

Материал из Википедии — свободной энциклопедии

C++ (читается *си-плюс-плюс*^{[2][3]}) — компилируемый, статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков.^{[4][5]} В сравнении с его предшественником — языком C — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.^[5]

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования^{[мнения 1][мнения 2]}. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также игр. Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ. Например, на платформе x86 это GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder и другие. C++ оказал огромное влияние на другие языки программирования, в первую очередь на Java и C#.

Синтаксис C++ унаследован от языка C. Одним из принципов разработки было сохранение совместимости с C. Тем не менее C++ не является в строгом смысле надмножеством C; множество программ, которые могут одинаково успешно

	<div><div><div><div><div><div></div></div></div><div><div><div></div><div></div></div></div><div><div><div></div><div></div></div></div><div><div><div></div></div></div></div></div></div> <div>C++</div>
--	--

транслироваться как компиляторами С, так и компиляторами С++, довольно велико, но не включает все возможные программы на С.

Содержание

История

- Создание
- Развитие и стандартизация языка
- История названия
- Философия С++

Обзор языка

- Типы
- Наследование
- Полиморфизм
- Инкапсуляция
- Друзья
- Специальные функции
- Шаблоны

Стандартная библиотека

- Общая структура
- Состав
- Реализации

Отличия от языка С

- Совместимость с языком С
- Новые возможности
- С++ не включает в себя С
- Средства С, которых рекомендуется избегать

Дальнейшее развитие

- Общие направления развития С++
- Стандарт С++11: дополнения в ядре языка

Примеры программ

- Пример № 1
- Пример № 2

Критика

- О критике С++ в целом
- Сравнение с альтернативными языками
 - С++ и Ада
 - С++ и Java
 - С++ и С
 - С++ и функциональные и скриптовые языки

Mars C++, Oracle Solaris Studio
C++ compiler, Turbo C++

Диалекты ISO/IEC 14882 C++

Испытал влияние Си, Симула, Алгол 68, Клу, ML и Ада

Сайт isocpp.org (англ.)



Медиафайлы на Викискладе

Критика отдельных элементов и концепций

Контроль за поведением

Компонентное и объектно-ориентированное программирование

Метапрограммирование

Кроссплатформенность

Отсутствие возможностей

Избыточные и опасные возможности

Встроенные средства обхода ограничений

Неконтролируемая макроподстановка

Проблемы перегрузки

Вычислительная эффективность

Результирующий объём исполнимого кода

Потенциал оптимизации

Эффективное управление памятью

Результативность

Качество и культура программирования

Исправление исправного

Сложность ради самой сложности

Саботаж

Ненадёжность продукта

Менеджмент проектов

Влияние и альтернативы

См. также

Примечания

Литература

Ссылки

История

Создание

Язык возник в начале 1980-х годов, когда сотрудник фирмы Bell Labs Бьёрн Страуструп придумал ряд усовершенствований к языку C под собственные нужды. ^[7] Когда в конце 1970-х годов Страуструп начал работать в Bell Labs над задачами теории очередей (в приложении к моделированию телефонных вызовов), он обнаружил, что попытки применения существующих в то время языков моделирования оказываются неэффективными, а применение высокоэффективных машинных языков слишком сложно из-за их ограниченной выразительности. Так, язык Симула имеет такие возможности, которые были бы очень полезны для разработки большого программного

обеспечения, но работает слишком медленно, а язык BCPL достаточно быстр, но слишком близок к языкам низкого уровня и не подходит для разработки большого программного обеспечения.

Вспомнив опыт своей диссертации, Страуструп решил дополнить язык C (преемник BCPL) возможностями, имеющимися в языке Симула. Язык C, будучи базовым языком системы UNIX, на которой работали компьютеры Bell,

является быстрым, многофункциональным и переносимым. Страуструп добавил к нему возможность работы с классами и объектами. В результате практические задачи моделирования оказались доступными для решения как с точки зрения времени разработки (благодаря использованию Симула-подобных классов), так и с точки зрения времени вычислений (благодаря быстройдействию C). В первую очередь в C были добавлены классы (с инкапсуляцией), наследование классов, строгая проверка типов, inline-функции и аргументы по умолчанию. Ранние версии языка, первоначально именовавшегося «C with classes» («Си с классами»), стали доступны с 1980 года.

Исторический этап развития ^[6]	Год
Язык BCPL	1966
Язык Би (оригинальная разработка Томпсона под UNIX)	1969
Язык Си	1972
Си с классами	1980
C84	1984
Cfront (выпуск E)	1984
Cfront (выпуск 1.0)	1985
Множественное/виртуальное наследование	1988
Обобщённое программирование (шаблоны)	1991
ANSI C++ / ISO-C++	1996
ISO/IEC 14882:1998	1998
ISO/IEC 14882:2003	2003
C++/CLI	2005
TR1	2005
C++11	2011
C++14	2014
C++17	2017
C++20	2020

Разрабатывая C с классами, Страуструп написал программу cfront — транслятор, перерабатывающий исходный код C с классами в исходный код простого C. Это позволило работать над новым языком и использовать его на практике, применяя уже имеющуюся в UNIX инфраструктуру для разработки на C. Новый язык, неожиданно для автора, приобрёл большую популярность среди коллег и вскоре Страуструп уже не мог лично поддерживать его, отвечая на тысячи вопросов.

К 1983 году в язык были добавлены новые возможности, такие как виртуальные функции, перегрузка функций и операторов, ссылки, константы, пользовательский контроль над управлением свободной памятью, улучшенная проверка типов и новый стиль комментариев (//). Получившийся язык уже перестал быть просто дополненной версией классического C и был переименован из C с классами в «C++». Его первый коммерческий выпуск состоялся в октябре 1985 года.

До начала официальной стандартизации язык развивался в основном силами Страуструпа в ответ на запросы программистского сообщества. Функцию стандартных описаний языка выполняли написанные Страуструпом печатные работы по C++ (описание языка, справочное руководство и так далее). Лишь в 1998 году был ратифицирован международный стандарт языка C++: ISO/IEC 14882:1998 «Standard for the C++ Programming Language»; после принятия технических исправлений к стандарту в 2003 году — следующая версия этого стандарта — ISO/IEC 14882:2003.^[8]

Развитие и стандартизация языка

В 1985 году вышло первое издание «Языка программирования C++», обеспечивающее первое описание этого языка, что было чрезвычайно важно из-за отсутствия официального стандарта. В 1989 году состоялся выход C++ версии 2.0. Его новые возможности включали множественное наследование, абстрактные классы, статические функции-члены, функции-константы и защищённые члены. В 1990 году вышло «Комментированное справочное руководство по C++», положенное впоследствии в основу стандарта. Последние обновления включали шаблоны, исключения, пространства имён, новые способы приведения типов и булевский тип.

Стандартная библиотека C++ также развивалась вместе с ним. Первым добавлением к стандартной библиотеке C++ стали потоки ввода-вывода, обеспечивающие средства для замены традиционных функций `C printf` и `scanf`. Позднее самым значительным развитием стандартной библиотеки стало включение в неё Стандартной библиотеки шаблонов.

В 1998 году был опубликован стандарт языка ISO/IEC 14882:1998 (известный как C++98),^[9] разработанный комитетом по стандартизации C++ (ISO/IEC JTC1/SC22/WG21 working group). Стандарт C++ не описывает способы именования объектов, некоторые детали обработки исключений и другие возможности, связанные с деталями реализации, что делает несовместимым объектный код, созданный различными компиляторами. Однако для этого третьими лицами создано множество стандартов для конкретных архитектур и операционных систем.

В 2003 году был опубликован стандарт языка ISO/IEC 14882:2003, где были исправлены выявленные ошибки и недочёты предыдущей версии стандарта.

В 2005 году был выпущен отчёт Library Technical Report 1 (кратко называемый TR1). Не являясь официально частью стандарта, отчёт описывает расширения стандартной библиотеки, которые, как ожидалось авторами, должны быть включены в следующую версию языка C++. Степень поддержки TR1 улучшается почти во всех поддерживаемых компиляторах языка C++.

С 2009 года велась работа по обновлению предыдущего стандарта. Предварительная версия нового стандарта сначала называлась C++09, спустя год — C++0x. Стандарт был опубликован в 2011 году и получил название C++11. В него были включены дополнения в ядро языка и расширение стандартной библиотеки, в том числе большую часть TR1. Следующая версия стандарта, C++14, вышла в августе 2014 года. Она содержит в основном уточнения и исправления ошибок предыдущей версии.

Последняя на текущий момент действующая версия стандарта — C++17, опубликованная в декабре 2017 года. Основным изменением стало введение в стандартную библиотеку параллельных версий стандартных алгоритмов и удаление устаревших и крайне редко используемых элементов.

C++ продолжает развиваться, чтобы отвечать современным требованиям. Одна из групп, разрабатывающих язык C++ и направляющих комитету по стандартизации C++ предложения по его улучшению — это Boost, которая занимается, в том числе, совершенствованием возможностей языка путём добавления в него особенностей метапрограммирования.

Никто не обладает правами на язык C++, он является свободным. Однако сам документ стандарта языка (за исключением черновиков) не доступен бесплатно.^[10] В рамках процесса стандартизации, ISO выпускает несколько видов изданий. В частности, технические доклады и технические характеристики публикуются, когда «видно будущее, но нет немедленной возможности соглашения для публикации международного стандарта.» До 2011 года было опубликовано три технических отчёта по C++: TR 19768: 2007 (также известный как C++, Технический отчёт 1) для расширений библиотеки в основном интегрирован в C++11, TR 29124: 2010 для специальных математических

функций, и TR 24733: 2011 для десятичной арифметики с плавающей точкой. Техническая спецификация DTS 18822: 2 014 (по файловой системе) была утверждена в начале 2015 года, и остальные технические характеристики находятся в стадии разработки и ожидают одобрения^[11]

В марте 2016 года в России была создана рабочая группа РГ21 C++. Группа была организована для сбора предложений к стандарту C++, отправки их в комитет и защиты на общих собраниях Международной организации по стандартизации (ISO)^[12].

История названия

Имя языка, получившееся в итоге, происходит от оператора унарного постфиксного инкремента C++ (увеличение значения переменной на единицу). Имя C+ не было использовано потому, что является синтаксической ошибкой в C и, кроме того, это имя было занято другим языком. Язык также не был назван D, поскольку «является расширением C и не пытается устранять проблемы путём удаления элементов C».^[7]

Философия C++

В книге «Дизайн и эволюция C++»^[13] Бьёрн Страуструп описывает принципы, которых он придерживался при проектировании C++. Эти принципы объясняют, почему C++ именно такой, какой он есть. Некоторые из них:

- Получить универсальный язык со статическими типами данных, эффективностью и переносимостью языка C.
- Непосредственно и всесторонне поддерживать множество стилей программирования, в том числе процедурное программирование, абстракцию данных, объектно-ориентированное программирование и обобщённое программирование.
- Дать программисту свободу выбора, даже если это даст ему возможность выбирать неправильно.
- Максимально сохранить совместимость с C, тем самым делая возможным лёгкий переход от программирования на C.
- Избежать разночтений между C и C++: любая конструкция, допустимая в обоих языках, должна в каждом из них обозначать одно и то же и приводить к одному и тому же поведению программы.
- Избегать особенностей, которые зависят от платформы или не являются универсальными.
- «Не платить за то, что не используется» — никакое языковое средство не должно приводить к снижению производительности программ, не использующих его.
- Не требовать слишком усложнённой среды программирования.

Обзор языка

Стандарт C++ состоит из двух основных частей: описание ядра языка и описание стандартной библиотеки.

Первое время язык развивался вне формальных рамок, спонтанно, по мере встававших перед ним задач. Развитию языка сопутствовало развитие кросс-компилятора cffront. Новшества в языке отражались в изменении номера версии кросс-компилятора. Эти номера версий кросс-компилятора

распространялись и на сам язык, но применительно к настоящему времени речь о версиях языка C++ не ведут. Лишь в 1998 году язык стал стандартизированным.

- C++ поддерживает как комментарии в стиле C (`/* комментарий */`), так и однострочные (`//` вся оставшаяся часть строки является комментарием), где `//` обозначает начало комментария, а ближайший последующий символ новой строки, который не предварён символом `\` (либо эквивалентным ему обозначением `??/`), считается окончанием комментария. Плюс этого комментария в том, что его не обязательно заканчивать, то есть обозначать окончание комментария.
- Спецификатор `inline` для функций. Функция, определённая внутри тела класса, является `inline` по умолчанию. Данный спецификатор является подсказкой компилятору и может встроить тело функции в код вместо её непосредственного вызова.
- Квалификаторы `const` и `volatile`. В отличие от C, где `const` обозначает только доступ на чтение, в C++ переменная с квалификатором `const` должна быть инициализирована. `volatile` используется в описании переменных и информирует компилятор, что значение данной переменной может быть изменено способом, который компилятор не в состоянии отследить. Для переменных, объявленных `volatile`, компилятор не должен применять средства оптимизации, изменяющие положение переменной в памяти (например, помещающие её в регистр) или полагающиеся на неизменность значения переменной в промежутке между двумя присваиваниями ей значения. В многоядерной системе `volatile` помогает избегать барьеров памяти 2-го типа.
- Пространства имён (`namespace`). Пример:

```
namespace Foo
{
    const int x = 5;
}
const int y = Foo::x;
```

Специальным случаем является безымянное пространство имён. Все имена, описанные в нём, доступны только в текущей единице трансляции и имеют локальное связывание. Пространство имён `std` содержит в себе стандартные библиотеки C++.

- Для работы с памятью введены операторы `new`, `new[]`, `delete` и `delete[]`. В отличие от библиотечных `malloc` и `free`, пришедших из C, данные операторы производят инициализацию объекта. Для классов это вызов конструктора, для POD типов инициализацию можно либо не проводить (`new Pod;`), либо провести инициализацию нулевыми значениями (`new Pod(); new Pod{};`).

Типы

В C++ доступны следующие встроенные типы. Типы C++ практически полностью повторяют типы данных в C:

- Символьные: `char`, `wchar_t` (`char16_t` и `char32_t`, в стандарте C++11).
- Целочисленные знаковые: `signed char`, `short int`, `int`, `long int` (и `long long`, в стандарте C++11).
- Целочисленные беззнаковые: `unsigned char`, `unsigned short int`, `unsigned int`, `unsigned long int` (и `unsigned long long`, в стандарте C++11).
- С плавающей точкой: `float`, `double`, `long double`.
- Логический: `bool`, имеющий значения `true` или `false`.

Операции сравнения возвращают тип `bool`. Выражения в скобках после `if`, `while` приводятся к типу `bool`.^[14]

Язык ввёл понятие ссылок, а начиная с одиннадцатой версии стандарта *rvalue*-ссылки и передаваемые ссылки (англ. *forwarding reference*). (см. Ссылка (C++))

C++ добавляет к C объектно-ориентированные возможности. Он вводит классы, которые обеспечивают три самых важных свойства ООП: инкапсуляцию, наследование и полиморфизм.

В стандарте C++ под классом (`class`) подразумевается пользовательский тип, объявленный с использованием одного из ключевых слов `class`, `struct` или `union`, под структурой (`structure`) подразумевается класс, определённый через ключевое слово `struct`, и под объединением (`union`) подразумевается класс, определённый через ключевое слово `union`.

В теле определения класса можно указать как объявления функций, так и их определение. В последнем случае функция является встраиваемой (`inline`). Нестатические функции-члены могут иметь квалификаторы `const` и `volatile`, а также ссылочный квалификатор (`&` или `&&`).

Наследование

C++ поддерживает множественное наследование. Базовые классы (классы-предки) указываются в заголовке описания класса, возможно, со спецификаторами доступа. Наследование от каждого класса может быть публичным, защищённым или закрытым:

Доступ члена базового класса/режим наследования	private-член	protected-член	public-член
private-наследование	недоступен	private	private
protected-наследование	недоступен	protected	protected
public-наследование	недоступен	protected	public

По умолчанию базовый класс наследуется как `private`.

В результате наследования класс-потомок получает все поля классов-предков и все их методы; можно сказать, что каждый экземпляр класса-потомка содержит *подэкземпляр* каждого из классов-предков. Если один класс-предок наследуется несколько раз (это возможно, если он является предком нескольких базовых классов создаваемого класса), то экземпляры класса-потомка будут включать столько же подэкземпляров данного класса-предка. Чтобы избежать такого эффекта, если он нежелателен, C++ поддерживает концепцию *виртуального наследования*. При наследовании базовый класс может объявляться виртуальным; на все виртуальные вхождения класса-предка в дерево наследования класса-потомка в потомке создаётся только один подэкземпляр.

Полиморфизм

C++ поддерживает динамический полиморфизм и параметрический полиморфизм.

Параметрический полиморфизм представлен:

- Аргументами по умолчанию для функций. К примеру, для функции `void f(int x, int y=5, int z=10)`, вызовы `f(1)`, `f(1, 5)` и `f(1, 5, 10)` эквивалентны.
- Перегрузка функций: функция с одним именем может иметь разное число и разные по типу аргументы. Например :


```
void Print(int x);
void Print(double x);
void Print(int x, int y);
```

Частным случаем перегрузки функций можно считать перегрузку операторов.

■ Механизмом шаблонов

Динамический полиморфизм реализуется с помощью виртуальных методов и иерархии наследования. Полиморфным в C++ является тип имеющий хотя бы один виртуальный метод. Пример иерархии:

```
class Figure
{
public:
    virtual void Draw() = 0; // чистый виртуальный метод
    virtual ~Figure();       // при наличии хотя бы одного виртуального метода деструктор
                             // следует сделать виртуальным
};

class Square : public Figure
{
public:
    void Draw() override;
};

class Circle : public Figure
{
public:
    void Draw() override;
};
```

Здесь класс Figure является абстрактным (и, даже, интерфейсным), так как метод Draw не определён. Объекты данного класса нельзя создать, зато можно использовать ссылки или указатели с типом Figure. Выбор реализации метода Draw будет производиться во время выполнения исходя из реального типа объекта.

Инкапсуляция

Инкапсуляция в C++ реализуется через указание уровня доступа к членам класса: они бывают публичными (открытыми, public), защищёнными (protected) и приватными (закрытыми, private). В C++ структуры формально отличаются от классов лишь тем, что по умолчанию уровень доступа к членам класса и тип наследования у структуры публичные, а у класса — приватные.

Доступ	private	protected	public
Сам класс	да	да	да
Друзья	да	да	да
Наследники	нет	да	да
Извне	нет	нет	да

Проверка доступа происходит во время компиляции, попытка обращения к недоступному члену класса вызовет ошибку компиляции.

Друзья

Функции-друзья — это функции, не являющиеся функциями-членами и тем не менее имеющие доступ к защищённым и закрытым членам класса. Они должны быть объявлены в теле класса как `friend`. Например:

```
class Matrix {  
    friend Matrix Multiply(Matrix m1, Matrix m2);  
};
```

Здесь функция `Multiply` может обращаться к любым полям и функциям-членам класса `Matrix`.

Дружественным может быть объявлен как весь класс, так и функция-член класса. Четыре важных ограничения, накладываемых на отношения дружественности в C++:

- **Дружественность не транзитивна.** Если `A` объявляет другом `B`, а `B`, в свою очередь, объявляет другом `C`, то `C` не становится автоматически другом для `A`. Для этого `A` должен явно объявить `C` своим другом.
- **Дружественность не взаимна.** Если класс `A` объявляет другом класс `B`, то он не становится автоматически другом для `B`. Для этого должно существовать явное объявление дружественности `A` в классе `B`.
- **Дружественность не наследуется.** Если `A` объявляет класс `B` своим другом, то потомки `B` не становятся автоматически друзьями `A`. Для этого каждый из них должен быть объявлен другом `A` в явной форме.
- **Дружественность не распространяется на потомков.** Если класс `A` объявляет `B` другом, то `B` не становится автоматически другом для классов-потомков `A`. Каждый потомок, если это нужно, должен объявить `B` своим другом самостоятельно.

В общем виде это правило можно сформулировать следующим образом: «Отношение дружественности существует только между теми классами (классом и функцией), для которых оно явно объявлено в коде, и действует только в том направлении, в котором оно объявлено».

Специальные функции

Класс по умолчанию может иметь шесть специальных функций: конструктор по умолчанию, конструктор копирования, конструктор перемещения, деструктор, оператор присваивания копированием, оператор присваивания перемещением. Также можно явно определить их все (см. Правило трёх).

```
class Array {  
public:  
    Array() = default; // компилятор создаст конструктор по умолчанию сам  
    Array(size_t _len) :  
        len(_len) {  
        val = new double[_len];  
    }  
    Array(const Array & a) = delete; // конструктор копирования явно удалён  
    Array(Array && a); // конструктор перемещения  
    ~Array() {  
        delete[] val;  
    }  
    Array& operator=(const Array& rhs); // оператор присваивания копированием  
    Array& operator=(Array&& rhs); // оператор присваивания перемещением  
    double& operator[](size_t i) {  
        return val[i];  
    }  
    const double& operator[](size_t i) const {
```

```
        return val[i];
    }

protected:
    std::size_t len {0}; // инициализация поля
    double* val {nullptr};
};
```

Конструктор вызывается для инициализации объекта (соответствующего типа) при его создании, а деструктор — для уничтожения объекта. Класс может иметь несколько конструкторов, но деструктор может иметь только один. Конструкторы в C++ не могут быть объявлены виртуальными, а деструкторы — могут, и обычно объявляются для всех полиморфных типов, чтобы гарантировать правильное уничтожение доступного по ссылке или указателю объекта независимо от того, какого типа ссылка или указатель. При наличии хотя бы у одного из базовых классов виртуального деструктора, деструктор класса потомка автоматически становится виртуальным.

Шаблоны

Шаблоны позволяют порождать функции и классы, параметризованные определённым типом или значением. Например, предыдущий класс мог бы реализовывать массив для любого типа данных:

```
template <typename T>
class Array {
    ...
    T& operator[](size_t i) {
        return val[i];
    }
protected:
    std::size_t len {0}; // инициализация поля
    T* val {nullptr};
};
```

Стандартная библиотека

Общая структура

Стандартная библиотека C++ включает в себя набор средств, которые должны быть доступны для любой реализации языка, чтобы обеспечить программистам удобное пользование языковыми средствами и создать базу для разработки как прикладных приложений самого широкого спектра, так и специализированных библиотек. Стандартная библиотека C++ включает в себя часть стандартной библиотеки C. Стандарт C++ содержит нормативную ссылку на стандарт C от 1990 года и не определяет самостоятельно те функции стандартной библиотеки, которые заимствуются из стандартной библиотеки C.

Доступ к возможностям стандартной библиотеки C++ обеспечивается с помощью включения в программу (посредством директивы `#include`) соответствующих стандартных заголовочных файлов. Всего в стандарте C++11 определено 79 таких файлов. Средства стандартной библиотеки объявляются как входящие в пространство имён `std`. Заголовочные файлы, имена которых соответствуют шаблону «сХ», где Х — имя заголовочного файла стандартной библиотеки C без расширения (`cstdlib`, `cstring`, `cstdio` и пр.), содержат объявления, соответствующие данной части стандартной библиотеки C. Стандартные функции библиотеки C также находятся в пространстве имён `std`.

Состав

Стандартная библиотека включает в себя следующие разделы:

- *Поддержка языка.* Включает средства, которые необходимы для работы программ, а также сведения об особенностях реализации. Выделение памяти, RTTI, базовые исключения, пределы значений для числовых типов данных, базовые средства взаимодействия со средой, такие как системные часы, обработка сигналов UNIX, завершение программы.
- *Стандартные контейнеры.* В стандартную библиотеку входят шаблоны для следующих контейнеров: динамический массив(vector), статический массив(array), одно- и двунаправленные списки(list, forward_list), стек(stack), дек(deque), ассоциативные массивы(map, multimap), множества(set, multiset), очередь с приоритетом(priority_queue).
- *Основные утилиты.* В этот раздел входит описание основных базовых элементов, применяемых в стандартной библиотеке, распределителей памяти и поддержка времени и даты в стиле C.
- *Итераторы.* Обеспечивают шаблоны итераторов, с помощью которых в стандартной библиотеке реализуется стандартный механизм группового применения алгоритмов обработки данных к элементам контейнеров.
- *Алгоритмы.* Шаблоны для описания операций обработки, которые с помощью механизмов стандартной библиотеки могут применяться к любой последовательности элементов, в том числе к элементам в контейнерах. Также в этот раздел входят описания функций bsearch() и qsort() из стандартной библиотеки C.
- *Строки.* Шаблоны строк в стиле C++. Также в этот раздел попадает часть библиотек для работы со строками и символами в стиле C.
- *Ввод-вывод.* Шаблоны и вспомогательные классы для потоков ввода-вывода общего вида, строкового ввода-вывода, манипуляторы (средства управления форматом потокового ввода-вывода в стиле C++).
- *Локализация.* Определения, используемые для поддержки национальных особенностей и форматов представления (дат, валют и т. д.) в стиле C++ и в стиле C.
- *Диагностика.* Определения ряда исключений и механизмов проверки утверждений во время выполнения (assert). Поддержка обработки ошибок в стиле C.
- *Числа.* Определения для работы с комплексными числами, математическими векторами, поддержка общих математических функций, генератор случайных чисел.

Контейнеры, строки, алгоритмы, итераторы и основные утилиты, за исключением заимствований из библиотеки C, собирательно называются STL (Standard Template Library — стандартная шаблонная библиотека). Изначально эта библиотека была отдельным продуктом и её аббревиатура расшифровывалась иначе, но потом она вошла в стандартную библиотеку C++ в качестве неотъемлемого элемента. В названии отражено то, что для реализации средств общего вида (контейнеров, строк, алгоритмов) использованы механизмы обобщённого программирования (шаблоны C++ — template). В работах Страуструпа подробно описываются причины, по которым был сделан именно такой выбор. Основными из них являются бóльшая универсальность выбранного решения (шаблонные контейнеры, в отличие от объектных, могут легко использоваться для не объектных типов и не требуют наличия общего предка у типов элементов) и его техническая эффективность (как правило, операции шаблонного контейнера не требуют вызовов виртуальных функций и могут легко встраиваться (inline), что в итоге даёт выигрыш в производительности).

Начиная со стандарта C++11 добавились следующие возможности:

- Добавлена библиотека `<regex>`, реализующая общепринятые механизмы поиска и подстановки с помощью регулярных выражений.

- Добавлена поддержка многопоточности.
- Атомарные операции
- unordered варианты ассоциативных массивов и множеств.
- Умные указатели, обеспечивающие автоматическое освобождение выделенной памяти.

Реализации

STL до включения в стандарт C++ была сторонней разработкой, вначале — фирмы HP, а затем SGI. Стандарт языка не называет её «STL», так как эта библиотека стала неотъемлемой частью языка, однако многие люди до сих пор используют это название, чтобы отличать её от остальной части стандартной библиотеки (поток ввода-вывода (iostream), подраздел C и другие).

Проект под названием STLport^[15], основанный на SGI STL, осуществляет постоянное обновление STL, Iostream и строковых классов. Некоторые другие проекты также занимаются разработкой частных применений стандартной библиотеки.

Отличия от языка C

Совместимость с языком C

Выбор именно C в качестве базы для создания нового языка программирования объясняется тем, что язык C:

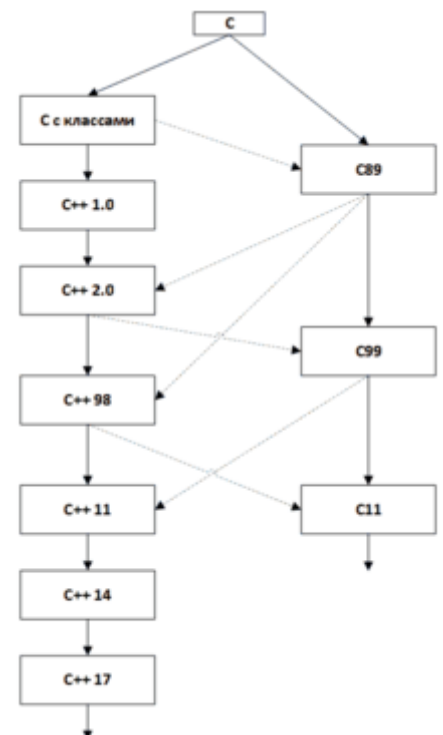
1. является многоцелевым, лаконичным и относительно низкоуровневым языком;
2. подходит для решения большинства системных задач;
3. исполняется везде и на всём;
4. стыкуется со средой программирования UNIX.

— Б. Страуструп. Язык программирования C++.
Раздел 1.6^[16]

Несмотря на ряд известных недостатков языка C, Страуструп пошёл на его использование в качестве основы, так как «в C есть свои проблемы, но их имел бы и разработанный с нуля язык, а проблемы C нам известны». Кроме того, это позволило быстро получить прототип компилятора (cfront), который лишь выполнял трансляцию добавленных синтаксических элементов в оригинальный язык C.

По мере разработки C++ в него были включены другие средства, которые перекрывали возможности конструкций C, в связи с чем неоднократно поднимался вопрос об отказе от совместимости языков путём удаления устаревших конструкций. Тем не менее, совместимость была сохранена из следующих соображений:

- сохранение действующего кода, написанного изначально на C и прямо перенесённого в C++;



Генеалогия и взаимовлияние версий C и C++ (по Б. Страуструп, «Язык программирования C++, краткий курс»)

- исключение необходимости переучивания программистов, ранее изучавших С (им требуется только изучить новые средства С++);
- исключение путаницы между языками при их совместном использовании («если два языка используются совместно, их различия должны быть или минимальными, или настолько большими, чтобы языки было невозможно перепутать»).

Новые возможности

Новые возможности С++ включают объявления в виде выражений, преобразования типов в виде функций, операторы `new` и `delete`, тип `bool`, ссылки, расширенное понятие константности, подставляемые функции, аргументы по умолчанию, переопределения, пространства имён, классы (включая и все связанные с классами возможности, такие как наследование, функции-члены, виртуальные функции, абстрактные классы и конструкторы), переопределения операторов, шаблоны, оператор `::`, обработку исключений, динамическую идентификацию и многое другое. Язык С++ также во многих случаях строже относится к проверке типов, чем С.

В С++ появились комментарии в виде двойной косой черты (`//`), которые были в предшественнике С — языке BCPL.

Некоторые особенности С++ позднее были перенесены в С, например, ключевые слова `const` и `inline`, объявления в циклах `for` и комментарии в стиле С++ (`//`). В более поздних реализациях С также были представлены возможности, которых нет в С++, например макросы `va_arg` и улучшенная работа с массивами-параметрами.

С++ не включает в себя С

Несмотря на то, что большая часть кода С будет справедлива и для С++, С++ не является надмножеством С и не включает его в себя. Существует и такой верный для С код, который неверен для С++. Это отличает его от Objective С, ещё одного усовершенствования С для ООП, как раз являющегося надмножеством С.

Существуют и другие различия. Например, С++ не разрешает вызывать функцию `main()` внутри программы, в то время как в С это действие правомерно. Кроме того, С++ более строг в некоторых вопросах; например, он не допускает неявное приведение типов между несвязанными типами указателей и не разрешает использовать функции, которые ещё не объявлены.

Более того, код, верный для обоих языков, может давать разные результаты в зависимости от того, компилятором какого языка он оттранслирован. Например, на большинстве платформ следующая программа печатает «С», если компилируется компилятором С, и «С++» — если компилятором С++. Так происходит из-за того, что символьные константы в С (например, `'a'`) имеют тип `int`, а в С++ — тип `char`, а размеры этих типов обычно различаются.

```
#include <stdio.h>

int main()
{
    printf("%s\n", (sizeof('a') == sizeof(char)) ? "C++" : "C");
    return 0;
}
```

Средства С, которых рекомендуется избегать

По замечанию Страуструпа, «чем лучше вы знаете С, тем труднее вам будет избежать программирования на С++ в стиле С, теряя при этом потенциальные преимущества С++». В связи с этим он даёт следующий набор рекомендаций для программистов на С, чтобы в полной мере воспользоваться преимуществами С++:

- Не использовать макроопределения `#define`. Для объявления констант применять `const`, групп констант (перечислений) — `enum`, для прямого включения функций — `inline`, для определения семейств функций или типов — `template`.
- Не использовать предварительные объявления переменных. Объявлять переменные в блоке, где они реально используются, всегда совмещая объявление с инициализацией.
- Отказаться от использования `malloc()`^[17] в пользу оператора `new`, от `realloc()`^[18] — в пользу типа `vector`. Более безопасным будет использование умных указателей, таких как `shared_ptr` и `unique_ptr`, доступных с одиннадцатой версии стандарта.
- Избегать бестиповых указателей, арифметики указателей, неявных приведений типов, объединений, за исключением, возможно, низкоуровневого кода. Использовать «новые» преобразования типов, как более точно выражающие действительные намерения программиста и более безопасные.
- Свести к минимуму использование массивов символов и строк в стиле С, заменив их на типы `string` и `vector` из STL. Вообще не стремиться создавать собственные реализации того, что уже имеется в стандартной библиотеке.

Дальнейшее развитие

Текущий стандарт языка ISO/IEC 14882:2017 был опубликован в декабре 2017 года. Неофициально его обозначают как С++17. Следующая версия стандарта, запланированная на 2020 год, имеет неофициальное обозначение С++20.

Общие направления развития С++

По мнению автора языка Бьёрна Страуструпа^{[19][20][21]}, говоря о дальнейшем развитии и перспективах языка, можно выделить следующее:

- В основном дальнейшее развитие языка будет идти по пути внесения дополнений в стандартную библиотеку. Одним из основных источников этих дополнений является известная библиотека boost.
- Изменения в ядре языка не должны приводить к снижению уже достигнутой эффективности С++. С точки зрения Страуструпа, предпочтительнее внесение в ядро нескольких серьёзных больших изменений, чем множества мелких правок.
- Базовыми направлениями развития С++ на ближайшее время является расширение возможностей и доработка средств обобщённого программирования, стандартизация механизмов параллельной обработки, а также доработка средств безопасного программирования, таких как различные проверки и безопасные преобразования типов, проверка условий и так далее.
- В целом С++ спроектирован и развивается как мультипарадигменный язык, впитывающий в себя различные методы и технологии программирования, но реализующий их на платформе, обеспечивающей высокую техническую эффективность. Поэтому в будущем не исключено добавление в язык средств функционального программирования, автоматической сборки мусора и других отсутствующих в нём сейчас

механизмов. Но в любом случае это будет делаться на имеющейся платформе высокоэффективного компилируемого языка.

- Хотя формально одним из принципов С++ остаётся сохранение совместимости с языком С, фактически группы по стандартизации этих языков не взаимодействуют, а вносимые ими изменения не только не коррелируют, но и нередко принципиально противоречат друг другу идеологически. Так, элементы, которые новые стандарты С добавляют в ядро, в стандарте С++ являются элементами стандартной библиотеки и в ядре вообще отсутствуют, например, динамические массивы, массивы с фиксированными границами, средства параллельной обработки. Как считает Страуструп, объединение разработки этих двух языков принесло бы большую пользу, но оно вряд ли возможно по политическим соображениям. Так что практическая совместимость между С и С++ постепенно будет утрачиваться.

Стандарт С++11: дополнения в ядре языка

- Явно определяемые константные функции и выражения **constexpr**.
- Универсальная инициализация.
- Конструкторы и операторы присваивания с семантикой переноса.
- Вывод типов.

Для применения в шаблонах, там, где затруднительно указать конкретный тип переменной, введены два новых механизма: переменные типа **auto** и описание **decltype**.

- Цикл по коллекции.

Вслед за многими современными языками в С++ введена конструкция «цикл по коллекции» вида **for (type &x : array) { ... }**. Здесь тело цикла выполняется для каждого элемента коллекции **array**, а **x** в каждой итерации будет ссылаться на очередной элемент коллекции. В качестве коллекции может выступать С-массив или любой контейнер стандартной библиотеки, для которого определены итераторы **begin** и **end**.

- Лямбда-выражения.

Добавлена возможность объявлять лямбда-выражения (безымянные функции, определяемые в точке применения), в том числе зависящие от внешних переменных (замыкания). Лямбда-выражения могут присваиваться переменным и использоваться везде, где требуется функция соответствующего типа, например, в алгоритмах стандартной библиотеки.

- Изменения в описании виртуальных методов.

Добавлен необязательный модификатор **override**, который употребляется в объявлении метода, замещающего виртуальный метод родительского класса. Описание замещения с **override** вызывает проверку на наличие в родительском классе замещаемого метода и на совпадение сигнатур методов. Добавлен также модификатор **final**, как и в Java, запрещающий дальнейшее замещение помеченного им метода. Также **final** может быть объявлен класс — в таком случае от него запрещено наследовать новые классы.

- Добавлена возможность описания вариативных шаблонов.
- Различные синтаксические дополнения.

Определено ключевое слово для константы — нулевого указателя: **nullptr**. Внесены изменения в семантику и, частично, синтаксис перечислений и объединений. Добавлена возможность создавать типобезопасные перечисления, с объединений снят ряд ограничений на структуру. От компилятора требуется правильный лексический разбор текста программы с несколькими закрывающимися угловыми скобками подряд (ранее последовательность «>>>» воспринималась однозначно как операция побитового сдвига вправо, поэтому в записи вложенных шаблонных конструкций требовалось обязательно разделять знаки «больше» пробелами или переводами строк).

Примеры программ

Пример № 1

Это пример программы Hello, world!, которая выводит сообщение, используя стандартную библиотеку, и завершается.

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello, world!" << endl;
    return 0;
}
```

Пример № 2

Современный C++ позволяет решать простым способом и более сложные задачи. Этот пример демонстрирует, кроме всего прочего, использование контейнеров стандартной библиотеки шаблонов (STL).

```
#include <iostream> // для использования std::cout
#include <vector> // содержит динамический массив
#include <map> // содержит тип данных словарь
#include <string>

int main()
{
    // импортируем все объявления в пространстве имён "std" в глобальное пространство имён.
    using namespace std;
    // Объявляем ассоциативный контейнер со строковыми ключами и данными в виде векторов строк
    map< string, vector<string> > items;

    // Добавим в этот ассоциативный контейнер пару человек и дадим им несколько предметов
    items["Anya"].push_back("scarf");
    items["Dmitry"].push_back("tickets");
    items["Anya"].push_back("puppy");

    // Переберём все объекты в контейнере
    for(const auto& person : items) {
        // person - это пара объектов: person.first - это его имя,
        // person.second - это список его предметов (вектор строк)
        cout << person.first << " is carrying " << person.second.size() << " items" << endl;
    }
}
```

В этом примере для простоты импортируются все имена из пространства имён `std`. В настоящей же программе так делать не рекомендуется, так как можно столкнуться с коллизией имён. Язык позволяет импортировать отдельные объекты:

```
#include <vector>

int main()
{
    using std::vector;
    vector<int> my_vector;
}
```

В C++ (как и в C), если выполнение программы доходит до конца функции `main()`, то это эквивалентно `return 0;`. Это неверно для любой другой функции кроме `main()`.

Критика

О критике C++ в целом

Чаще всего критики не противопоставляют C++ какой-либо другой конкретный язык, а утверждают, что отказ от использования единственного языка, имеющего многочисленные недостатки, в пользу декомпозиции проекта на подзадачи, решаемые на различных, наиболее подходящих для них, языках, делает разработку существенно менее трудоёмкой при одновременном повышении показателей качества программирования^{[22][23]}. По этой же причине критикуется сохранение совместимости с Си: если часть задачи требует низкоуровневых возможностей, разумнее выделить эту часть в отдельную подсистему и написать её на Си.

В свою очередь, сторонники C++ заявляют, что устранение технических и организационных проблем межъязыкового взаимодействия за счёт использование одного универсального языка вместо нескольких специализированных важнее, чем потери от несовершенства этого универсального языка, то есть сама широта набора возможностей C++ является оправданием недостатков каждой отдельной возможности; в том числе недостатки, унаследованные от Си, оправданы преимуществами совместимости (см. выше).

Таким образом, одни и те же свойства C++ — объём, сложность, эклектичность и отсутствие конкретной целевой ниши применения — рассматривается сторонниками как «главное достоинство», а критиками — как «главный недостаток».

Сравнение с альтернативными языками

Известно несколько исследований, в которых была сделана попытка более или менее объективно сравнить несколько языков программирования, одним из которых является C++. В частности:

- В научной статье «Haskell vs. Ada vs. C++ vs. Awk vs. ...» Пауля Худака и Марка Джонса^[24] даётся отчёт об исследовании ряда императивных и функциональных языков на решении модельной задачи быстрого прототипирования ГИС-системы военного назначения.
- В научной статье «DSL implementation in metaocaml, template haskell, and C++» четырёх авторов^[25] проводится методичное исследование применения C++ и двух функциональных языков в роли базовых инструментов для языково-ориентированного программирования методом порождающего программирования.

- В работе Лутца Прехельта^[26] рассмотрено семь языков (C, C++, Java, Perl, Python, Rexx и Tcl) в задаче написания простой программы преобразования телефонных номеров в слова по определённым правилам.
- В статье Дэвида Велера «Ada, C, C++, and Java vs. The Steelman»^[27] приведено сопоставление языков Ада, C++, Си, Java с документом «Steelman» — списком требований к языку для военных разработок встроенных систем, который был выработан комитетом по языку высокого уровня Министерства обороны США в 1978 году. Хотя этот документ сильно устарел и не учитывает многих существенных свойств современных языков, сравнение демонстрирует, что C++ по набору востребованных в отрасли возможностей не так уж сильно отличается от языков, которые можно считать его реальными конкурентами.

С++ и Ада

Язык Ада близок к C++ по набору возможностей и по сферам применения: это компилируемый структурный язык с Симула-подобным объектно-ориентированным дополнением (та же модель «Алгол с классами», что и в C++), статической типизацией, средствами обобщённого программирования, предназначенный для разработки крупных и сложных программных систем. В то же время он принципиально отличается по идеологии: в отличие от C++, Ада строилась на основе предварительно тщательно проработанных условий производителей сложного ПО с повышенными требованиями к надёжности, что наложило отпечаток на синтаксис и семантику языка.

Прямых сравнений эффективности кодирования на Аде и C++ немного. В упомянутой выше статье^[24] решение модельной задачи на Аде привело к получению кода примерно на 30 % меньшего по объёму (в строках), чем на C++. Сравнение свойств самих языков приводится во многих источниках, например, в статье Джима Роджерса на AdaHome^[28] содержится перечисление более 50 пунктов различий свойств этих языков, большая часть которых — в пользу Ады (больше возможностей, более гибкое поведение, меньше вероятность ошибок). Хотя многие утверждения сторонников Ады спорны, а часть из них явно устарела, в целом можно заключить:

- Синтаксис Ады гораздо строже, чем C++. Язык требует соблюдения дисциплины программирования, не поощряет «программистские трюки», стимулирует написание простого, логичного и легко понимаемого кода, удобного в сопровождении.
- В отличие от C++, Ада максимально типобезопасна. Развитая система типов позволяет, при соблюдении дисциплины их объявления и использования, максимально полно статически контролировать корректность использования данных и защищает от случайных ошибок. Автоматические преобразования типов сведены к минимуму.
- Указатели в Аде контролируются гораздо более строго, чем в C++, а адресная арифметика доступна только через отдельную системную библиотеку.
- Настраиваемые модули Ады по возможностям аналогичны шаблонам C++, но обеспечивают лучший контроль.
- Ада имеет встроенную в язык модульность и стандартизованную систему отдельной компиляции, тогда как C++ применяет включение текстовых файлов и внешние средства управления компиляцией и сборкой.
- Встроенная многозадачность Ады включает параллельные задачи и механизм их коммуникации (входы, рандеву, оператор `select`). В C++ всё это реализуется только на уровне библиотек.
- Ада строго стандартизована, за счёт чего обеспечивает лучшую переносимость.

В статье Стефена Цейгера из Rational Software Corporation^[29], утверждается, что в целом разработка на Аде обходится на 60 % дешевле, и приводит к получению кода, имеющего в 9 раз меньше дефектов, чем на Си. Хотя эти результаты не могут быть прямо перенесены на С++, но всё же представляют интерес с учётом того, что многие недостатки С++ унаследованы от Си.

С++ и Java

Java не может считаться в полной мере заменой С++, она создана как безопасный язык с низким порогом вхождения для разработки прикладных пользовательских приложений с высокими показателями портируемости^[30] и принципиально непригодна для некоторых типов приложений, которые разрабатываются на С++. Однако в пределах своей области Java составляет вполне реальную конкуренцию С++. В качестве преимуществ Java обычно называют:

- Безопасность: отсутствие поддержки указателей и адресной арифметики, автоматическое управление памятью со сборкой мусора, встроенные средства, защищающие от распространённых ошибок программ С++, таких как переполнение буфера или выход за границы массива.
- Наличие разработанной системы модулей и отдельной компиляции, значительно более быстрой и менее подверженной ошибкам, чем препроцессор и ручная сборка С++.
- Полная стандартизация и исполнение в виртуальной машине, развитое окружение, включающие библиотеки для графики, интерфейса пользователя, доступа к базам данных прочих типовых задач, как следствие — реальная многоплатформенность.
- Встроенная многопоточность.
- Объектная подсистема Java в значительно более высокой степени, чем С++, отвечает фундаментальному принципу ООП «*всё — объект*». Интерфейсы позволяют обеспечить большинство преимуществ множественного наследования, не вызывая его негативных эффектов.
- Рефлексия значительно более развита, чем в С++ и позволяет реально определять и изменять структуру объектов во время работы программы.

В то же время использование сборщика мусора и виртуальной машины создают труднопреодолимые ограничения. Программы на Java, как правило, медленнее, требуют значительно больше памяти, к тому же виртуальная машина изолирует программу от операционной системы, делая невозможным низкоуровневое программирование. Также можно заметить, что эмпирическое исследование^[26] не обнаружило существенной разницы в скорости разработки на С++ и на Java.

С++ и С

Оригинальный Си продолжает развиваться, на нём разрабатываются многие масштабные проекты: он является основным языком разработки операционных систем, на нём написаны игровые движки многих динамических игр и большое число прикладных приложений. Ряд специалистов утверждает, что замена Си на С++ не повышает эффективность разработки, но приводит к ненужному усложнению проекта, снижению надёжности и увеличению затрат на сопровождение. В частности:

- По мнению Линуса Торвальдса, «С++ провоцирует на написание ... значительного объёма кода, не имеющего принципиального значения с точки зрения функциональности программы»^[мнения 3].
- Поддержка ООП, шаблоны и STL не являются решающим преимуществом С++, так как всё, для чего они применяются, реализуемо и средствами Си. При этом устраняется раздувание кода, а некоторое усложнение, которое к тому же далеко не обязательно,

компенсируется большей гибкостью, более простым тестированием, лучшими показателями производительности.

- Автоматизация доступа к памяти в С++ увеличивает затраты памяти и замедляет работу программ.
- Использование исключений С++ вынуждает следовать RAII, приводит к росту исполняемых файлов, замедлению программ. Дополнительные трудности возникают в параллельных и распределённых программах. Показательно, что стандарт кодирования на С++ компании Google прямо запрещает использование исключений.^[31]
- Код на С++ сложнее для понимания и тестирования, его отладка затрудняется использованием сложных иерархий классов с наследованием поведения и шаблонов. К тому же в средах программирования на С++ больше ошибок, как в компиляторах, так и в библиотеках.
- Многие детали поведения кода стандартом С++ не специфицированы, что ухудшает переносимость и может являться причиной трудно обнаруживаемых ошибок.
- Квалифицированных программистов на Си существенно больше, чем на С++.

Нет убедительных данных о преимуществе С++ перед Си ни по производительности программистов, ни по свойствам программ. Хотя есть исследования^[32] утверждающие, что программисты на Си тратят 30 % — 40 % общего времени разработки (не считая отладки) на управление памятью, при сопоставлении общей производительности разработчиков^[24] Си и С++ оказываются близки.

В низкоуровневом программировании значительная часть новых возможностей С++ оказывается неприменимой из-за увеличения накладных расходов: виртуальные функции требуют динамического вычисления реального адреса (RVA), шаблоны приводят к раздуванию кода и ухудшению возможностей оптимизации, библиотека времени исполнения (RTL) очень велика, а отказ от неё лишает большинства возможностей С++ (хотя бы из-за недоступности операций `new/delete`). В результате программисту придётся ограничиться функционалом, унаследованным от Си, что делает бессмысленным применение С++:

... единственный способ иметь хороший, эффективный, низкоуровневый и портируемый С++ сводится к тому, чтобы ограничиться всеми теми вещами, которые элементарно доступны в Си. А ограничение проекта рамками Си будет означать, что люди его не выкинут, и что будет доступно множество программистов, действительно хорошо понимающих низкоуровневые особенности и не отказывающихся от них из-за идиотской ерунды про «объектные модели».

... когда эффективность является первостепенным требованием, «преимущества» С++ будут огромной ошибкой.

— Линус Торвальдс,^[33]

С++ и функциональные и скриптовые языки

В одном эксперименте^[24] скриптовые и функциональные языки, в частности, `Haskell`, показали 2-3 кратный выигрыш во времени программирования и объёме кода по сравнению с программами на С++. С другой стороны, программы на С++ оказались во столько же раз быстрее. Авторы признают, что полученные ими данные не составляют репрезентативной выборки и воздерживаются от категоричных выводов.

В другом эксперименте^[34] строгие функциональные языки (Standard ML, OCaml) показали общее ускорение разработки в 10 раз (в основном за счёт раннего выявления ошибок) при примерно равных показателях быстродействия (использовалось множество компиляторов в нескольких режимах).

В исследовании Лутца Прехельта^[26] по результатам обработки около 80 решений, написанных добровольцами, получены, в частности, следующие выводы:

- Perl, Python, Rexx, Tcl обеспечили скорость разработки вдвое больше, чем C, C++ и Java, причём полученный код был также вдвое короче.
- Программы на скриптовых языках потребляли примерно вдвое больше памяти, чем C/C++

Критика отдельных элементов и концепций

Контроль за поведением

Идеология языка смешивает «контроль за поведением» с «контролем за *эффективностью*», то есть предполагает, что обеспечение полного контроля программиста за всеми аспектами исполнения программы на довольно низком уровне является необходимым и достаточным условием достижения высокой эффективности кода. В действительности для сколько-нибудь крупных программ это неверно, так как их сложность настолько высока, что осознание её в полном объёме на низком уровне превышает возможности человека. Принцип «*не платишь за то, что не используешь*», заявленный как средство обеспечения эффективности, на практике приводит к отказу от параметрического полиморфизма и необходимости явного описания различного поведения для различных ситуаций под единым идентификатором (перегрузки функций), с ручной оптимизацией кода для каждого такого варианта, что вызывает значительное увеличение объёма и сложности кода, усугубляя проблемы управления сложностью. Возложение на программиста низкоуровневой оптимизации, которую качественный компилятор предметно-ориентированного языка способен выполнить заведомо более эффективно, приводит лишь к росту трудоёмкости программирования и снижению показателей понимаемости и тестируемости кода. Таким образом, принцип «не платить за то, что не используется» в действительности не даёт желаемых выгод в эффективности, но негативно сказывается на качестве.

Компонентное и объектно-ориентированное программирование

По мнению Алана Кэя, объектная модель «Алгол с классами», использованная в C++, уступает модели «всё — объект»^[35], используемой в Objective-C, по общему объёму возможностей, показателям повторного использования кода, понимаемости, модифицируемости и тестируемости.

Модель наследования C++ сложна, трудна в реализации и при этом провоцирует создание сложных иерархий с неестественными отношениями между классами (например, наследование вместо вложения). Результатом становится создание сильно зацепленных классов с нечётко разделённым функционалом. Например, в ^[36] приводится учебно-рекомендательный пример реализации класса «список» как подкласса от класса «элемент списка», который, в свою очередь, содержит функции доступа к другим элементам списка. Такое отношение типов является абсурдом с точки зрения математики и невоспроизводимо на более строгих языках. Идеология некоторых библиотек требует ручного приведения типов вверх и вниз по иерархии классов (`static_cast` и `dynamic_cast`),

что нарушает типобезопасность языка. Высокая вязкость решений на С++ может требовать повторной разработки значительных частей проекта при необходимости внесения минимальных изменений на поздних стадиях разработки. Яркий пример подобных проблем можно найти в^[22]

Как отмечает Ян Джойнер^[37], С++ ошибочно отождествляет инкапсуляцию (то есть помещение данных внутрь объектов и отделение реализации от интерфейса) и сокрытие реализации. Это усложняет доступ к данным класса и требует реализовывать его интерфейс практически исключительно через функции доступа (что, в свою очередь, увеличивает объём кода и усложняет его).

Совпадение типов в С++ определяется на уровне идентификаторов, а не сигнатур. Это затрудняет реализацию абстрактных механизмов работы с данными, так как делает невозможной подмену компонентов, основанную на совпадении их интерфейсной функциональности. В результате для включения в систему новой функциональности, реализованной на уровне библиотек, оказывается необходимо вручную модифицировать уже имеющийся код для адаптации его под новый модуль^[38]. Как отмечает Линус Торвальдс^[33], в С++ «код кажется абстрактным лишь до тех пор, пока не возникает необходимость его изменить».

Критика С++ с позиций только ООП (без сравнения методологий проектирования) с описанием вреда от влияния С++ на другие языки приведена в работе^[37].

Метапрограммирование

Порождающее метапрограммирование С++ основано на шаблонах и препроцессоре, оно трудоёмко и ограничено по возможностям. Система шаблонов С++ фактически является вариантом примитивного функционального языка программирования, исполняемого на этапе компиляции. Этот язык почти не пересекается с самим С++, из-за чего потенциал роста сложности абстракций оказывается ограниченным. Программы, использующие шаблоны С++, имеют крайне низкие показатели понимаемости и тестируемости, а само разворачивание шаблонов порождает неэффективный код, так как язык шаблонов не предоставляет никаких средств для оптимизации (см. также раздел #Вычислительная эффективность). Встраиваемые предметно-специфичные языки, реализуемые таким образом, всё равно требуют знания самого С++, что не обеспечивает полноценного разделения труда. Таким образом, возможности С++ по расширению возможностей самого С++ весьма ограничены.^{[39][40]}

Кроссплатформенность

Для написания портируемого кода на С++ требуется огромное мастерство и опыт, и «небрежные» коды на С++ с высокой вероятностью могут оказаться непортируемыми^[41]. По мнению Линуса Торвальдса, для обеспечения на С++ портируемости, аналогичной Си, программист должен ограничиться возможностями С++, унаследованными от Си^[33]. Стандарт содержит множество элементов, определённых как «implementation-defined» (например, размер указателей на методы классов в различных компиляторах варьируется в диапазоне от 4 до 20 байт^[42]), что ухудшает портируемость программ с их использованием.

Директивный характер стандартизации языка, неполная обратная совместимость и противоречивость требований разных версий стандарта приводят к проблемам в переносе программ между различными компиляторами и даже версиями одних и тех же компиляторов.

Отсутствие возможностей

Рефлексивное метапрограммирование

Интроспекция в C++ реализована отдельно от основной системы типов, что делает её практически бесполезной. Наибольшее, что можно получить — параметризацию поведения на заранее известном наборе вариантов. Это препятствует применению C++ в большинстве подходов к реализации Искусственного Интеллекта.

Функциональное программирование

Явная поддержка функционального программирования присутствует только в стандарте C++0x, ранее пробел устранялся библиотеками (Loki, Boost), использующими язык шаблонов, но их качество значительно уступает решениям, встроенным в функциональные языки,^[пояснения 1] как и качеству реализаций возможностей C++ (таких как ООП) посредством функциональных языков. Реализованные в C++ возможности ФП не дают возможности применения присущих функциональному программированию оптимизационных методик, а ограничивается вызовами функциональных библиотек и реализацией отдельных методов. Это практически не даёт преимуществ в проектировании программ (см. Соответствие Карри — Ховарда).

Избыточные и опасные возможности

Встроенные средства обхода ограничений

Язык содержит средства, позволяющие программисту нарушать заданную в конкретном случае дисциплину программирования. Например, модификатор `const` задаёт для объекта свойство неизменности состояния, но модификатор `mutable` предназначен *именно для* принудительного разрешения изменения состояния внутри константного объекта, то есть для нарушения ограничения константности. Более того, допускается динамически удалить атрибут `const` с константного объекта, превращая его в леводопустимый (L-value). Наличие в языке таких возможностей делает попытки формальной верификации кода бессмысленными, а использование ограничений для оптимизации невозможным.

Неконтролируемая макроподстановка

Средства макроподстановки Си (`#define`) являются сколь мощным, столь же опасным средством. Они сохранены в C++ несмотря на то, что для решения всех задач, для которых они были предусмотрены в Си, в C++ были предоставлены более строгие и специализированные средства — шаблоны, перегрузка функций, `inline`-функции, пространства имён, более развитая типизация, расширение применения модификатора `const`, и др. В унаследованных от Си стандартных библиотеках много потенциально опасных макросов.^[43] Шаблонное метапрограммирование также порой совмещается с использованием макроподстановки для обеспечения т. н. «синтаксического сахара».

Проблемы перегрузки

Принятые в C++ принципы перегрузки функций и операторов приводят к значительному дублированию кода. Перегрузка операторов, исходно предназначенная для введения так называемого «синтаксического сахара», в C++ поощряет бесконтрольное изменение поведения элементарных операций для различных типов. Это резко повышает риск ошибок, тем более что вводить новый синтаксис и изменять существующий (например, создавать новые операторы или менять приоритеты или ассоциативность) нельзя, хотя синтаксис стандартных операторов C++ адекватен семантике далеко не всех типов, которые может потребоваться ввести в программу.

Отдельные проблемы создаёт возможность лёгкой перегрузки операторов `new/delete`, способной породить крайне коварные и трудновывяемые ошибки. При этом некоторые интуитивно ожидаемые операции (подчистка динамических объектов в случае генерации исключений) в C++ не выполняются, а значительная часть перегруженных функций и операторов вызывается неявно (приведение типов, создание временных экземпляров классов и др.). В результате средства, изначально предназначенные для того, чтобы сделать программы более ясными и повысить удобство разработки и сопровождения, превращаются в ещё один источник неоправданного усложнения и снижения надёжности кода.

Вычислительная эффективность

Результирующий объём исполнимого кода

Использование шаблонов C++ представляет собой параметрический полиморфизм на уровне исходного кода, но при трансляции он превращается в ситуативный (ad hoc) полиморфизм (то есть перегрузку функций), что приводит к существенному увеличению объёма машинного кода в сравнении с языками, имеющими истинно полиморфную систему типов (потомками ML). Для снижения размера машинного кода пытаются автоматически обрабатывать исходный код до этапа раскрутки шаблонов^{[44][45]}. Другим решением могла бы быть стандартизованная ещё в 1998 году возможность экспорта шаблонов, но она доступна далеко не во всех компиляторах, так как её трудно реализовать^{[46][47][мнения 4]} и для импорта библиотек шаблонов C++ в языки с существенно отличной от C++ семантикой она всё равно была бы бесполезна. Сторонники C++ оспаривают масштабы раздувания кода как преувеличенные^[48], игнорируя даже тот факт, что в Си параметрический полиморфизм транслируется непосредственно, то есть без дублирования тел функций вообще. При этом сторонники C++ считают, что параметрический полиморфизм в Си опасен — то есть более опасен, чем переход от Си к C++ (противники C++ утверждают обратное — см. выше).

Потенциал оптимизации

Из-за слабой системы типов и изобилия побочных эффектов становится крайне затруднительным эквивалентное преобразование программ, а значит и встраивание в компилятор многих оптимизирующих алгоритмов, таких как автоматическое распараллеливание программ, удаление общих подвыражений, λ-подъём, вызовы процедур с передачей продолжений, суперкомпиляция и др. В результате реальная эффективность программ на C++ ограничивается имеющейся квалификацией программистов и вложенными в конкретный проект усилиями, и «небрежная» реализация может существенно уступать по эффективности «небрежным» реализациям на языках более высокого уровня, что подтверждается сравнительными испытаниями языков^[34]. Это является существенным препятствием против применения C++ в индустрии data mining.

Эффективное управление памятью

Обязанность по эффективному управлению памятью ложится на плечи разработчика и зависит от навыков разработчика. Для автоматического управления памятью в C++ традиционно используются т. н. «умные указатели», ручное же управление памятью снижает эффективность самих программистов (см. раздел Результативность). Многочисленные реализации сборки мусора, таких как статический вывод регионов не применимы для C++ программ (точнее, это требует реализации поверх языка C++ интерпретатора нового языка, сильно отличающегося от C++ как большинством объективных свойств, так и общей идеологией) по причине необходимости прямого доступа к AST.

Результативность

Соотнесение факторов результативности с затратами на разработку, а также общая культивируемая в сообществе программистов дисциплина и культура программирования важны для заказчиков, выбирающих язык (и, соответственно, предпочитающих этот язык разработчиков) для реализации задуманных ими проектов, а также для людей, начинающих изучать программирование, особенно с намерением программировать для собственных нужд.

Качество и культура программирования

Принцип C++ «не навязывать „хороший“ стиль программирования» противоречит промышленному подходу к программированию, в котором ведущую роль играют качество программного обеспечения и возможность сопровождения кода не только автором, и для которого предпочтительны языки, сводящие к минимуму влияние человеческого фактора, то есть как раз «навязывающие „хороший“ стиль программирования», хотя такие языки и могут иметь более высокий порог вхождения.

Существует мнение, что предпочтение использования C++ (при возможности выбора альтернативных языков) отрицательно характеризует профессиональные качества программиста. В частности, Линус Торвальдс говорит, что использует положительное мнение кандидатов о C++ в качестве критерия отсева^[мнения 3].

C++ — кошмарный язык. Его делает ещё более кошмарным тот факт, что множество недостаточно грамотных программистов используют его... Откровенно говоря, даже если нет **никаких** причин для выбора Си, кроме того чтобы держать C++-программистов подальше — то одно это уже будет достаточно веским основанием для использования Си. ...Я пришёл к выводу, что **действительно** предпочту выгнать любого, кто предпочтёт вести разработку проекта на C++, нежели на Си, чтобы этот человек не загубил проект, в который я вовлечён.

— Линус Торвальдс,^[33]

Исправление исправного

Непрерывная эволюция языка побуждает (а порой вынуждает) программистов раз за разом изменять уже отлаженный код — это не только удорожает разработку, но и несёт риск внедрения в отлаженный код новых ошибок. В частности, хотя изначально обратная совместимость с Си была одним из базовых принципов C++, с 1999 года Си перестал быть подмножеством C++, так что отлаженный код на Си уже не может использоваться в проекте на C++ без изменений.

Сложность ради самой сложности

C++ определяется его апологетами как «мощнейший» *именно потому*, что он изобилует опасными взаимно-противоречивыми возможностями. По мнению Эрика Реймонда, это делает язык сам по себе почвой для личного самоутверждения программистов, превращения процесса разработки в самоцель:

Программисты — это зачастую яркие люди, которые гордятся ... своей способностью справляться со сложностями и ловко обращаться с абстракциями. Часто они состязаются друг с другом, пытаясь выяснить, кто может создать «самые замысловатые и красивые

сложности». ... соперники полагают, что должны соревноваться с чужими «украшательствами» путём добавления собственных. Довольно скоро «массивная опухоль» становится индустриальным стандартом, и все используют большие, переполненные ошибками программы, которые не способны удовлетворить даже их создателей.

...

... такой подход может обернуться неприятностями, если программисты реализуют простые вещи сложными способами, просто потому что им известны эти способы и они умеют ими пользоваться.

— Эрик Реймонд в ^[49]

Саботаж

Отмечены случаи, когда нерадивые программисты, пользуясь сильной контекстной зависимостью C++ и отсутствием возможности отслеживания макроопределений компилятором тормозили разработку проекта, написав одну-две лишних, корректных с точки зрения компилятора, строки кода, но внедрив за их счёт труднообнаружимую спонтанно проявляющуюся ошибку. Например:

```
#define if(a) if(rand())
```

```
#define j i
```

В языках с доказанной корректностью, даже с развитыми макросредствами, нанести урон подобным образом невозможно.

Ненадёжность продукта

Неоправданное обилие побочных эффектов в сочетании с отсутствием контроля со стороны системы времени исполнения языка и слабой системой типов делает программы на C++ подверженными непредсказуемым фатальным сбоям (общеизвестные падения с сообщениями типа «Access violation», «Pure virtual function call» или «Программа выполнила недопустимую операцию и будет закрыта»), что исключает применение C++ при высоких требованиях к отказоустойчивости. Кроме того, это увеличивает длительность самого процесса разработки^[34].

Менеджмент проектов

Перечисленные выше факторы делают сложность менеджмента проектов на C++ одной из самых высоких в индустрии разработки ПО.

Джеймс Коггинс, в течение четырёх лет ведущий колонку в *The C++ Report*, даёт такое объяснение:

— Проблема в том, что программисты, работающие в ООП, экспериментировали с кровосмесительными приложениями и были нацелены на низкий уровень абстракции. Например, они строили такие классы как «связанный список», вместо «интерфейс

пользователя», или «луч радиации», или «модель из конечных элементов». К несчастью, строгая проверка типов, которая помогает программистам С++ избегать ошибок, одновременно затрудняет построение больших объектов из маленьких.

— Ф. Брукс, Мифический человеко-месяц

Влияние и альтернативы

Единственным прямым потомком С++ является язык D, задуманный как переработка С++ для устранения наиболее очевидных его проблем. Авторы отказались от совместимости с Си, сохранив синтаксис и многие базовые принципы С++ и введя в язык возможности, характерные для новых языков. В D нет препроцессора, заголовочных файлов, множественного наследования, но есть система модулей, интерфейсы, ассоциативные массивы, поддержка unicode в строках, сборка мусора (при сохранении возможности ручного управления памятью) встроенная многопоточность, вывод типов, явное объявление чистых функций и неизменяемых значений. Использование D весьма ограничено, считать его реальным конкурентом С++ нельзя.

Старейшим конкурентом С++ в задачах низкого уровня является Objective-C, также построенный по принципу объединения Си с объектной моделью, только объектная модель унаследована от Smalltalk. Objective-C, как и его потомок Swift, широко используется для разработки ПО под macOS и iOS.

Одной из первых альтернатив С++ в прикладном программировании стал язык Java. Его часто ошибочно считают прямым потомком С++; в действительности семантика Java унаследована от языка Модуля-2, и основы семантики С++ в Java не прослеживаются. Учитывая это, а также генеалогию языков (Модуля-2 является потомком Симулы, как и С++, но им не является Си), Java правильнее называть «*троюродным племянником*» С++, нежели «*наследником*». То же можно сказать о языке C#.

Попыткой совмещения безопасности и скорости разработки, характерных для Java и C#, с возможностями С++ явился диалект Managed C++ (впоследствии — C++/CLI). Он разработан Microsoft в основном для переноса существующих проектов на С++ под платформу Microsoft.NET. Программы выполняются под управлением CLR и могут использовать весь массив библиотек .NET, но при этом накладывается ряд ограничений на использование возможностей С++, что фактически сводит С++ к C#. Данный диалект не получил широкого признания.

Альтернативный путь развития языка Си — совмещение его не с объектно-ориентированным, а с аппликативным программированием, то есть улучшение абстракции, строгости и модульности низкоуровневых программ посредством обеспечения предсказуемости поведения и ссылочной прозрачности. Примерами работ в этом русле служат языки BitC, Cyclone и Limbo. Хотя есть и успешные попытки применения ФП в задачах реального времени без интеграции со средствами Си^{[50][51][52]}, всё же на данный момент (2013 г.) в низкоуровневой разработке применение в той или иной мере средств Си имеет лучшее соотношение трудоёмкости с результативностью. Много усилий было приложено разработчиками Python и Lua для обеспечения использования этих языков программистами на С++, так что из всех языков, достаточно тесно связанных с ФП, именно они чаще всего отмечаются в совместном использовании с С++ в одном проекте. Наиболее значимыми точками соприкосновения С++ с ФП можно считать привязки разработанных на С++ библиотек wxWidgets и Qt с характерной для С++ идеологией к языкам Lisp, Haskell и Python (в большинстве случаев привязки к функциональным языкам делают для библиотек, написанных на Си или на других функциональных языках).

Ещё одним языком, рассматриваемым как конкурент C++, стал Nemerle, являющийся результатом попытки совместить модель типизации Хиндли-Милнера и макроподмножество Common Lisp с языком C#.^[53] В том же русле находится созданный Microsoft язык F# — диалект ML, адаптированный для среды .NET.

Попыткой создать промышленную замену C/C++ стал разработанный в корпорации Google в 2009 году язык программирования Go. Авторы языка прямо указывают, что мотивом для его создания были недостатки процесса разработки, вызванные особенностями языков Си и C++.^[54] Go — компактный, несложный по структуре императивный язык с Си-подобным синтаксисом, без препроцессора, со статической типизацией, строгим контролем типов, системой пакетов, автоматическим управлением памятью, некоторыми функциональными чертами, экономно построенной ООП-подсистемой без поддержки наследования реализации, но с интерфейсами и утиной типизацией, встроенной многопоточностью, основанной на сопрограммах и каналах (a-la Оссам). Язык позиционируется как альтернатива C++, то есть, в первую очередь, средство групповой разработки высокоэффективных вычислительных систем большой сложности, в том числе распределённых, допускающее, при необходимости, низкоуровневое программирование.



В одной экологической нише с C/C++ находится разработанный в 2010 году и поддерживаемый корпорацией Mozilla язык Rust, ориентированный на безопасное управление памятью без использования сборщика мусора. В частности о планах частичной замены C/C++ на Rust объявила в 2019 компания Microsoft.^[55]

См. также



- Совместимость Си и C++
- wxWidgets
- Qt
- Тернарная условная операция в C++
- Стандартная библиотека языка Си
- Стандартная библиотека языка C++
- C
- C#
- AspectC++

Примечания

1. ISO/IEC 14882:2020 Programming languages — C++ (<https://www.iso.org/standard/79358.html>) — 2020.
2. Губина Г. Г. Компьютерный английский. Ч. I. Computer English. Part I. Учебное пособие (<http://books.google.ru/books?id=RcrhAwAAQBAJ&pg=PA385>). — С. 385.
3. Bjarne Stroustrup's FAQ (http://www.stroustrup.com/bs_faq.html#name) (англ.). Bjarne Stroustrup (October 1, 2017). — «The name C++ (pronounced "see plus plus")». Дата обращения: 4 декабря 2017.
4. Шилдт, 1998.
5. Страуструп, 1999, 2.1. Что такое C++?, с. 57.
6. Стэнли Липпман, Pure C++: Hello, C++/CLI (<http://msdn.microsoft.com/en-us/magazine/cc163681.aspx>) (англ.)
7. Страуструп, 1999, 1.4. Исторические замечания, с. 46.
8. C++ — Standards (<http://www.open-std.org/jtc1/sc22/wg21/docs/standards>)

9. *Bjarne Stroustrup. C++ Glossary* (<http://www.research.att.com/~bs/glossary.html>) (недоступная ссылка). Дата обращения: 8 июня 2007. Архивировано (<https://web.archive.org/web/20110501150638/http://www2.research.att.com/~bs/glossary.html>) 1 мая 2011 года.
10. Where do I find the current C or C++ standard documents?
<http://stackoverflow.com/questions/81656/where-do-i-find-the-current-c-or-c-standard-documents>
11. See a list at <http://en.cppreference.com/w/cpp/experimental> Visited 5 January 2015.
12. Яндекс организует рабочую группу по стандартизации языка C++ (<http://www.iksmedia.ru/news/5298228-Yandeks-sozdaet-rabochuyu-gruppu.html>). ИКС Медиа. Дата обращения: 29 августа 2018.
13. Страуструп, Дизайн и эволюция C++, 2007.
14. ISO/IEC 14882:1998, раздел 6.4, пункт 4: «The value of a *condition* that is an initialized declaration in a statement other than a `switch` statement is the value of the declared variable implicitly converted to `bool` ... The value of a *condition* that is an expression is the value of the expression, implicitly converted to `bool` for statements other than `switch`; if that conversion is ill-formed, the program is ill-formed».
15. STLport: Welcome! (<http://www.stlport.org/>)
16. Страуструп, 1999, 1.6.
17. `std::vector` — [cppreference.com](http://en.cppreference.com/w/cpp/container/vector) (<http://en.cppreference.com/w/cpp/container/vector>)
18. `std::realloc` — [cppreference.com](http://en.cppreference.com/w/cpp/memory/c/realloc) (<http://en.cppreference.com/w/cpp/memory/c/realloc>)
19. Интервью Б. Страуструпа [LinuxWorld.com](http://www.codenet.ru/progr/cpp/stroustrup.php) (<http://www.codenet.ru/progr/cpp/stroustrup.php>)
20. Интервью Б. Страуструпа журналу «Системный администратор» (<http://samag.ru/archive/article/1034>)
21. CNews: Эксклюзивное интервью с создателем языка программирования C++ (<http://www.cnews.ru/reviews/index.shtml?2010%2F09%2F29%2F410282>) (недоступная ссылка). Дата обращения: 1 ноября 2019. Архивировано (<https://web.archive.org/web/20150317122705/http://www.cnews.ru/reviews/index.shtml?2010%2F09%2F29%2F410282>) 17 марта 2015 года.
22. *Martin Ward*. (<http://www.cse.dmu.ac.uk/~mward/martin/papers/middle-out-t.pdf>)  *Language Oriented Programming*. — Computer Science Department, Science Labs, 1994.
23. *Paul Hudak*. *Modular Domain Specific Languages and Tools*. — Department of Computer Science, Yale University.
24. *An Experiment in Software Prototyping Productivity*. Paul Hudak, Mark P. Jones. Yale University, Department of Computer Science, New Haven, CT 06518. July 4, 1994.
25. *K. Czarnecki, J. O'Donnell, J. Striegnitz, W. Taha*. *DSL implementation in metaocaml, template haskell, and C++* (https://www.researchgate.net/publication/221025927_DSL_implementation_in_MetaOCaml_template_Haskell_and_C). — University of Waterloo, University of Glasgow, Research Centre Julich, Rice University, 2004..
Цитата: *C++ Template Metaprogramming suffers from a number of limitations, including portability problems due to compiler limitations (although this has significantly improved in the last few years), lack of debugging support or IO during template instantiation, long compilation times, long and incomprehensible errors, poor readability of the code, and poor error reporting.*
26. *Lutz Prechelt (Universität Karlsruhe)*. *An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program* (http://page.mi.fu-berlin.de/prechelt/Biblio/jccpprt_computer2000.pdf)  (англ.) (pdf). *Freie Universität Berlin* (14 March 2000). Дата обращения: 20 ноября 2019.
27. «Ada, C, C++, and Java vs. The Steelman» David A. Wheeler July/August 1997 (<https://dwheeler.com/steelman/steeltab.htm>) (недоступная ссылка). Дата обращения: 23 марта 2019. Архивировано (<https://web.archive.org/web/20190323164511/https://dwheeler.com/steelman/steeltab.htm>) 23 марта 2019 года.
28. *Comparison of Ada and C++ Features (en)* (http://www.adahome.com/articles/9703/ada_vs_cpp.html)

29. Stephen Zeigler, Comparing Development Costs of C and Ada. (http://www.adaic.com/whyada/ada-vs-c/cada_art.html) Архивировано (https://web.archive.org/web/20070404120024/http://www.adaic.com/whyada/ada-vs-c/cada_art.html) 4 апреля 2007 года.
30. 1.2 Design Goals of the Java™ Programming Language (<http://www.oracle.com/technetwork/java/intro-141325.html>). The Java Language Environment (англ.). Oracle (1 January 1999). Дата обращения: 14 января 2013.
31. Google C++ Style Guide. Exceptions. (<https://google.github.io/styleguide/cppguide.html#Exceptions>)
32. Boehm H. Advantages and Disadvantages of Conservative Garbage Collection (http://www.hpl.hp.com/personal/Hans_Boehm/gc/issues.html). Архивировано (https://web.archive.org/web/20130724030259/http://www.hpl.hp.com/personal/Hans_Boehm/gc/issues.html) 24 июля 2013 года. (ссылка из *Реймонд, Эрик. Искусство программирования для Unix. — 2005. — С. 357. — 544 с. — ISBN 5-8459-0791-8.*)
33. Открытая переписка [gmane.comp.version-control.git](http://thread.gmane.org/gmane.comp.version-control.git/57643/focus=57918) от 06.09.2007 (<http://thread.gmane.org/gmane.comp.version-control.git/57643/focus=57918>) (недоступная ссылка). Дата обращения: 5 августа 2013. Архивировано (<https://web.archive.org/web/20131209133259/http://thread.gmane.org/gmane.comp.version-control.git/57643/focus=57918>) 9 декабря 2013 года.
34. **Ray Tracer Language Comparison** (бенчмарк языков программирования — ffconsultancy.com/languages/ray_tracer/)
35. Alan Kay's Definition Of Object Oriented Programming (<http://c2.com/cgi/wiki?AlanKaysDefinitionOfObjectOriented>) (недоступная ссылка). Дата обращения: 5 августа 2013. Архивировано (<https://web.archive.org/web/20130824235515/http://c2.com/cgi/wiki?AlanKaysDefinitionOfObjectOriented>) 24 августа 2013 года.
36. Шилдт, Теория и практика C++, 1996, с. 64—67.
37. Ian Joyner. A Critique of C++ and Programming and Language Trends of the 1990s - 3rd Edition (<http://www.quinn.echidna.id.au/quinn/C++-Critique-3ed.pdf>)  — копирайт и список изданий (<http://www.emu.edu.tr/aelci/Courses/D-318/D-318-Files/cppcrit/index008.htm>).
38. Paulson, Lawrence C. ML for the Working Programmer, 2nd edition. — University of Cambridge: Cambridge University Press, 1996. — ISBN 0-521-57050-6 (hardback), 0-521-56543-X (paperback)., с.271-285
39. Walid Taha. Domain-Specific Languages (<http://www.cs.rice.edu/~taha/publications/conference/icc08.pdf>) . — Department of Computer Science, Rice University. Архивировано (<https://web.archive.org/web/20131024072658/http://www.cs.rice.edu/~taha/publications/conference/icc08.pdf>)  24 октября 2013 года.
40. Lugovsky V.S. Using a hierarchy of Domain Specific Languages in complex software systems design (<http://arxiv.org/abs/cs/0409016v1>). — 2008.
41. Андрей Карнов. 20 ловушек переноса Си++ - кода на 64-битную платформу (<http://www.rsdn.ru/?article/cpp/XXtraps64bit.xml>). — RSDN Magazine #1-2007, 2007.
42. Don Clugston, CSG Solar Pty Ltd (Перевод: Денис Буличенко). Указатели на функции-члены и реализация самых быстрых делегатов на C++ (<http://www.rsdn.ru/?article/cpp/FastDelegate/FastDelegate.xml>). — RSDN Magazine #6-2004.
43. Страуструп, Программирование: принципы и практика использования C++, 2001
44. Dave Gottner. Templates Without Code Bloat (<http://www.ddj.com/cpp/184403053>). — Dr. Dobb's Journal, январь 1995.
45. Adrian Stone. Minimizing Code Bloat: Redundant Template Instantiation (<http://gameangst.com/?p=246>) (недоступная ссылка). Game Angst (22 сентября 2009). Дата обращения: 19 января 2010. Архивировано (<https://web.archive.org/web/20111208031943/http://gameangst.com/?p=246>) 8 декабря 2011 года.
46. Herb Sutter. C++ Conformance Roundup (<http://www.ddj.com/cpp/184401381>). — Dr. Dobb's Journal, январь 2001.

47. Are there any compilers that implement all of this? (<http://www.comeaucomputing.com/csc/faq.html#C8>) (недоступная ссылка). *comp.std.c++ frequently asked questions / The C++ language*. Comeau Computing (10 декабря 2008). Дата обращения: 19 января 2010. Архивировано (<https://web.archive.org/web/20090430065535/http://www.comeaucomputing.com/csc/faq.html#C8>) 30 апреля 2009 года.
48. *Scott Meyers*. Code Bloat due to Templates (https://groups.google.com/group/comp.lang.c++.moderated/browse_thread/thread/2b00649a935997f5/5feec243078c5fd8). *comp.lang.c++.moderated*. Usenet (16 мая 2002). Дата обращения: 19 января 2010.
49. *Эрик Реймонд*. Искусство программирования для Unix = The Art of Unix. — Вильямс, 2005. — ISBN 5-8459-0791-8.
50. *Zhanyong Wan and Paul Hudak*. Event-Driven FRP (<http://www.cs.rice.edu/~taha/publications/conference/padl02.pdf>)  // Department of Computer Science, Yale University. Архивировано (<https://web.archive.org/web/20110816234039/http://www.cs.rice.edu/~taha/publications/conference/padl02.pdf>)  16 августа 2011 года.
51. Walid Taha (<http://www.cs.rice.edu/~taha/publications/>) (англ.) (недоступная ссылка). *School of Engineering Rice University*. Дата обращения: 20 ноября 2019. Архивировано (<https://www.webcitation.org/6lqLu1AP0?url=http://www.cs.rice.edu/~taha/publications/>) 13 августа 2013 года.
52. MLKit (http://www.it-c.dk/research/mlkit/index.php/Main_Page) (недоступная ссылка). Дата обращения: 30 июля 2013. Архивировано (https://web.archive.org/web/20130917074849/http://www.it-c.dk/research/mlkit/index.php/Main_Page) 17 сентября 2013 года.
53. *Чистяков Влад aka VladD2*. Синтаксический сахар или C++ vs. Nemerle :) (<http://rstdn.ru/?article/philosophy/SyntacticSugar.xml>) // RSDN Magazine #1-2006. — 24.05.2006.
54. Go at Google: Language Design in the Service of Software Engineering (<https://talks.golang.org/2012/splash.article>). *talks.golang.org*. Дата обращения: 19 сентября 2017.
55. *Catalin Cimpanu*. Microsoft to explore using Rust (<https://www.zdnet.com/article/microsoft-to-explore-using-rust/>) (англ.). ZDNet (17 June 2019). Дата обращения: 26 сентября 2019.

Мнения

1. *Programming Language Popularity* (<http://www.langpop.com/>) (недоступная ссылка) (2009). Дата обращения: 16 января 2009. Архивировано (<https://web.archive.org/web/20090116080326/http://www.langpop.com/>) 16 января 2009 года.
2. *TIOBE Programming Community Index* (<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>) (недоступная ссылка) (2009). Дата обращения: 6 мая 2009. Архивировано (<https://web.archive.org/web/20090504181627/http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>) 4 мая 2009 года.
3. Открытая переписка *gmane.comp.version-control.git* от 06.09.2007 (<http://thread.gmane.org/gmane.comp.version-control.git/57643/focus=57918>) (недоступная ссылка). Дата обращения: 5 августа 2013. Архивировано (<https://web.archive.org/web/20131209133259/http://thread.gmane.org/gmane.comp.version-control.git/57643/focus=57918>) 9 декабря 2013 года.
4. *vanDooren*. C++ keyword of the day: export (<http://msmvps.com/blogs/vandooren/archive/2008/09/24/c-keyword-of-the-day-export.aspx>) (недоступная ссылка). *Blogs@MSMVPs* (24 сентября 2008). — «The export keyword is a bit like the Higgs boson of C++. Theoretically it exists, it is described by the standard, and noone has seen it in the wild. ... There is 1 C++ compiler front-end in the world which actually supports it». Дата обращения: 19 января 2010. Архивировано (<https://web.archive.org/web/20090506053736/http://msmvps.com/blogs/vandooren/archive/2008/09/24/c-keyword-of-the-day-export.aspx>) 6 мая 2009 года.

Пояснения

1. Например, *Boost . Lambda* позиционируется как лямбда-функция, однако C++ не воплощает лямбда-исчисление Чёрча целиком; имитация комбинаторов посредством языка шаблонов слишком затруднено, чтобы ожидать их использование на практике;

продолжения, унаследованные от Си, считаются неидеоматичными и опасными, а их реализация посредством языка шаблонов невозможна; кроме того, применение λ-подъёма для оптимизации С++ невозможно,— так что фактически Boost . Lambda — это просто анонимная функция, а не объект лямбда-исчисления.

Литература

- Бьёрн Страуструп. Язык программирования С++ = The C++ Programming Language / Пер. с англ. — 3-е изд. — СПб.; М.: Невский диалект — Бином, 1999. — 991 с. — 3000 экз. — ISBN 5-7940-0031-7 (Невский диалект), ISBN 5-7989-0127-0 (Бином), ISBN 0-201-88954-4 (англ.).
- Бьёрн Страуструп. Язык программирования С++. Специальное издание = The C++ programming language. Special edition. — М.: Бином-Пресс, 2007. — 1104 с. — ISBN 5-7989-0223-4.
- Бьёрн Страуструп. Программирование: принципы и практика использования С++, исправленное издание = Programming: Principles and Practice Using C++. — М.: Вильямс, 2011. — С. 1248. — ISBN 978-5-8459-1705-8.
- Бьёрн Страуструп. Дизайн и эволюция С++ = The Design and Evolution of C++. — СПб.: Питер, 2007. — 445 с. — ISBN 5-469-01217-4.
- Бьёрн Страуструп. Язык программирования С++. Краткий курс. — 2019. — 320 с. — ISBN 978-5-907144-12-5.
- Бьёрн Страуструп. Программирование: принципы и практика с использованием С++. — 2016. — 1328 с. — ISBN 978-5-8459-1949-6.
- Сиддхартха Рао. Освой самостоятельно С++ за 21 день, 7-е издание (С++11) = Sams Teach Yourself C++ in One Hour a Day, 7th Edition. — М.: «Вильямс», 2013. — 688 с. — ISBN 978-5-8459-1825-3.
- Стефенс Д. Р. С++. Сборник рецептов. — КУДИЦ-ПРЕСС, 2007. — 624 с. — ISBN 5-91136-030-6.
- Стивен Прама. Язык программирования С++ (С++11). Лекции и упражнения = C++ Primer Plus, 6th Edition (Developer's Library). — 6-е изд. — М.: Вильямс, 2012. — 1248 с. — ISBN 978-5-8459-1778-2.
- Стивен Прама. Язык программирования С. Лекции и упражнения. — М.: Вильямс, 2015. — 928 с. — ISBN 978-5-8459-1950-2.
- Айвор Хортон. Visual C++ 2010: полный курс = Ivor Horton's Beginning Visual C++ 2010. — М.: Диалектика, 2010. — С. 1216. — ISBN 978-5-8459-1698-3.
- Герберт Шилдт. Полный справочник по С++ = C++: The Complete Reference. — 4-е изд. — М.: Вильямс, 2011. — С. 800. — ISBN 978-5-8459-0489-8.
- Герберт Шилдт. Теория и практика С++ = Shildt's Expert C++. — СПб.: BHV — Санкт-Петербург, 1996. — ISBN 0-07-882209-2, 5-7791-0029-2.
- P.J. Plauger. Programming Language Guessing Games - If C++ is the Answer, what's the question? // Dr. Dobbs's Journal. — Октябрь, 1993.
- The Unix-Haters Handbook (<http://web.mit.edu/~simsong/www/ugh.pdf>)  (неопр.). — International Data Group, 1994.
- Ian Joyner. A Critique of C++ and Programming and Language Trends of the 1990s - 3rd Edition (<http://www.quinn.echidna.id.au/quinn/C++-Critique-3ed.pdf>)  // копирайт и список изданий (<http://www.emu.edu.tr/aelci/Courses/D-318/D-318-Files/cppcrit/index008.htm>). — 1996.
- Скотт Мейерс. Эффективный и современный С++: 42 рекомендации по использованию С++11 и С++14 = Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14 / Пер. с англ. — Вильямс, 2016. — 304 с. — ISBN 978-5-8459-2000-3.

- *Herbert Schildt*. C++ The Complete Reference Third Edition. — Osborne McGraw-Hill, 1998. — ISBN 978-0-07-882476-0.
- Jeremy A. Hansen. The Rook's Guide to C++ (A Creative Commons-Licensed Textbook) (<http://rooksguide.org>).

Ссылки

-
- C++ (<http://curlie.org/World/Russian/Компьютеры/Программирование/Языки/C++>) в каталоге ссылок [Open Directory Project](#) (dmoz)
- isocpp.org (англ.) — официальный сайт C++
 - рабочие материалы комитета по стандартизации за 2009-й год (<http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2009/>)
- Бьёрн Страуструп. [CppCoreGuidelines](https://github.com/isocpp/) (<https://github.com/isocpp/>) (англ.) The C++ Core Guidelines are a set of tried-and-true guidelines, rules, and best practices about coding in C++
- C++11 (известный как C++0x) — новый стандарт языка C++ (<http://events.yandex.ru/events/uac/2011/talks/19/>) — доклад [Yet another Conference](#)
- Бьёрн Страуструп. Краткий обзор C++0x (<http://www.artima.com/cppsource/cpp0x.html>) (англ.)
- [comp.lang.c++.moderated](https://groups.google.com/group/comp.lang.c++.moderated) (<https://groups.google.com/group/comp.lang.c++.moderated/topics>) (англ.)
- Учебник C++ для начинающих (<http://code-live.ru/tag/cpp-manual/>) (рус.)
- Окончательный список ресурсов для изучения C и C ++ (<https://www.toptal.com/c/the-ultimate-list-of-resources-to-learn-c-and-c-plus-plus>) (англ.)
- Yossi Kreinin. C++ FAQ Lite (<http://yosefk.com/c++faq/index.html>).
- *Линус Торвальдс*. [Convert builin-mailinfo.c to use The Better String Library](http://thread.gmane.org/gmane.comp.version-control.git/57643/focus=57918) (<http://thread.gmane.org/gmane.comp.version-control.git/57643/focus=57918>) (недоступная ссылка). [Gmane forum](http://thread.gmane.org) (<http://thread.gmane.org>) (6 сентября 2007). Архивировано (<https://web.archive.org/web/20131209133259/http://thread.gmane.org/gmane.comp.version-control.git/57643/focus=57918>) 9 декабря 2013 года.

Источник — <https://ru.wikipedia.org/w/index.php?title=C%2B%2B&oldid=114184292>

Эта страница в последний раз была отредактирована 13 мая 2021 в 19:04.

Текст доступен по лицензии Creative Commons Attribution-ShareAlike; в отдельных случаях могут действовать дополнительные условия.

Wikipedia® — зарегистрированный товарный знак некоммерческой организации Wikimedia Foundation, Inc.