

Квентин Окем

---



---

# Ada для программистов C++ или Java

---

Перевод с английского

Минск  
«Колорград»  
2017

УДК 004.43  
ББК 32.973.26-018.2  
О-50

Перевод с английского и комментарии С. И. Киркорова

Перевод осуществлен по статье:

«Ada for the C++ or Java Developer» // [http://www.adacore.com/uploads\\_gems/Ada\\_for\\_the\\_C++\\_or\\_Java\\_Developer-cc.pdf](http://www.adacore.com/uploads_gems/Ada_for_the_C++_or_Java_Developer-cc.pdf)

Статья любезно предоставлена компанией AdaCore

**Окем, К.**

О-50 Ada для программистов C++ или Java / Квентин Окем ; перевод с англ. и комментарии С. И. Киркорова. – Минск : Колорград, 2017. – 170 с.

ISBN 978-985-7170-58-6.

Ада – мощнейший объектно-ориентированный язык общего назначения ориентированный на разработку надежного программного обеспечения. В язык включены механизмы поддержки параллельного исполнения, обработки исключений, настраиваемых модулей, поддержки распределенных вычислений, стандартные интерфейсы с другими языками и библиотеками. Ада имеет компиляторы под практически любую операционную систему плюс Java байткод.

Перевод статьи будет полезен программистам, использующим языки программирования C++, Java и конечно Ada. Также для аспирантов и студентов ВУЗов.

УДК 004.43  
ББК 32.973.26-018.2

ISBN 978-985-7170-58-6

© Quentin Ochem, 2013  
© Киркоров С. И.,  
перевод на русский язык, 2017  
© Оформление.  
ЧПТУП «Колорград», 2017

# Содержание

Историческая справка C++, Java, Ada или предисловие переводчика . . . . .	5
Первая глава. ПРЕДИСЛОВИЕ . . . . .	21
Вторая глава. ОСНОВНЫЕ ПОНЯТИЯ . . . . .	23
Третья глава. СТРУКТУРА ЕДИНИЦЫ КОМПИЛЯЦИИ . . . . .	25
Четвертая глава. ОПЕРАТОРЫ, ОБЪЯВЛЕНИЯ, И УПРАВЛЯЮЩИЕ СТРУКТУРЫ . . . . .	27
1. Операторы и объявления . . . . .	27
2. Условные операторы . . . . .	29
3. Циклы . . . . .	31
Пятая глава. СИСТЕМА ТИПОВ . . . . .	34
1. Строгая типизация . . . . .	34
2. Предопределенные типы языка . . . . .	35
3. Типы, определенные приложением . . . . .	35
4. Диапазоны типа . . . . .	37
5. Обобщенные контракты типа: предикаты подтипа . . . . .	40
6. Атрибуты . . . . .	40
7. Массивы и строки . . . . .	41
8. Разнородные структуры данных . . . . .	45
9. Указатели . . . . .	46
Глава шестая. ФУНКЦИИ И ПРОЦЕДУРЫ . . . . .	50
1. Основная форма . . . . .	50
2. Перегрузка . . . . .	52
3. Контракты подпрограмм . . . . .	52
Глава седьмая. ПАКЕТЫ . . . . .	54
1. Объявление Protection . . . . .	54
2. Иерархия пакетов . . . . .	55
3. Использование сущностей из пакетов . . . . .	55
Глава восьмая. КЛАССЫ И ОБЪЕКТНО–ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ . . . . .	57
1. Примитивные подпрограммы . . . . .	57
2. Наследование (Derivation) и динамическая диспетчеризация (Dispatch) . . . . .	58
3. Конструкторы и деструкторы (Constructors and Destructors) . . . . .	61
4. Инкапсулирование . . . . .	62
5. Абстрактные типы и интерфейсы . . . . .	63
6. Инвариант . . . . .	65
Глава девятая. ОБОБЩЕНИЯ (GENERIC FORMALISM) . . . . .	67
1. Обобщение для подпрограммы . . . . .	67
2. Обобщенные пакеты . . . . .	68
3. Параметры для обобщений . . . . .	70

Глава десятая. ИСКЛЮЧЕНИЯ (EXCEPTIONS FORMALISM) . . . . .	71
1. Стандартные Исключения . . . . .	71
2. Пользовательские исключения . . . . .	72
Глава одиннадцатая. ПАРАЛЛЕЛИЗМ (CONCURRENCY FORMALISM). . . . .	74
1. Задачи. . . . .	74
2. Рандеву. . . . .	78
3. Выборочное рандеву . . . . .	79
4. Защищенные объекты. . . . .	81
Глава двенадцатая. НИЗКОУРОВНЕВОЕ ПРОГРАММИРОВАНИЕ . . . . .	83
1. Уровни представления . . . . .	83
2. Встроенный ассемблерный код . . . . .	84
3. Взаимодействие через интерфейс с С. . . . .	85
Глава тринадцатая. ЗАКЛЮЧЕНИЕ . . . . .	87
Глава четырнадцатая. ССЫЛКИ . . . . .	88
1. Дополнение от переводчика . . . . .	88
2. Новая библиотека OEM как база для доверенных платформ программирования на языке Ada в Win32. . . . .	110
3. Пакет GWindows библиотеки OEM. . . . .	125
4. ЛИТЕРАТУРА . . . . .	170

## Историческая справка C++, Java, Ada или предисловие переводчика

Почти реальная история или притча:

*Студент остался вечером в общежитии один. И так ему захотелось пообедать, что вместо игры, где он был «главным танкистом», начал искать в Интернете рецепт, чтобы приготовить обед.*

*Нашел рецепт борща.*

*«Отлично!» – подумал студент. – «Просто, сытно, быстро и все на кухне есть, а главное, полезная еда».*

*Первый пункт рецепта: «Возьмите 0,5 литра воды».*

*«Да! Вода есть в кране! Но как отмерить точно 0,5 литра??? «Придумал!!! Возьму целую бутылку пива из холодильника. Отмечу фломастером уровень. Вывью содержимое. Залью до уровня воды и получу точно 0,5 литра воды».*

*Пожедакнул студент. Выпил он пиво. И пошел студент после этого к своему компьютеру, не стал он дальше готовить борщ. Так и остался он «главным танкистом». **Не научился студент готовить борщ.***

Профессор факультета компьютерных наук «Харьковского национального университета имени В. Н. Каразина» д. т. н. проф. Виктор Олегович Мищенко ответил на вопрос, зачем изучать язык Ada:

*«Я бы по-разному сказал об этом разным людям.*

*— Тем, кто хочет подготовиться к участию в самых разных проектах создания программных средств (ПС):*

*Никакой язык не покрывает всех требований программирования и никакой не может быть вечно на острие моды.*

В отношении Си, языка ассемблерного типа, на котором написан Unix, это иллюстрируется взрывом его популярности в 80-е годы и поиском высокоуровневых приемников в 90-е. Так синтаксис, импонирующий «посвященным» (забывающим себе голову правилами преобразования, умолчания и т. п.), оказался основой нынешних индустриальных языков, которые вместе с C# возглавляют рейтинги наиболее часто используемых ЯП. Но пройдет и это. Да и популярность по всему свету не означает целесообразности в конкретных условиях! Чем тогда будут замещаться C-подобные системы программирования? Либо «хорошо забытым старым», либо результатами «скрещивания» давно зарекомендовавших себя решений (в истории развития ЯП такое уже не раз бывало). Подготовленным к таким переменам окажется тот, кто владеет несколькими НЕПОХОЖИМИ между собой языками! При этом уникальное свойство Ada развиваться с гарантированным сохранением АКТУАЛЬНОСТИ когда либо написанных программ в новых Ada-системах, подсказывает, что она, скорее всего, будет востребована ЦЕЛИКОМ.

---

*Тем, кто видит себя специалистом по ЯП:*

Знание нескольких «ортогональных» между собой ЯП (т.е. почти во всём непохожих) намного выгоднее, чем того же числа языков, между собой близких. Например, знающий C++, Ada, Пролог, несомненно, знает о ЯП больше и легче может овладеть совершенно новым супермодным языком, чем тот, кто знает только C, C++, C#.

---

*Тем, кто ищет перспективные ниши для своей программистской деятельности.*

Специалистов по Ada на постсоветских просторах – единицы. А область применений этого языка в мире по «абсолютной величине» огромна (прим. автора: имеется в виду – в денежном эквиваленте), хотя в относительном измерении может казаться скромной. Это практически всё ПО критического назначения (не считая «вооружения»), это авионика, атомная энергетика, управляющие системы в других промышленных отраслях, часто встроенные, к которым на Java ну никак не подъедешь!»

Рассмотрим, как появились языки C++, Java, Ada.

Algol 60 можно считать самым значительным шагом в развитии (когда-то существовал также его менее знаменитый предшественник Algol 58, от которого произошел язык Jovial, использовавшийся в военных сферах США). Algol дал ощущение того, что написание ПО – это больше, чем просто коддинг.

При создании Algol было сделано два больших шага вперед. Во-первых, появилось понимание того, что присваивание не есть равенство. Это привело к появлению обозначения := для операции присваивания. Для обозначения различных управляющих структур стали использовать английские слова, в результате отпала необходимость в многочисленных инструкциях goto и метках, которые так затрудняли чтение программы на Fortran и autocode. На втором моменте стоит остановиться более подробно.

Сначала рассмотрим следующие две инструкции в Algol 60:

```
if X > 0 then
    Action(...);
Otherstuff(...);
```

Суть в том, что если X больше нуля, то вызывается процедура Action. Вне зависимости от этого мы далее вызываем Otherstuff. Т.е. действие условия здесь распространяется только на первую инструкцию после then. Если нам необходимо несколько инструкций, например, вызвать две процедуры This и That, то мы должны объединить их в составную инструкцию следующим образом:

```
if X > 0 then
begin
    This(...);
    That(...);
end;
Otherstuff(...);
```

Здесь возникает опасность сделать две ошибки. Во-первых, мы можем забыть добавить `begin` и `end`. Результат соберется без ошибок, но процедура `That` будет вызываться в любом случае. Еще хуже выйдет, если мы нечаянно добавим лишнюю точку с запятой. Наверное, Algol 60 был первым языком, где использовалась точка с запятой как разделитель инструкций. Итак, мы получим:

```
if X > 0 then;
begin
    This (...);
    That (...);
end;
Otherstuff (...);
```

К несчастью, в Algol 60 эта запись означает неявную пустую инструкцию после `then`. В результате условие не будет влиять на вызовы подпрограмм `This` и `That`. Аналогичные проблемы в Algol 60 есть и для циклов.

Разработчики Algol 68 осознали эту проблему и ввели скобочную запись условной инструкции, т. е.:

```
if X > 0 then
    This (...);
    That (...);
fi;
Otherstuff (...);
```

Аналогичная запись появилась для циклов, где слову `do` соответствует `od`, а для `case` есть `esac`. Это полностью решает проблему. Теперь совершенно очевидно, что условная инструкция захватит оба вызова подпрограмм. Появление лишней точки с запятой после `then` приведет к синтаксической ошибке, которая легко будет обнаружена компилятором. Конечно, появление таких записей, как `fi`, `od` и `esac` выглядит причудливо, что может помешать серьезно отнестись к решаемой проблеме.

По какой-то причине разработчики языка Pascal проигнорировали этот здравый подход и оставили уязвимую запись инструкций из Algol 60. Они исправили свою ошибку гораздо позже, уже при проектировании Modula 2. К тому моменту Ada уже давно существовала.

Вероятно, в языке Ada впервые появилась удачная форма структурных инструкций, не использовавшая причудливую запись. На языке Ada мы бы написали:

```
if X > 0 then
    This (...);
    That (...);
end if;
Otherstuff (...);
```

После этого многие языки переняли такую безопасную форму записи. К таковым относится даже макроязык для Microsoft Word for DOS и Visual Basic в составе Word for Windows.

Еще одним значимым языком можно считать CPL. Он был разработан в 1962 г. И использовался в двух новых вычислительных машинах в университетах Кембриджа и Лондона.

CPL (как и Algol 60) использовал `:=` для обозначения присваивания и `=` для сравнения. Вот небольшой фрагмент кода на CPL:

```
§ let t, s, n = 1, 0, 1
  let x be real
  Read[x]
  t, s, n := tx/n, s + t, n + 1
  repeat until t << 1
  Write[s] §
```

Интересная особенность CPL заключается в использовании `=` (вместо `:=`) при задании начальных значений, ввиду того, что при этом не происходит изменения значений. В CPL был ряд интересных решений, например, параллельное присваивание и обработка списков. Но CPL остался лишь академической игрушкой и не был реализован.

Для группировки инструкций язык CPL использовал запись, аналогичную принятой в Algol 60. Например

```
if X > 0 then do
  § This(...)
  That(...) |||§
Otherstuff(...);
```

Для группирования инструкций использовались странные символы параграфа и параграфа с вертикальной чертой.

Хотя сам язык CPL никогда не был реализован, в Кембридже изобрели его упрощенный вариант BCPL (Basic CPL). В отличие от имеющего строгую типизацию CPL, BCPL вообще не имел типов, а массивы были реализованы с помощью адресной арифметики. Так BCPL положил начало проблеме переполнения буфера, от которой мы страдаем по сей день.

BCPL преобразился в B, а затем в C, C++ и т. д. В BCPL использовалось обозначение `:=` для операции присваивания, но по пути кто-то забыл, в чем смысл, и C остался с обозначением `=`. Поскольку знак `=` оказался занят, для операции сравнения пришлось ввести обозначение `==`, а вместе с этим получить букет проблем.

Язык C унаследовал принцип группировки инструкций от CPL, но заменил его странные символы на фигурные скобки. То есть в C мы напишем

```
if (x > 0)
{
  this(...);
  that(...);
};
otherstuff(...);
```



Что же, в C от CPL практически ничего не осталось, ну разве что скобочки для группировки инструкций.

Отметим, что использование знака равенства для обозначения присваивания однозначно осуждал Кристофер Страчи.

Много лет назад, на лекциях НАТО, Кристофер Страчи, один из авторов CPL сказал: «То, как люди учатся программировать, отвратительно. Они снова и снова учатся каламбурить. Они используют операции сдвига вместо умножения, запутывают код, используя битовые маски и числовые литералы, и вообще говорят одно, когда имеют в виду что-то совсем другое. Я думаю, у нас не будет инженерного подхода к разработке программного обеспечения до тех пор, пока у нас не закрепятся профессиональные стандарты о том, как писать программы. А добиться этого можно лишь начиная обучение программированию с того, как писать программы должным образом. Я убежден, что в первую очередь необходимо начать говорить именно то, что вы хотите сказать, а не что-то другое».

История языка Java восходит из проекта Oberon. В 1993 г. в ЕТН приехали представители Sun Microsystems во главе с Биллом Джоом. Они приобрели лицензию на систему Oberon и пригласили выступить у них с ответным визитом лучших учеников Вирта – Микаэль Франц сразу после защиты соответствующей диссертации в ЕТН делал доклад по динамической кодогенерации в Sun Labs в марте 1994 г. Доклад был сделан за 14 месяцев до выхода Java и за полгода до разработки браузера HotJava.

В 1994 г. Франц, разрабатывавший ранее кодогенератор Оберона для MC680x0 (Macintosh), завершил кодогенератор в промежуточный код – OMI (Oberon Module Interchange). Идея Франца была проста – вместо традиционной схемы «компилятор – компоновщик – загрузчик» получить схему «компилятор – кодогенерирующий загрузчик», иными словами, совместить генерацию кода с компоновщиком и загрузчиком в одном флаконе.

Концепция «code-generation on-the-fly» (динамическая кодогенерация, кодогенерация на лету) с использованием компактного древовидного представления вместо классического байт-кода была положена в основу одноименной диссертации М. Франца, которую он защищал в ЕТН в феврале 1994 г. Его научными руководителями были Никлаус Вирт и Юрг Гуткнехт. Крайне интересная диссертация. О ней в среде Modula- и Оберон-сообщества только и говорили (почти с придыханием). Редкий случай – в Цюрихе в марте 1994 г. она была переиздана в виде книги.

В Sun не рискнули сразу копировать все из Oberon (идеи браузерной среды языка, аплетов и трансляции в мобильный код взяли, а вот путь реализации мобильного кода выбрали свой). В 1991 г. автор Java Джеймс Гослинг при реализации Oak (прототипа языка Java) взял старую идею Р-кода, которую хорошо знал: в 1975 г. Гослинг вместе с Недом Китлицем и Бобом Сайдботемом участвовал в построении среды программирования Puxis/Multics Pascal, способной по быстрдействию кода и удобству интеграции на равных конкурировать в Multics с родным для этой ОС языком ПЛ/1. А начинали они с поддержки компилятора ЕТН/Zurich Pascal, разработанного в Цюрихе группой профессора Вирта. В 1979 г. Гослинг реализовал PERQ – транслятор с Р-кода в машинный код DEC VAX.

В 1994 г. в Sun не стали рисковать и включать новейшую хитроумную реализацию мобильного кода в древовидном представлении, что предлагал в диссертации Франц, а сохранили готовый подход Гослинга. Для всей отрасли модель Sun на долгие годы стала эталоном.

Java стремительно ворвалась в ту нишу, которую подготовил себе Оберон, нахраписто и без какого-либо упоминания вырвала многие его идеи, высосала лучшие кадры.

В последние годы ИТ-индустрия насильно превращает университеты в ремесленные училища. На рынке информационных технологий все более усиливается пропагандистская война, битва за умы. Здесь неуместны рассуждения о технологическом совершенстве и, упаси боже, о какой-то там науке!

Со всем по-другому сложилась судьба языка Ada. Именно экономический аргумент Министерства Обороны США послужил и до сих пор служит соединению технологического совершенства и востребованности языка Ada. Давайте освежим в памяти историю проекта Ada.

Проект разработки языка Ada возник в конце золотого века программирования. В то время делалось огромное количество трансляторов и диалектов языков программирования. Министерство обороны США стало замечать, что нарастает количество ситуаций, когда разработчики программного обеспечения не укладываются ни в бюджет, ни во время, и качество результатов тоже резко пошло вниз. Поэтому был запущен проект, в рамках которого должны были определить, какой из аспектов самый критичный. Был исследован вопрос, куда уходят ресурсы.

Результаты исследований показали следующее:

- 5% – на научные исследования
- 19% – на обработку данных
- 56% – встроенные программные системы
- 20% – прочее

Также было замечено, что расходование ресурсов распределяется следующим образом:

- 20% – разработка
- 80% – сопровождение

В 70-е годы по контракту Пентагона использовалось порядка 450 языков программирования общего назначения. Ситуация осложнялась тем, что часто некоторая контора получала заказ от Пентагона на разработку, и для этого разрабатывался новый язык программирования, которым никто не смог бы пользоваться, кроме тех, кто придумал этот новый язык программирования. Им необходимо было удерживать контракт у себя, но к моменту изготовления даже опытного образца контора, получившая заказ, могла уже не существовать, а разработанное ПО и уникальные средства разработки необходимо было сопровождать.

Исследования показали, что можно получить огромную экономию средств на программное обеспечение (около 24 млрд долл. за период 1983–1999 гг., и это при учете затрат на создание инструментария и обучение программистов), если министерство обороны воспользуется единым языком программирования для решения всех своих задач вместо примерно 450 языков программирования и несовместимых диалектов, используемых программистами. Для примера, одна атомная субмарина обходилась не больше 1 млрд долл. Был предложен проект