, однако можно сэмулировать через закрытое наследование, открывая члены, которые не нужно переименовать через директиву using, а если нужно переименовать — просто определить метод с новым названием и вызвать в нём метод родителя
181. В Rust нет наследования, только реализация типажей (*traits*, аналог интерфейсов).
182. ↑ Только совместное использование посредством виртуального наследования
183. ↑ Для каждого члена класса — выбор дублирование (через переименование), или слияние (иначе, если не было переопределения)
184. ↑ CLHS: Section 4.3.5 (http://www.lispworks.com/documentation/HyperSpec/Body/04_ce.htm)
185. ↑ Functions — D Programming Language 2.0 — Digital Mars (http://www.digitalmars.com/d/2.0/function.html#pure-functions)
186. ↑ в форме указателей на функции
187. ↑ std::function в c++0x
188. Появились в Delphi2009, как анонимные функции. Ранее — через указатели.
189. ↑ C++0x. Лямбда-выражения в C++0x (http://blog.olendarenko.org.ua/2009/08/c0x.html)
190. ↑ Анонимные делегаты присутствуют в языке с версии 2.0. В C# 3.0 появились полноценные анонимные функции.
191. ↑ Макрос LAMBDA (http://www.lispworks.com/documentation/HyperSpec/Body/m_lambda.htm)
192. ↑ С существенными ограничениями
193. ↑ lambda-функции в c++0x поддерживают замыкания как по ссылке, так и по значению
194. ↑ Через анонимные классы
195. ↑ Начиная с версии 5.3
196. ↑ Появились в Delphi2009, как анонимные функции.
197. ↑ boost::bind, std::bind1st, std::bind2nd или сэмулировать с помощью анонимных функций
198. с помощью возвращения делегата
199. ↑ С помощью Function.prototype.bind (http://msdn.microsoft.com/en-us/magazine/gg575560.aspx)
200. ↑ Реализуется сторонними библиотеками, например Sub::Curry (http://search.cpan.org/~lodin/Sub-Curry-0.8/lib/Sub/Curry.pm) и Sub::Curried (http://search.cpan.org/~osfameron/Sub-Curried-0.11/lib/Sub/Curried.pm)
201. ↑ functools.partial в стандартной библиотеке начиная с Python 2.5
202. По состоянию в Rust 1.19, может быть реализовано (https://rosettacode.org/wiki/Currying#Rust) при включении `#![feature(conservative_impl_trait)]`.
203. ↑ с помощью lambda-функций в c++0x
204. ↑ Proc#curry, появился в Ruby 1.9
205. ↑ Начиная с Delphi 2009
206. Макросы/C. Посредством препроцессора C.
207. ↑ Макросы/Haskell. Template Haskell — препроцессор, встроенный в GHC.
208. ↑ Фильтры [8] (http://perldoc.perl.org/perlfilter.html), в том числе, C/C++ препроцессор Filter::cpp
209. ↑ Встроены в Visual Studio (нет в Express Edition)
210. ↑ Штатный препроцессор camlp4
211. Неприменимо в языках с динамической типизацией.
212. ↑ Generics/Haskell. Прямых аналогов шаблонов в языке нет, однако имеются не менее мощные средства обобщенного программирования.
213. ↑ Generics/Delphi. Доступно начиная с Delphi 2009.
214. ↑ Unicode в идентификаторах/C. Доступно в компиляторах gcc начиная с 4.2
215. ↑ Unicode в идентификаторах/C++. Доступно в компиляторах от MS, начиная с MSVS++ 2005 и в gcc начиная с 4.2
216. ↑ В большинстве современных реализаций
217. ↑ Unicode в идентификаторах/Python. Доступно начиная с Python 3.0.
218. ↑ Unicode в идентификаторах/Ruby. Доступно начиная с Ruby 1.9.
219. ↑ По состоянию в Rust 1.19, поддерживается с некоторыми ограничениями (https://rosettacode.org/wiki/Unicode_variable_names#Rust) при включении `#![feature(non_ascii_idents)]`.
220. ↑ Перегрузка функций/JavaScript. Можно сымитировать, используя проверку передаваемых параметров с помощью рефлексии.
221. ↑ Обобщенные функции можно перегружать по типам или значениям нескольких параметров
222. ↑ Только перегрузка операторов [9] (http://perldoc.perl.org/overload.html).
223. Перегрузка функций и сопоставление с образцом/Python. Реализовано в сторонней библиотеке PEAK-rules (http://pypi.python.org/pypi/PEAK-Rules).
224. ↑ implicit-parameters (http://cvs.haskell.org/Hugs/pages/users_guide/implicit-parameters.html)
225. ↑ макросы DEFVAR (http://www.lispworks.com/documentation/HyperSpec/Body/m_defpar.htm) и DEFPARAMETER (http://www.lispworks.com/documentation/HyperSpec/Body/m_defpar.htm), а также декларация SPECIAL (http://www.lispworks.com/documentation/HyperSpec/Body/d_specia.htm), создают динамические биндинги.
226. Именованные аргументы и параметры по умолчанию/C#. Доступно начиная с C# 4.0.
227. ↑ Именованные параметры/JavaScript. Можно сымитировать, передав в качестве параметра функции объект: `f ({param1: "value1", param2: "value2"})`.
228. ↑ Спецификатор «&key» в списке аргументов объявляемой функции объявляет именованный параметр.
229. ↑ Именованные параметры/Smalltalk. Можно называть методы в стиле `сделатьЧтоНибудьС:используя:и:` — в таком случае двоеточия обозначают места, куда будут подставляться параметры при вызове метода, например `сделатьЧтоНибудьС: парам1 используя: парам2 и: парам3`. Названия подбирают таким образом, чтобы при вызове было понятно, для чего будут использоваться параметры.
230. ↑ Именованные параметры/Delphi: Могут использоваться при вызове OLE: `Word.Openfile(filename='1.doc')`
231. ↑ Значения параметров по умолчанию/Erlang. Можно сымитировать с помощью арности функции.
232. ↑ «&key» и «&optional» параметры допускают значения по умолчанию
233. ↑ Значения параметров по умолчанию/Perl. Можно элементарно сымитировать, см. [10] (http://www.devshed.com/c/a/Perl/Perl-Subroutines-Arguments-and-Values/1/).
234. ↑ Локальные функции/C. Поддерживаются в компиляторе gcc как нестандартное расширение языка, см. [11] (http://gcc.gnu.org/onlinedocs/gcc-4.1.2/gcc/Nested-Functions.html).
235. ↑ Локальные функции/C++. с помощью lambda-функций в c++0x

236. ↑ Локальные функции/Java. Внутри метода можно определять безымянные (анонимные) локальные классы, которые фактически позволяют создавать экземпляры объектов, перекрывающие методы своего класса.
237. ↑ Специальный оператор LABELS (http://www.lispworks.com/documentation/HyperSpec/Body/s_flet_.htm)
238. ↑ Макрос DESTRUCTURING-BIND (http://www.lispworks.com/documentation/HyperSpec/Body/m_destru.htm) и EQL спецификатор в обобщенных функциях (http://www.lispworks.com/documentation/HyperSpec/Body/t_eql.htm) можно рассматривать как аналоги некоторых подмножеств функциональности сопоставления с образцом.
239. Посредством библиотеки Code Contracts (http://research.microsoft.com/en-us/projects/contracts/) из состава FCL 4.0.
240. ↑ Контрактное программирование/Java. На основе аннотаций Java 5, используя библиотеку OVal (http://oval.sourceforge.net/) и аспектный компилятор AspectJ (http://www.eclipse.org/aspectj/), а также iContract [12] (http://www.javaworld.com/javaworld/jw-02-2001/jw-0216-cooltools.html) .
241. ↑ Контрактное программирование/Haskell. Посредством библиотеки QuickCheck.
242. DirectX через Net, OpenGL через стороннюю библиотеку OpenTK
243. ↑ большинство мультимедийных библиотек доступны через обёртку Derelict
244. ↑ libraries: cl-opengl (http://lisper.ru/wiki/libraries:cl-opengl)
245. ↑ Существует реализация OpenGL библиотеки для php — phpOpenGL project (http://sourceforge.net/projects/phpopengl/) (Зеркало (https://github.com/yoya/phpopengl/) на Github)
246. ↑ Объявления структур данных, констант и интерфейсов DirectX, OpenGL присутствуют в стандартной библиотеке языка, есть сторонние врапперы для большинства используемых библиотек, в том числе — игровых, звуковых и физических движков.

Терминология

Парадигмы

Императивная

Противоположность декларативному. Императивный язык должен описывать не столько саму задачу (описание, «ЧТО» нужно получить), сколько её решение («КАК» получить). Некоторыми авторами считается, что данное определение скорее относится к «процедурной» парадигме, которая, помимо императивного, включает в себя функциональное программирование.

Объектно-ориентированная

Основана на представлении всего в виде объектов, являющихся экземплярами того или иного класса и воплощает применение концепции абстрагирования. Объект при этом соединяет внутри себя как данные, так и методы, их обрабатывающие. Как правило, поддерживаются характерные возможности: наследование, инкапсуляция и полиморфизм. Некоторые авторы языки без наследования относят к просто «объектным».

Рефлексивная

Наличие в языке мощных механизмов интроспекции, функции eval. Возможность программы на данном языке оперировать собственным кодом как данными.

Функциональная

Позволяет записывать программу как композицию функций. В чистом функциональном языке нет переменных. Так как функции не имеют побочных эффектов, они могут выполняться в любом порядке.

Обобщенное программирование

Обобщенное программирование позволяет записывать алгоритмы, принимающие данные любого типа.

Логическая

Программа представляет собой описание фактов и правил вывода в некотором логическом исчислении. Желаемый результат, который часто записывается как вопрос, получается системой в результате попытки применения описанных правил — путём логического вывода. Интересными особенностями являются отсутствие детерминированности в общем случае, внутренняя склонность к распараллеливанию.

Доказательная

Направлен на разработку алгоритмов и программ с доказательствами их правильности с использованием спецификаций программ.

Декларативная

Противоположность императивному. Декларативный язык описывает не столько решение задачи, сколько саму задачу («ЧТО» нужно получить), а каким образом получить решение, уже должен определять компьютер.

Распределенная

Язык, содержащий специальные конструкции для поддержки распараллеливания программы на несколько компьютеров.

Типизация

Статическая типизация

(См. статическая типизация). Переменные и параметры методов/функций связываются с типами в момент объявления и не могут быть изменены позже.

Динамическая типизация

(См. динамическая типизация). Переменные и параметры методов/функций связываются с типами в момент присваивания значения (или передачи параметра в метод/функцию), а не в момент объявления переменной или параметра. Одна и та же переменная в разные моменты может хранить значения разных типов.

Явная типизация

Типы переменных и параметров указываются явно.

Неявная типизация

Типы переменных и параметров не указываются явно. Неявная типизация может быть и статической, в таком случае типы переменных и параметров вычисляются компилятором.

Явное приведение типов

Для использования переменной какого-либо типа там, где предполагается использование переменной другого типа, нужно (возможно) явно выполнить преобразование типа.

Неявное приведение типов без потери данных

Неявное приведение типов в таких ситуациях, где не происходит потери данных — например, использование целого числа там, где предполагалось использование числа с плавающей точкой.

Неявное приведение типов с потерей данных

Неявное приведение типов в таких ситуациях, где может произойти потеря данных — например, использование числа с плавающей точкой там, где предполагалось использование целого числа.

Неявное приведение типов в неоднозначных ситуациях

Например, использование строки там, где предполагалось число или наоборот. Классический пример: сложить число 1 со строкой «2» — результат может быть как число 3, так и строка «12». Другой пример — использование целого числа там, где ожидается логическое значение (boolean).

Алиасы типов

Возможность определить видимый глобально (за пределами единицы компиляции) алиас типа, полностью эквивалентный исходному типу. Например, typedef в Си. Директива using в C# не подходит под этот критерий из-за локальной области действия.

Вывод типов переменных из инициализатора

Возможность не указывать явно тип переменной, если для неё задан инициализатор. Если возможность действует для локальных переменных, но не действует для полей класса, все равно ставьте +. Характеристика не применима к языкам с динамической типизацией.

Вывод типов переменных из использования

Возможность не указывать явно тип переменной, если её тип может быть выведен из дальнейшего использования. Если возможность действует для локальных переменных, но не действует для полей класса, все равно ставьте +. Характеристика не применима к языкам с динамической типизацией.

Вывод типов-аргументов при вызове метода

Возможность не указывать явно типы-аргументы при вызове generic-метода, если они могут быть выведены из типов обычных аргументов.

Вывод сигнатуры для локальных функций

Может ли сигнатура локальной функции быть выведена из использования. Неприменимо для языков с динамической типизацией. Ставьте -, если язык не поддерживает локальные функции.

Параметрический полиморфизм

Наличие типобезопасного параметрического полиморфизма (aka generic types). Подразумевает возможность указывать constraints или type classes (http://en.wikipedia.org/wiki/Type_classes) для типов-параметров.

Параметрический полиморфизм с ковариантностью

Наличие ко- и контравариантных type parameters. В некоторых языках может быть лишь частичная поддержка (например, только в интерфейсах и делегатах). В таком случае, отмечайте +/-.

Параметрический полиморфизм высших порядков

Возможность создавать type constructors высших порядков (как в Scala). См. Towards Equal Rights for Higher-kinded Types (https://web.archive.org/web/20091229052347/http://www.cs.kuleuven.be/~adriaan/files/higher.pdf)

Информация о типах в runtime

Возможность узнать точный тип объекта в runtime.

Информация о типах-параметрах в runtime

Возможность узнать в runtime информацию о типе, с которым инстанциирован generic-тип. Если язык не поддерживает generic-типы, то ставьте -. Если информация о типах стирается в runtime (используется erasure), то ставьте -.

Компилятор/интерпретатор

Open-source компилятор (интерпретатор)

Наличие полноценного open-source компилятора (для интерпретируемых языков — интерпретатора). Если существует open-source компилятор, но он поддерживает не все возможности языка, то ставьте +/- или -/+.

Возможность компиляции

Возможность компиляции в нативный код или в byte-код с возможностью JIT-компиляции. Если язык компилируется в код на другом языке (например, C), который потом компилируется в нативный код, то тоже ставьте +.

Bootstrapping

Наличие полноценного bootstrapping-компилятора (то есть компилятора, написанного на том же языке, который он компилирует, и успешно компилирующего самого себя). Если существует bootstrapping-компилятор, но он поддерживает не все возможности языка, то ставьте +/- или -/+.

Многопоточная компиляция

Возможность компилятора на многопроцессорных системах использовать несколько потоков для ускорения компиляции. Если язык не поддерживает компиляцию, то ставьте x (неприменимо).

Интерпретатор командной строки

Возможность вводить инструкции языка строка за строкой с их немедленным выполнением. Может использоваться в качестве калькулятора.

Условная компиляция

Возможность включать/выключать части кода в зависимости от значения символов условной компиляции (например, с помощью #if … #endif в C++)

Управление памятью

Объекты на стеке

Возможность создавать экземпляры объектов не в куче, а на стеке.

Неуправляемые указатели

Наличие неуправляемых указателей, адресная арифметика, прямой доступ к памяти.

Ручное управление памятью

Возможность явного выделения и освобождения памяти в куче (например, с помощью операторов new и delete в C++).

Сборка мусора

Возможность использовать автоматический процесс сборки мусора (освобождения памяти в куче, занятой неиспользуемыми объектами).

Управление потоком вычислений

Инструкция goto

Поддержка инструкции goto (безусловный переход на метку).

Инструкция break без метки

Поддержка инструкции break без метки (безусловный выход из ближайшего цикла), и соответствующей инструкции continue. Наличие в языке инструкции break, относящегося к switch или другой конструкции, не влияет на это поле.

Инструкция break с меткой

Поддержка инструкции break с меткой (безусловный выход из цикла, помеченного меткой), и соответствующей инструкции continue. Наличие в языке инструкции break, относящегося к switch или другой конструкции, не влияет на это поле.

Поддержка try/catch

Поддержка обработки исключений с помощью try/catch или эквивалентной конструкции.

Блок finally

Поддержка блока finally при обработке исключений или эквивалентной конструкции.

Блок else (исключения)

Поддержка блока else при обработке исключений (действия, выполняющиеся при завершении блока try без исключения).

Перезапуски

Исключения, не раскручивающие стек вызовов. Возможность из места перехвата исключения вернуться в место установки перезапуска.

Легковесные процессы

Эмуляция многопоточности рантаймом самого языка. В пределах одного потока ОС (или нескольких) выполняется множество потоков исходного кода

Типы и структуры данных

Многомерные массивы

Наличие встроенных в язык многомерных массивов. Если язык поддерживает только массивы массивов, ставьте +/-

Динамические массивы

Наличие встроенных в язык динамических массивов (способных изменять свой размер во время выполнения программы). Если динамические массивы представлены только векторами (то есть только одномерными массивами) или векторами векторов, ставьте +/-

Ассоциативные массивы

Наличие встроенных в язык ассоциативных массивов или хэш-таблиц.

Цикл foreach

Наличие возможности перебрать все элементы коллекции с помощью цикла foreach. Если в языке есть эквивалентная или более сильная возможность (наподобие list comprehensions), то будет +.

Списковые включения

Наличие списковых включений (или их аналога).

Кортежи

Возможность вернуть из функции/метода кортеж (tuple) — неименованный тип данных, содержащий несколько безымянных полей произвольного типа.

Целые числа произвольной длины

Поддержка целых чисел неограниченной разрядности. Должна быть возможность записать сколь угодно большое целое число с помощью литерала.

Целые числа с контролем границ

Возможность определить тип, значениями которого могут быть целые числа только определенного интервала, например [-5..27], при этом присвоение переменной такого типа значения, выходящего за указанные рамки, должно вызывать ошибку.

Объектно-ориентированные возможности

Интерфейсы

Семантическая и синтаксическая конструкция в коде программы, используемая для специфицирования услуг, предоставляемых классом.

Множественное наследование

Возможность наследовать класс сразу от нескольких классов (не интерфейсов).

Мультиметоды

Динамическая (run time) диспетчеризация функции в зависимости от типов нескольких аргументов.
В языках с «message passing» ООП похожая функциональность реализуется паттерном «Visitor».

Переименование членов при наследовании

Возможность в наследнике изменить имя поля/метода предка.

Решение конфликта имен при множественном наследовании

При множественном наследовании — решение для случая ромбовидного наследования (B потомок A, C потомок A, D потомок B и C). Решение может приниматься как для всего класса, так и для каждого поля/метода в отдельности.

Функциональные возможности

First class functions

Функции в данном языке являются объектами первого класса.

Лексические замыкания

Возможность использовать локальную или лямбда-функцию (анонимный делегат) за пределами функции-контейнера с автоматическим сохранением контекста (локальных переменных) функции-контейнера

Частичное применение

Возможность фиксировать часть аргументов функции, то есть имея функцию $f\colon (A \times B) \to C$, создать функцию $P(f,a)\colon B \to C$, где $(P(f,a))(b) = f(a,b)$. Не следует путать с каррированием (оператор каррирования — один из вариантов реализации частичного применения).

Разное

Макросы

Наличие в языке макро-системы, обрабатывающей код программы до времени её компиляции и/или выполнения. Например, макросы Лиспа, препроцессор Си или шаблоны C++.

Шаблоны/Generics

Наличие в данном статически типизированном языке инструмента для обобщенного программирования, наподобие templates в C++ или generics в C#.

Поддержка Unicode в идентификаторах

Возможность включения Unicode-символов (например, букв национальных алфавитов) в идентификаторы.

Перегрузка функций

Возможность перегрузки функций/методов по количеству и типам параметров.

Динамические переменные

Возможность создавать переменные, имеющие динамическую область видимости (англ.).

Именованные параметры

Возможность при вызове функции/метода указывать имена параметров и менять их местами.

Значения параметров по умолчанию

Возможность при вызове функции/метода опускать некоторые параметры, чтобы при этом подставлялось значение по умолчанию, указанное при определении функции.

Локальные функции

Возможность определять локальную функцию внутри другой функции/метода. Подразумевается возможность использовать внутри локальной функции локальные переменные из внешнего блока.

Сопоставление с образцом

Наличие сопоставления с образцом.

Контрактное программирование

Возможность задавать пред- и пост-условия для методов и инварианты для классов.

Ссылки

- Таблица сравнения языков от создателей D (+ обсуждение на RSDN) (http://www.rsdn.ru/forum/message/1905860.1.aspx)
- Созданная на её основе поклонниками других языков более объемлющая таблица (http://www.prowiki.org/wiki4d/wiki.cgi?LanguagesVersusD) (англ.)
- Microbenchmarking C++, C#, and Java (http://www.drdobbs.com/article/printableArticle.jhtml;jsessionid=CWK4ZLY0DXK2JQE1GHRSKHWATMY32JVN?articleId=184401976) (англ.)
