Java Code Conventions

1 - Введение

1.1 Зачем нужны соглашения по оформлению кода

Соглашения по оформлению кода имеют важное значение по нескольким причинам:

\* 80% от стоимости программного обеспечения приходится на его обслуживание.

\* Вряд ли какое-либо программное обеспечение поддерживается в течение всей своей жизни одним автором.

\* Соглашения по оформлению кода улучшают читабельность программы, позволяя разработчикам намного быстрее и тщательнее понять новый код.

\* Если вы предоставляете свой исходный код как готовую программу, вы должны его хорошо запаковать и очистить, как и остальные программы, которые создаёте.

1.2 Благодарности

Этот документ отражает стандарты языка Java, представленные в Java Language Specification, от Sun Microsystems. Основной вклад внесли Питер Кинг, Патрик Ноутон, Майк Демани, Джонни Канерва, Кэти Уолрат и Скот Хоммель.

По вопросам относительно адаптации, модификации или перераспределения этого документа, пожалуйста прочитайте наше уведомление об авторском праве по ссылке http://java.sun.com/docs/codeconv/html/Copyright.doc.html.

Комментарии к этому документу должны быть представлены в форме обратной связи по http://java.sun.com/docs/forms/sendusmail.html.

2 - Имена файлов

Эта секция перечисляет обычно используемые суффиксы файла и имена.

2.1 Суффиксы файла

JavaSoft использует следующие суффиксы файла:

File Type Suffix

Java source .java

Java bytecode .class

2.2 Имена общего файла

Часто используемые имена файлов включают:

File Name Use

GNUmakefile - Предпочтительное название makefiles.

Мы используем gnumake, чтобы построить наше программное обеспечение.

README - Предпочтительное название файла, который подводит итог содержание конкретного справочника.

3 - Организация файла

Файл состоит из секций, которые должны быть отделены пустыми строками и дополнительными комментариями идентификации каждой секции.

Файлы состоящие более , чем из 2000 строк являются тяжелыми и их нужно избегать.

Для примера JAVA программы, должным образом отформатированной, см. “ Java Source File Example” стр. 19.

3.1 Исходные файлы JAVA

Каждый java исходный файл содержит единственный общественный класс или интерфейс. Когда частные классы и интерфейсы связаны с общественным классом, Вы можете поместить их в тот же самый исходный файл как общественный класс. Общественный класс должен быть первым классом или интерфейсом в файле.

У исходных файлов есть следующий состав:

• Начальный комментарий (см. “Начинающиеся Комментарии” на странице 4),

• Пакет и заявления Импорта; например:

import java.applet.Applet;

import java.awt.\*;

import java.net.\*;

• Класс и интерфейсные декларации (см. “Класс и Интерфейсные Декларации” на странице 4),

3.1.1 Начало комментариев

Все исходные файлы должны начаться с комментария c-стиля, который перечисляет программиста (программистов), дату, a уведомление об авторском праве, и также краткое описание цели программы. Например:

/\*

\* Название класса

\*

\* Информация о версии

\*

\* Уведомление об авторском праве

\*/

3.1.2 package и import

Первая строчка в большинстве java исходных файлов - объявление пакета. После этого,

могут следовать объявление импорта. Например:

package java.awt;

import java.awt.peer.CanvasPeer;

3.1.3 Объявление классов и интерфейсов

Ниже описаны части класса или интерфейсной декларации.( См. “Явский Пример Исходного файла” на странице 19)

|  |  |  |
| --- | --- | --- |
|  | Часть Класса/Интерфейса  Примечания к декларации | Заметки |
| 1 | Документирующий комментарий класса/интерфейса (/\*\*...\*/) | См. “Комментарии Документации” на странице 9 |
| 2 | Оператор class или interface |  |
| 3 | Если необходимо, то указать комментарий реализации класса/интерфейса (/\*...\*/) | Здесь содержится любая дополнительная информация по классу или интерфейсу, которая не подходит для документирующего комментария. |
| 4 | Класса (статические) переменные | Сначала открытые (public), затем защищенные (protected) и, наконец, закрытые члены класса (private). |
| 5 | Переменные (поля) экземпляра | Сначала public, затем protected, после private. |
| 6 | Конструктор |  |
| 7 | Методы | Эти методы должны быть сгруппированы по функциональности, а не по области действия или доступности. Например, закрытый (private) метод класса может находится между двумя открытыми. Цель - сделать проще чтение и ясность кода. |

4 - Отступы

В качестве единицы отступа используется 4 пробела. Точное построение отступов (пробелы или табуляция) не определено. Табуляция должна быть установлена как 8 пробелов (не 4).

4.1 Длина линии

Избегайте строчек длиннее, чем 80 знаков, так как они не обрабатываются многими терминалами и инструментами.

Примечание: В приведенном примере для использования в документации должна быть более короткая строка — обычно не больше чем 70 знаков.

4.2 Оформление строк

Когда выражение не влезает на одну строку, разделять его нужно согласно этим общим принципам:

• Разрыв после запятой.

• Разрыв перед оператором.

• Выбирайте высокоуровневые разрывы, разрывам низшего уровня.

• Выровняйте новую строку с началом выражения на том же самом уровне на предыдущей строке.

Вот некоторые примеры переноса:

function(longExpression1, longExpression2, longExpression3,

longExpression4, longExpression5);

var = function1(longExpression1,

function2(longExpression2,

longExpression3));

Следующее - два примера переноса арифметического выражения.

longName1 = longName2 \* (longName3 + longName4 - longName5)

+ 4 \* longname6;//ПРЕДПОЧТИТЕЛЬНО

longName1 = longName2 \* (longName3 + longName4

- longName5) + 4 \* longname6;//ИЗБЕГАЮТ

Следующие два примера иллюстрируют правила. Первым - является обычный случай. Второй - следовало бы переместить вторые и третьи линии к правому краю, если бы использовались обычные отступы, а вместо этого ставятся только 8 пробелов.

//ОБЫЧНОЕ УГЛУБЛЕНИЕ

someMethod(int anArg, Object anotherArg, String yetAnotherArg,

Object andStillAnother) {

...

}

//ПЕРЕНОС 8 ПРОБЕЛОВ

private static synchronized horkingLongMethodName(int anArg,

Object anotherArg, String yetAnotherArg,

Object andStillAnother) {

...

Отступы делают код менее читаемым . Например:

//НЕ ИСПОЛЬЗУЙТЕ ЭТО

if ((condition1 && condition2)

|| (condition3 && condition4)

||!(condition5 && condition6)) { //BAD WRAPS

doSomethingAboutIt(); //MAKE THIS LINE EASY TO MISS

}

//ИСПОЛЬЗУЙТЕ

if ((condition1 && condition2)

|| (condition3 && condition4)

||!(condition5 && condition6)) {

doSomethingAboutIt();

}

//ИЛИ ИСПОЛЬЗУЙТЕ ЭТО

if ((condition1 && condition2) || (condition3 && condition4)

||!(condition5 && condition6)) {

doSomethingAboutIt();

}

Вот три приемлемых способа отформатировать троичные выражения:

alpha = (aLongBooleanExpression) ? beta : gamma;

alpha = (aLongBooleanExpression) ? beta

: gamma;

alpha = (aLongBooleanExpression)

? beta

: gamma;

5 - Комментарии

Программа на Java может иметь два вида комментариев: комментарий реализации и документирующий комментарий. Комментарий реализации те же, что и в C++, обозначающиеся /\* ... \*/ и //. Документирующие комментарии (известные как "doc comments" или "Javadoc") есть только в Java, и обозначаются /\*\* ... \*/. Javadoc может быть извлечен из кода в HTML файл, используя инструмент javadoc.

Комментарии кода используются для описания отдельных строк/блоков кода или целого алгоритма. Комментарии для документирования используются, чтобы описать спецификацию кода (его интерфейс), не зависящую от его реализации. Комментарии для документирования делают для разработчиков, которые будут использовать ваши программы (библиотеки классов) не имея их исходного кода.

Комментарии нужны, чтобы описать код или пояснить моменты, которые сложно понять непосредственно из кода. Комментарии должны содержать лишь ту информацию, которая необходима для чтения и понимания кода программы. Например информацию о том, как откомпилировать связанный пакет или в какой директории он находится не стоит описывать в комментарии.

Обсуждение нетривиальных или неочевидные решений необходимо, но не нужно описывать то, что и так ясно из кода. Такие "избыточные" комментарии очень быстро могут стать неактуальными. В общем, следует избегать любых комментариев, которые могут стать неактуальными по мере развития кода.

Примечание: большое количество комментариев иногда отражает низкое качество кода. Когда вы чувствуете, что необходимо добавить комментарий, подумайте: может лучше переписать код, чтобы он стал более понятным.

Не стоит делать огромных комментариев, отделенных от основного кода строками из "\*" или других символов.

Комментарии не должны содержать специальных символов, каких как символ конца страницы или backspace.

5.1 Форматы комментариев

У программ может быть четыре стиля комментариев : блок, одна линия, прицепные и

конец строки.

Блочные комментарии используются для предоставления описания файлов, методов, структур данных и алгоритмов. Блочные комментарии следует использовать в начале каждого файла и перед каждым методом. Они также могут быть использованы в других местах, например, внутри методов. Блочные комментарии внутри функции или метода должны иметь отступ на том же уровне, что и код, который они описывают.

Перед блочным комментарием следует оставлять пустую строку, чтобы визуально отделить его от кода. Каждая строка блочного комментария (кроме первой) должна начинаться с символа "\*".

/\*

\* Здесь блок комментариев.

\*/

Если блочный комментарий начинатся с "/\*-", это означает в данном блоке используется особое форматирование, которое нельзя потерять. (Такой блок не будет переформатирован средствами автоформатирования) Пример:

/\*

\* Этот блочный комментарий содержит очень специфичное

\* форматирование, которое должно игнорироваться средствами автоформатирования

\*

\* один

\* два

\* три

\*/

Примечание: Если вы не используете средства автоформатирования, вам не обязательно использовать "/\*-" в коде, но можете сделать это на случай того, что кто-то другой может запустить средства автоформатирования на вашем коде.

Смотри также "Комментарии для документирования" на странице 9

5.1.2 Однострочные комментарии

Короткие комментарии можно писать на одной строке используя отступ на уровне соответствующего блока кода. Если комментарий не помещается в одну строку, следует использовать блочный комментарий (см. раздел 5.1.1). Перед однострочным комментарием следует оставлять пустую строку. Вот пример однострочного комментария в Java коде (см. также "Комментарии для документирования» на стр. 9):

if (condition) {

/\* Обработка условия. \*/

...

}

5.1.3 Прицепные комментарии

Очень короткий комментарий можно расположить на той же строке, что и описываемый код. При этом его следует сдвинуть вправо настолько, чтобы он не сливался с кодом. Если в одном блоке кода присутствует более одного такого комментария, их начало должно быть на одном уровне.  
Старайтесь избегать комментирования каждой строчки кода подобным образом.

Рассмотрим пример прицепных комментариев в Java коде (смотри также "Комментарии для документирования" на странице 9):

if (a == 2) {

return TRUE; /\* частный случай \*/

} else {

return isprime(a); /\* работает только для нечетных a \*/

}

5.1.4 Комментарии конца линии

Сиволами / / начинается комментарий, который продолжится до следующей строки. Он может занимать всю строку или только ее часть. Его не следует использовать для многострочных комментариев, однако его можно использовать, чтобы закомментировать несколько строк кода. Рассмотрим пример всех трех вариантов его использования:5 - Комментарии

если (foo&gt; 1) {

//Сделайте двойной щелчок.

...

}

еще

возвратитесь ложный;//Объясняют почему здесь.

//если (бар&gt; 1) {

//

////Делают тройной щелчок.

//...

//}

//еще

//возвратитесь ложный;

5.2 Комментарии документации

Примечание: См. “Пример файла исходного кода Java” на странице 19 для примеров форматов комментария

описанный здесь.

Для получения дальнейшей информации см., “ Как писать документирующие комментарии для Javadoc ”, которая включает в себя информацию о тегах документирующих комментариев (@return, @param, @see):

http://java.sun.com/products/jdk/javadoc/writingdoccomments.html

Для получения большей информации о документирующих комментариях и Javadoc, посетите домашнюю страницу Javadoc:

http://java.sun.com/products/jdk/javadoc/

Документирующие комментарии описывают Java классы, интерфейсы, конструкторы, методы и поля. Каждый комментарий помещается в ограничители /\*\* ... \*/, один комментарий на один элемент API. Такой комментарий должен располагаться непосредственно перед объявлением:

/\*\*

\* Пример комментария к классу

\*/

class Example { ...

Заметьте, что классы и интерфейсы не имеют отступа, в отличие от их членов. Первая линия документирующего комментария (/\*\*) для класса или интерфейса не имеет отступа; последующие линии комментария имеют по одному пробелу отступа (для вертикального выравнивания звёздочек). Члены класса, включая конструкторы, имеют 4 пробела для первой строки документирующего комментария и 5 для остальных.

Если необходимо предоставить информацию о классе, интерфейсе, переменной или методе, не принадлежащую к документации, используйте блочный (см. раздел 5.1.1) или однострочный (см. раздел 5.1.2) комментарий непосредственно после объявления. Например, детали о реализуемом классе должны идти в таком комментарии в блоке реализации, а не в документирующем комментарии.

Документирующий комментарий не должен располагаться внутри блока реализации метода или конструктора, потому что Java связывает документирующий комментарий с первым после него объявлением.

6 - Декларации

6.1 Число объявлений в строке

Рекомендуется использовать одно объявление на строку, так как это облегчает комментирование. Другими словами,

int level; // уровень отступа

int size; // размер таблицы

предпочтительней, чем

int level, size;

Абсолютно в любом случае переменная и функция не должны быть объявлены на одной линии. Например:

long dbaddr, getDbaddr(); // НЕВЕРНО!

Не располагайте разные типы в одной строке. Пример:

int foo, fooarray[]; //НЕВЕРНО!

Заметьте: В вышеприведенных примерах используется один пробел между типом и идентификатором. Другой приемлемой альтернативой использования отступов является следующая:

int level; // уровень отступа

int size; // размер таблицы

Object currentEntry; // текущий выделенный экземпляр таблицы

6.2 Размещение

Поместите объявления только в начале блока. (Блок - это код, окруженный фигурными скобками "{" и "}".) Не ждите первого использования переменной перед её объявлением. Это может спутать неосторожного программиста и препятствовать переносимости кода.

void MyMethod() {

int int1; // начало блока метода

if (condition) {

int int2; // начало условного блока

...

}

}

Исключение из правила составляет случай индексирующей переменной в цикле, которая в Java может быть объявлена внутри оператора for:

for (int i = 0; i < maxLoops; i++) { ...

Избегайте локальных объявлений, перекрывающих объявления более высокого уровня. Например, не объявляйте то же имя переменной во внутреннем блоке.

6.3 Инициализация

Попытайтесь инициализировать местные переменные, там где они объявлены. Единственная причина не инициализировать переменную в месте её объявления — если её начальное значение зависит от некоторых предварительных вычислений. 6.4

6.4 Объявление классов и интерфейсов

При программирования Java классов и интерфейсов следует придерживаться следующих правил форматирования:

\* Нет пробелов между именем метода и скобкой "(", с которой начинается список параметров

\* Открывающая скобка "{" ставится в конце той же строки, где указан оператор объявления класса или интерфейса

\* Закрывающая скобка "}" ставится на отдельной строке с тем же отступом, что и у соответствующего ему открывающего оператора, кроме случаев, когда имеем пустой оператор, то "}" должна появиться сразу после "{"

class Sample extends Object {

int ivar1;

int ivar2;

Sample(int i, int j) {

ivar1 = i;

ivar2 = j;

}

int emptyMethod() {}

...

}

\* Методы разделяются одной пустой строкой

7 Операторы

7.1 Простые операторы

Каждая строка должна содержать не более одного выражения. Пример:

argv++; argc--; // ИЗБЕГАТЬ!

Не используйте запятую для группировки нескольких операторов, даже если это видно невооруженным глазом. Например:

if (err) {

Format.print(System.out, "error"), exit(1); //СОВЕРШЕННО НЕПРАВИЛЬНО!

}

7.2 Составные операторы

Составные операторы - это операторы, которые содержат списки операторов, заключенных в фигурные скобки "{операторы}". Примеры смотрите в последующих разделах.

\* Вложенные операторы должны иметь отступ на один уровень больше, чем составной оператор.

\* Открывающая скобка должна быть в конце той строки, с которой начинается составной оператор; закрывающая скобка должна начинаться с новой строки и с отступом, соответствующим началу составного оператора.

\* Скобки используются во всех операторах, даже в одиночных, когда они входят в состав управляющей структуры, таких, как оператор if-else или for. Это удобно, чтобы избегать ошибок в случае добавления операторов, когда забыли указать фигурные скобки.

7.3 Оператор return

Оператор return, возвращающий значение, не должен использовать скобки, если только их использование не сделает возвращаемое значение более понятным. Например:

return;

return myDisk.size();

return (size ? size : defaultSize);

7.4 Операторы if, if-else, if-else-if-else

Оператор if-else должен иметь следующие формы:

if (условие) {

операторы;

}

if (условие) {

операторы;

} else {

операторы;

}

if (условие) {

операторы;

} else if (условие) {

операторы;

} else if (условие) {

операторы;

}

Внимание: оператор if должен всегда использовать фигурные скобки {}. Избегайте следующих ошибок:

if (условие) //ИЗБЕГАТЬ! ОПУЩЕНЫ СКОБКИ {}!

оператор;

7.5 Оператор for

Оператор for должен иметь следующую форму:

for (начальное значение; условие; приращение) {

операторы;

}

Пустой оператор for (в котором вся работа состоит в установке начального значения, условия и обновления значения) должен иметь следующий вид:

for (начальное значение; условие; обновление);

Когда используется оператор запятая а блоке инициализации или обновления оператора for, избегайте запутанного использования более чем трех переменных. Если необходимо, используйте отдельные операторы перед циклом for (для случая блока инициализации) или в конце цикла (для случая блока обновления)

7.6 Оператор while

Оператор while должен иметь следующую форму:

while (условие) {

операторы;

}

Пустой оператор while должен иметь следующий вид:

while (условие);

7.7 Оператор do-while

Оператор do-while должен иметь следующую форму:

do {

операторы;

} while (условие);

7.8 Оператор switch

Оператор switch должен иметь следующую форму:

switch (условие) {

case значение1:

операторы;

/\* потерпеть неудачу \*/

case значение2:

операторы;

break;

case значение3:

операторы;

break;

default:

операторы;

break;

}

Каждый раз, когда выбор проваливается (не включая оператор break), добавить комментарий, где обычно находится оператор break. В предыдущем примере кода показан комментарий /\* потерпеть неудачу \*/.

Каждый оператор switch должен включать выбор по умолчанию (default). Оператор break в выборе по умолчанию лишний, но он предотвращает возникновение ошибки, если позже еще ​​один выбор добавится.

7.9 Оператор try-catch

Оператор try-catch должен иметь следующую форму:

try {

операторы;

} catch (ExceptionClass e) {

операторы;

}

8 Пробелы

8.1 Пустые строки

Пустые строки улучшают читабельность выделенных участков кода, которые логически связаны между собой.

Две пустые строки всегда должны использоваться в следующих случаях:

\* Между секциями в файле исходного кода

\* Между определениями класса и интерфейса

Одна пустая строка всегда должна использоваться в следующих случаях:

\* Между методами

\* Между локальными переменными метода и его первым оператором

\* Перед блочным (см. секцию 5.1.1) или однострочным (см. секцию 5.1.2) комментарием

\* Между логическими участками кода внутри метода для улучшения читабельности

8.2 Расстановка пробелов

Разделяющие пробелы должны ставиться при следующих обстоятельствах:

\* Ключевое слово и следующие за ним скобки должны быть разделены пробелом. Например:

while (true) {

...

}

Заметьте, что разделяющий пробел не используется между именем метода и его открывающей скобкой. Это помогает отличить ключевое слово от вызова функции.

\* Разделяющий пробел должен появляться после запятой в списке аргументов.

\* Все бинарные операции, исключая . должны быть отделены от их операндов пробелами. Разделяющие пробелы никогда не отделяют унарные операторы, такие как унарный минус, инкремент ("++") и декремент ("--") от их операндов. Например:

a += c + d;

a = (a + b) / (c \* d);

while (d++ = s++) {

n++;

}

prints("size is " + foo + "\n");

\* Выражения в операторе for должны быть разделены пробелами. Например:

for (expr1; expr2; expr3)

\* За приведением типа должен следовать пробел. Например:

myMethod((byte) aNum, (Object) x);

myFunc((int) (cp + 5), ((int) (i + 3))

+ 1);

9 Соглашение об именовании

Соглашение об именовании делает программы более понятными, облегчая их чтение. Они также могут дать информацию о функции идентификатора - например, является ли он константой, пакетом или классом, что может быть полезно для понимания кода.

Правила, приведенные в данном разделе, являются основополагающими. Более конкретные правила приведены в (нужно уточнить).

Тип идентификатора | Правила наименования/примеры

---------------------------------------------------------------------------------

Классы | Имя класса должно быть существительным, набранным в смешанном регистре с первым символом каждого слова в верхнем регистре. Старайтесь сохранить имя вашего класса простым и наглядным. Используйте целые слова - избегайте сокращений и аббревиатур (не считая аббревиатур, использующихся чаще своих длинных форм, таких как URL и HTML).

|class Raster; class ImageSprite;

Интерфейсы | Имена интерфейсов должны быть в верхнем регистре, так же, как и имена классов.

|interface RasterDelegate; interface Storing;

Методы | Методы должны быть глаголами, набранными в смешанном регистре с первым символом в нижнем регистре для первого, с первым символом в верхнем регистре для остальных слов.

|run(); runFast(); getBackground();

Переменные | Исключая случаев названия переменных, все экземпляры классов и констант классов набираются в смешанном регистре с первым символом в нижнем регистре. Последующие слова набираются с большой буквы.

| Имена переменных должны быть короткими, но осмысленными. Выбранное название переменной должно быть запоминающееся - то есть предназначено для краткого выражения своего содержимого для человека, временно использующего её. Односимвольные названия переменных следует избегать, исключая временных "одноразовых" переменных. i, j, k, m и n - это общеупотребительные имена для временных целочисленных переменных; c, d и e - для символьных.

|int i; char \*cp; float myWidth;

Константы | Названия переменных, объявленные константами класса, и ANSI константы должны быть набраны заглавными буквами с разделенными знаком подчеркивания "\_" словами. (ANSI констант следует избегать для более легко отладки.)

|int MIN\_WIDTH = 4; int MAX\_WIDTH = 999; int GET\_THE\_CPU = 1;

10 Приёмы программирования

10.1 Доступ к переменным класса и экземпляра

Не делайте переменную класса или экземпляра public без значительной причины. Часто встречаются случаи, когда поля класса не должны устанавливаться или считываться напрямую - их чтение и запись должны происходить лишь при вызове методов.

Одним из примеров уместного использования public полей может быть случай, когда класс описывает лишь структуру данных, без какого-либо поведения. Другими словами, если бы вы могли использовать struct вместо class (если бы Java поддерживал struct), тогда можно сделать переменные экземпляра класса public.

10.2 Обращение к переменным и методам класса

Избегайте использование объекта для доступа к статическим полям и методам класса. Вместо этого используйте имя класса. Например:

classMethod(); //OK

AClass.classMethod(); //OK

anObject.classMethod(); //ИЗБЕГАЙТЕ!

10.3 Константы

Числовые константы (литералы) не должны быть закодированы непосредственно, исключая -1, 0, 1, которые могут использоваться в циклах для управления счетчиком.

10.4 Присваивание значений переменным

Избегайте присваивания значения некоторым переменным в одном выражении. Это усложняет чтение. Например:

fooBar.fChar = barFoo.lchar = 'c'; // ИЗБЕГАТЬ!

Не используйте оператор присваивания в местах, где он может быть легко спутан с оператором сравнения. Например:

if (c++ = d++) { // ИЗБЕГАТЬ! В Java запрещено

...

}

должно быть записано как

if ((c++ = d++) != 0) {

...

}

Не используйте вложенные присваивания, пытаясь ускорить скорость выполнения программы. Это работа компилятора, и, кроме того, на самом деле редко помогает. Например:

d = (a = b + c) + r; // ИЗБЕГАТЬ!

должно быть записано как

a = b + c;

d = a + r;

10.5 Разные приёмы программирования

10.5.1 Скобки

Действительно хорошей практикой является свободное, избыточное использование скобок в выражениях, содержащих различные операторы, для избежания проблем с приоритетами операций. Даже если приоритет оператора понятен вам, другим же он может быть неясен - не стоит думать, что остальные программисты знают приоритеты операций так же хорошо, как и Вы.

if (a == b && c == d) // ИЗБЕГАТЬ!

if ((a == b) && (c == d)) // Так лучше

10.5.2 Возвращаемые значения

Попробуйте сделать структуру вашей программы понятной. Например:

if (booleanExpression) {

return TRUE;

} else {

return FALSE;

}

следует переписать, как

return booleanExpression;

Аналогичным образом

if (condition) {

return x;

}

return y;

следует переписать, как

return (condition ? x : y)

10.5.3 Выражения перед '?' в условном операторе

Если выражение содержит бинарный оператор, находящийся перед тернарным оператором ?:, он должен быть заключен в скобки. Например:

(x >= 0) ? x : -x

10.5.4 Специальные комментарии

Используйте XXX в комментарии для того, чтобы показать, что этот код неправильный, но работает. Используйте FIXME для того, чтобы показать, что код неправильный и не работает.