УДК 004.4'2

# БИБЛИОТЕКА УНИВЕРСАЛЬНОГО АВТОФОРМАТИРОВАНИЯ КОДА ДЛЯ ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ ЯЗЫКОВ

**Советов П.Н.**

*Федеральное государственное бюджетное образовательное учреждение высшего образования «МИРЭА Российский технологический университет» (РТУ МИРЭА), 119454, Россия, г. Москва, проспект Вернадского, 78, e-mail: sovetov@mirea.ru*

**В настоящее время существуют различные стандарты оформления кода, в том числе в рамках единственного языка программирования. Для удобства соблюдения этих стандартов в своих программах разработчики применяют инструменты автоматического форматирования кода. Существует также необходимость в универсальных средствах автоматического форматирования кода, подходящих для языков с самым различным синтаксисом. Данная необходимость, в частности, возникает в тех задачах, где применяются проблемно-ориентированные языки. В таких задачах часто предпочтительнее иметь не внешний инструмент автоформатирования кода, а интегрированное в общий программный проект решение, на основе библиотеки. Такая библиотека разработана автором на языке Python. В данной статье представлен обзор современных стандартов оформления кода, а также приведено описание и возможности разработанной библиотеки.**

Ключевые слова: автоматическое форматирование кода, стандарт оформления кода, проблемно-ориентированный язык, генератор программ, Python.

# AN UNIVERSAL CODE FORMATTING LIBRARY FOR DOMAIN-SPECIFIC LANGUAGES

**Sovietov P.N.**

*Federal State Educational Institution of Higher Education «MIREA - Russian Technological University» (RTU MIREA), 119454, Russia, Moscow, Vernadsky avenue, 78, e-mail: sovetov@mirea.ru*

**There are various code style standards, including the ones for a single programming language. For ease of compliance with these standards, developers use automatic code formatting tools in their programs. There is also a need for universal means of automatic formatting of code, suitable for languages with a very different syntax. This need, in particular, arises in those tasks where domain-specific languages are used. In such tasks, it is often preferable to not have an external code auto-formatting tool, but to have a solution based on a library integrated into a whole software project. Such a library was developed by the author in the Python language. This article provides an overview of modern code style standards and a description and features of the developed library.**

Keywords: automatic code formatting, code style standard, domain-specific language, Python.

**Введение**

В настоящее время широко распространены различные стандарты на оформление программного кода. Такие стандарты повышают читаемость программ, а также помогают коллективно разрабатывать программное обеспечение (ПО), не отвлекаясь на разрешение противоречий между личными предпочтениями в оформлении кода отдельных участников проекта.

Стандарты оформления кода для конкретного языка программирования могут использоваться на различных уровнях:

- стандарт на программный проект (примеры: проекты Linux Kernel [1], LLVM);
- корпоративный стандарт (примеры: компании Google [2], Mozilla);
- языковый стандарт (примеры: языки Python [4], Go [6]).

На сегодняшний день стандарт оформления кода можно встретить в виде вспомогательного документа, который не включается в официальный стандарт на сам язык программирования.

В таблице 1 представлено семь широко применяемых (C, C++, Java, Python, C#) или набирающих популярность на момент написания данной статьи (Go, Kotlin) языков программирования, с указанием соответствующих данным языкам стандартов оформления кода.

Таблица 1. Языки программирования и стандарты оформления кода.

| Язык | Стандарты оформления кода |
|---|---|
| C | Linux Kernel [1] |
| C++ | Google [2], LLVM, Chromium, Mozilla, WebKit |
| Java | Google [3] |
| Python | Официальный (PEP 8) [4] |
| C# | Официальный [5] |
| Go | (Go Code Review Comments) [6] |
| Kotlin | Официальный [7] |

В стандарт на оформление кода обычно входят соглашения по следующим пунктам:

- значение отступа (табуляция или некоторое число пробелов);
- допустимость наличия нескольких операторов в строке;
- максимально допустимая длина строки;
- расположение группирующих скобок при оформлении блоков операторов (примеры: стили K&R, Allman);
- оформление комментариев;
- именование идентификаторов (примеры: CamelCase, snake_case);
- специфичные для данного языка соглашения.

Существуют инструменты, позволяющие автоматически переформатировать программу для приведения ее в соответствие с некоторым стандартом оформления кода. Такие инструменты способны обеспечить выполнение лишь части пунктов стандарта: автоматическое изменение имен идентификаторов или переоформление комментариев в большинстве случаев не поддерживается. Тем не менее, инструменты автоформатирования кода способны оказать большую помощь разработчикам в избавлении от большей части рутинных действий на пути приведения программы к некоторому стандартному виду. В таблице 2 сравниваются соглашения из различных стандартов оформления кода и приводятся соответствующие этим стандартам инструменты автоформатирования кода.

Таблица 2. Стандарты оформления кода и инструменты автоформатирования кода.

| Стандарт оформления кода | Значение отступа | Несколько операторов в строке | Макс. длина строки | Оформление блоков операторов | Инструмент автоматического форматирования кода |
|---|---|---|---|---|---|
| Linux Kernel C | Табуляция | Не разрешено | 80 | K&R | ClangFormat |
| Google C++ | 2 пробела | Разрешено | 80 | K&R | ClangFormat |
| Google Java | 2 пробела | Не разрешено | 100 | K&R | google-java-format |
| Python | 4 пробела | Не разрешено | 79 | Зафиксировано в синтаксисе | autopep8 |
| C# | 4 пробела | Не разрешено | Нет | Allman | CodeFormatter |
| Go | Табуляция | Разрешено | Нет | K&R | gofmt |
| Kotlin | 4 пробела | Разрешено | Нет | K&R | плагин для IntelliJ IDEA |

Инструменты автоформатирования кода широко применяются сегодня. Например, в среде программистов на языке Go принято использовать утилиту gofmt перед обменом исходными текстами с другими разработчиками. При этом, как видно из таблицы 2, для каждого языка программирования обычно

используется свой собственный инструмент автоформатирования кода (в случае ClangFormat поддерживается семейство языков с Си-подобным синтаксисом).

В некоторых случаях возникает необходимость в использовании универсального, нейтрального по отношению к языку, инструмента автоформатирования кода. В частности, такой инструмент может понадобиться для некоторого проблемно-ориентированного языка (DSL, domain-specific language), разработанного в рамках сложного программного проекта. Если DSL используется целым рядом специалистов в некоторой предметной области, то введение автоматизации процесса оформления кода по определенному стандарту представляется вполне оправданным. Кроме того, универсальный инструмент оформления кода может быть использован в работе генераторов программ. Порожденный автоматически код часто приходится оформлять в соответствии с остальной частью программного проекта и здесь также полезным может оказаться инструмент автоформатирования кода.

Предлагаемый в таких работах, как [8], [9] и [10], универсальный подход к автоформатированию кода реализуется на основе внешних инструментов. Из рассмотренных примеров с DSL и генераторами программ видно, что инструмент автоформатирования кода может быть также полезно иметь во встроенном в общий программный проект виде — в форме небольшой программной библиотеки.

Таким образом, возникает задача создания библиотеки, которая обладала бы следующими свойствами:

- выразительное описание различных вариантов синтаксиса выходных представлений с элементами форматирования по некоторому стандарту;
- интеграция с каким-либо существующим средством создания DSL;
- простая, расширяемая реализация на современном языке программирования.

**Библиотека универсального автоформатирования кода**

Для удобного описания процесса автоматического форматирования кода было принято решение использовать формальные правила, представляющие в декларативном духе преобразования с уровня абстрактного синтаксического дерева (AST, abstract syntax tree) на уровень описания по модели Box [8]. Описание программы на уровне модели Box соответствует конкретному выходному языку и учитывает выбранный вариант форматирования кода. Конструкции упрощенной версии языка Box представлены в таблице 3.

Таблица 3. Конструкции языка Box.

| Конструкция | Описание |
|---|---|
| H(box1, box2, …, sep=разделитель) | Горизонтальное расположение элементов с возможным указанием разделителя между ними |
| V(box1, box2, ...., tab=отступ) | Вертикальное расположение элементов с возможным отступом |
| "Текст" | Простая строка, которая возвращается как есть |

Каждому выходному языку соответствует свой набор правил, преобразующих AST программы в представление на языке Box. Представление программы на языке Box передается на вход функции fmt, которая выдает отформатированный результат в виде текста.

Общая схема работы данной библиотеки показана на рис. 1.

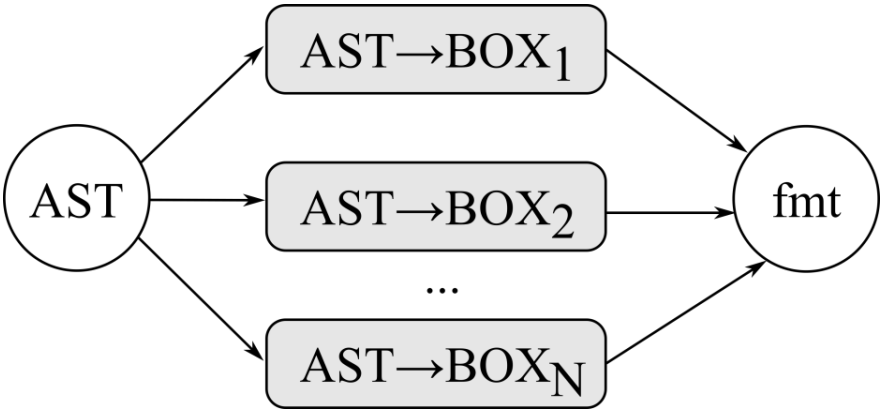


Рис 1. Схема работы библиотеки универсального автоформатирования кода.

Задача автоформатирования кода тесно связана с задачей «красивой печати» (pretty-printing) структур кода и данных. В рамках этой задачи традиционно рассматривается проблема интеллектуального разбиения исходного текста на строки, которые бы не превышали некоторый предел по количеству символов. Здесь следует отметить работу [11], развитием подходов из которой занимались многие поздние исследователи, рассматривающие задачу pretty-printing. Вместе с тем, в стандартах на оформление кода для современных языков программирования, как показывает таблица 2, не имеется ограничений на максимальную длину строки программы. Рассматриваемая в данном разделе библиотека также не имеет подобного ограничения на число символов в строке. Данное решение позволило упростить реализацию библиотеки.

В качестве языка реализации рассматриваемой библиотеки выбран язык Python. Для обеспечения декларативного стиля работы на уровне AST-термов с использованием сопоставления с образцом на уровне термов применен инструментарий быстрой разработки DSL-компиляторов raddsl [12] для Python. Пример использования ранней версии данного инструментария описан в [13].

Инструментарий raddsl состоит из двух модулей – библиотек комбинаторов:

1) parse.py. Модуль предназначен для реализации лексического и синтаксического анализа.

2) rewrite.py. Модуль предназначен для реализации преобразований уровня AST и порождения целевого представления.

Для задач автоформатирования кода использован модуль rewrite.py.

Пример набора AST-термов для описания промежуточного представления учебного языка приведен в таблице 4.

Таблица 4. Набор AST-термов учебного языка.

| Терм | Описание |
|---|---|
| Int(значение) | Целое число |
| Id(имя) | Идентификатор |
| Bop(операция, выраж., выраж.) | Бинарная операция |
| Assign(Id(имя), выраж.) | Операция присваивания |
| If(выраж., блок) | Условный оператор |
| Func(Id(имя), [], блок) | Функция без аргументов и возвращаемого значения |

При реализации преобразований AST→Box используется стратегия обхода дерева сверху вниз, которая позволяет учесть особые случаи оформления конструкций языка перед обработкой общих случаев.

Работа функции fmt осуществляется в два этапа:

1) Unbox («распаковка»). Слияние вложенных H-элементов и простых строк, с учетом разделителей, в единственный H-элемент, представляющий собой строку кода. Перевод V-элементов в упрощенную списковую форму.

2) Flatten («сглаживание»). Объединение списков с учетом отступов и переводов строк.

Реализация функции fmt на языке Python показана на рис. 2.

```
def fmt(box):
  def unbox(box):
    if is_term(box):
      hd, tl = box[0], [unbox(x) for x in box[1:]]
      if hd == "H":
        return hd.get("sep", " ").join(tl)
      if hd == "V":
        return [hd.get("tab", ""), tl]
    return box

  def flatten(lst, tab=""):
    if isinstance(lst, list):
      return "\n".join([flatten(y, tab + lst[0]) for y in lst[1]])
    return tab + lst

  return flatten(unbox(box))
```

Рис. 2. Реализация функции fmt на языке Python.

**Примеры описания целевых представлений**

В примерах, представленных далее, демонстрируется работа библиотеки. В качестве входного AST-представления используется набор термов из предыдущего раздела. AST-представление может быть получено для программы на некотором DSL с помощью средств разработки синтаксических анализаторов из состава raddsl. В случае использования генератора программ AST-представление может быть построено самим генератором.

В первом примере представлено преобразование AST в подмножество языка Си, программа на котором оформлена в соответствии со стандартом Linux Kernel. Правила преобразований и пример результата показаны во второй строке таблицы 5. Во втором примере представлено преобразование AST в подмножество языка Python, программа на котором оформлена в соответствии со стандартом PEP 8. Правила преобразований и пример результата показаны в третьей строке таблицы 5.

Таблица 5. Правила преобразований с примерами порождения целевого кода.

| Наборы правил AST→Box | Пример результата работы fmt |
|---|---|
| `kernel = alt(`<br>`  rule(Id(X), to(lambda v: H(v.X))),`<br>`  rule(Int(X), to(lambda v: H(str(v.X)))),`<br>`  rule(Assign(X, Y),`<br>`  to(lambda v: H(v.X, "=", H(v.Y, ";", sep=""))),`<br>`  rule(Bop(let(O=id), X, Y),`<br>`  to(lambda v: H(v.X, v.O, v.Y))),`<br>`  rule(If(let(C=id), X), to(lambda v: V(`<br>`  H("if", H("(", v.C, ")", sep=""), "{"),`<br>`  V(*v.X, tab="\t"),`<br>`  "}"`<br>`  ))),`<br>`  rule(Func(X, [], Y), to(lambda v: V(`<br>`  H("void", H(v.X, "(void)", sep="")),`<br>`  "{",`<br>`  V(*v.Y, tab="\t"),`<br>`  "}"`<br>`  )))`<br>`)` | `void foo(void)`<br>`{`<br>`  if (x > 0) {`<br>`  x = 0;`<br>`  z = y + 1;`<br>`  }`<br>`  y = z;`<br>`}` |
| `python = alt(`<br>`  rule(Id(X), to(lambda v: H(v.X))),`<br>`  rule(Int(X), to(lambda v: H(str(v.X)))),`<br>`  rule(Assign(X, Y),`<br>`  to(lambda v: H(v.X, "=", v.Y))),`<br>`  rule(Bop(let(O=id), X, Y),`<br>`  to(lambda v: H(v.X, v.O, v.Y))),`<br>`  rule(If(let(C=id), X), to(lambda v: V(`<br>`  H("if", H(v.C, ":", sep="")),`<br>`  V(*v.X, tab=" " * 4)))),`<br>`  rule(Func(X, [], Y), to(lambda v: V(`<br>`  H("def", H(v.X, "():", sep="")),`<br>`  V(*v.Y, tab=" " * 4)`<br>`  )))`<br>`)` | `def foo():`<br>`  if x > 0:`<br>`  x = 0`<br>`  z = y + 1`<br>`  y = z` |

**Заключение**

В данной статье представлен обзор современных стандартов оформления кода. Для решения задачи автоматического форматирования кода в рамках программных проектов, использующих DSL, предложена библиотека на языке Python. Данная библиотека обладает следующими свойствами:

- выразительное описание преобразований из AST-представления программ в целевое представление на языках с различным синтаксисом и с использованием некоторого стандарта форматирования кода;

- интеграция с инструментарием raddsl для порождения DSL-компиляторов;

- простая реализация (код основной функций fmt занимает менее 20 строк).

Расширение функциональности данной библиотеки возможно в рамках выбранного формата описания правил. В частности, это касается возможности добавления в библиотеку учета ограничений на длину строки.

В качестве примеров использования разработанной библиотеки в статье приведены преобразования из AST-представления в представления с синтаксисом Си (стандарт оформления Linux Kernel) и Python (стандарт оформления PEP 8). Библиотека является проектом с открытым кодом и доступна по ссылке [12].

Список литературы


1. Linux kernel coding style [Электронный ресурс]. - Режим доступа: URL: https://www.kernel.org/doc/html/v4.10/process/coding-style.html (04.11.2018)
2. Google C++ Style Guide [Электронный ресурс]. - Режим доступа: URL: https://google.github.io/styleguide/cppguide.html (04.11.2018)
3. Google Java Style Guide [Электронный ресурс]. - Режим доступа: URL: https://google.github.io/styleguide/javaguide.html (04.11.2018)
4. PEP 8 -- Style Guide for Python Code [Электронный ресурс]. - Режим доступа: URL: https://www.python.org/dev/peps/pep-0008/ (04.11.2018)
5. Соглашения о написании кода на C# (Руководство по программированию на C#) [Электронный ресурс]. - Режим доступа: URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/inside-a-program/coding-conventions (04.11.2018)
6. Go Code Review Comments [Электронный ресурс]. - Режим доступа: URL: https://github.com/golang/go/wiki/CodeReviewComments (04.11.2018)
7. Coding Conventions [Электронный ресурс]. - Режим доступа: URL: https://kotlinlang.org/docs/reference/coding-conventions.html (04.11.2018)
8. Van den Brand M. G. J. et al. A language independent framework for context-sensitive formatting //Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on. – IEEE, 2006. – С. 10 pp.-112.
9. Kiselyov O., Peyton-Jones S., Sabry A. Lazy v. yield: Incremental, linear pretty-printing //Asian Symposium on Programming Languages and Systems. – Springer, Berlin, Heidelberg, 2012. – С. 190-206.
10. Parr T., Vinju J. Towards a universal code formatter through machine learning //Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering. – ACM, 2016. – С. 137-151.
11. Oppen D. C. Prettyprinting //ACM Transactions on Programming Languages and Systems (TOPLAS). – 1980. – Т. 2. – №. 4. – С. 465-483.
12. Tools for rapid prototyping of DSL compilers [Электронный ресурс]. - Режим доступа: URL: https://github.com/true-grue/raddsl (04.11.2018)
13. Советов П. Н., Тарасов И. Е. Разработка многопоточного софт-процессора со стековой архитектурой на основе совместной оптимизации программной модели и системной архитектуры //Многоядерные процессоры, параллельное программирование, ПЛИС, системы обработки сигналов. – 2017. – №. 7. – С. 8-19.


References


1. Linux kernel coding style. // URL: https://www.kernel.org/doc/html/v4.10/process/coding-style.html (04.11.2018)
2. Google C++ Style Guide. // URL: https://google.github.io/styleguide/cppguide.html (04.11.2018)
3. Google Java Style Guide. // URL: https://google.github.io/styleguide/javaguide.html (04.11.2018)
4. PEP 8 -- Style Guide for Python Code. // URL: https://www.python.org/dev/peps/pep-0008/ (04.11.2018)
5. C# Coding Conventions (C# Programming Guide) // URL: hhttps://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions (04.11.2018)
6. Go Code Review Comments. // URL: https://github.com/golang/go/wiki/CodeReviewComments (04.11.2018)
7. Coding Conventions. // URL: https://kotlinlang.org/docs/reference/coding-conventions.html (04.11.2018)
8. Van den Brand M. G. J. et al. A language independent framework for context-sensitive formatting //Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on. – IEEE, 2006. – С. 10 pp.-112.

9. Kiselyov O., Peyton-Jones S., Sabry A. Lazy v. yield: Incremental, linear pretty-printing //Asian Symposium on Programming Languages and Systems. – Springer, Berlin, Heidelberg, 2012. – С. 190-206.

10. Parr T., Vinju J. Towards a universal code formatter through machine learning //Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering. – ACM, 2016. – С. 137-151.

11. Oppen D. C. Prettyprinting //ACM Transactions on Programming Languages and Systems (TOPLAS). – 1980. – Т. 2. – №. 4. – С. 465-483.

12. Tools for rapid prototyping of DSL compilers. // URL: https://github.com/true-grue/raddsl (04.11.2018)

13. Sovetov P. N., Tarasov I. E. Razrabotka mnogopotochnogo soft-protsessora so stekovoj arkhitekturoj na osnove sovmestnoj optimizatsii programmnoj modeli i sistemnoj arkhitektury //Mnogoyadernye protsessory, parallel'noe programmirovanie, PLIS, sistemy obrabotki signalov. – 2017. – no. 7. – pp. 8-19.