Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
Высшего профессионального образования

Московский государственный технический университет имени Н.Э. Баумана

Расчетно-пояснительная записка
к дипломному проекту
НА ТЕМУ:

Проверка корректности расстановки отступов исходных текстов программ на языке C.

Студент группы ИУ9-82
_____ Разборщикова Анастасия Викторовна
«___» _____ 2019 г.
Преподаватель
_____ Скоробогатов Сергей Юрьевич
«___» _____ 2019 г.

Москва
2018

Оглавление

Введение

При разработке программ необходимо уделять внимание не только логическому проектированию программы и ее коррекности, но и стилю оформления кода (coding style). Этой проблеме посвящаются книги и научные статьи. Наименование переменых, длина строк, использование шаблонов (patterns) проектирования, — все это напрямую влияет на понимание программы, возможность распространять ее и избежать проблем с дальнейшей поддержкой, ведь большую часть вренеми при разработке занимает именно чтение кода [1]. Были проведены исследования, доказывающие, что форматирование (стиль расстановки скобок, пробелы и т.п.) влияет на восприятие программы [2].

На сегодняшний день разработано множество стандартов оформления кода, для многих крупных проектов разрабатываются собственные соглашения (например [3], [4]). Однако студенты, только начавшие изучать программирование, часто не знакомы с ними, либо пренебрегают этими правилами и не стремятся к аккуратному оформлению кода.

На кафедре ИУ9 «Теоретическая информатика и компьютерные технологии» факультета «Информатики и систем управления» МГТУ им. Н. Э. Баумана для проверки работ студентов используется автоматическая система тестирования. После проверки программы на корректность и заимствования, она попадает на проверку преподавателю. На этом этапе встает вопрос о читаемости кода, адекватности его оформления. В связи с этим возникла необходимость разработки инструмента, который бы также проверял код на соответствие некоторым правилам форматирования и выдавал студентам сообщения об ошибках, фильтруя поток так, чтобы преподаватель мог проверять только корректно отформатированный код. Сложность задачи состоит в том, чтобы не делать условия корректности слишком жесткими (многие среды разработки программ автоматически форматируют код в соответствии со своими стандартами, которые могут отличаться, при этом являясь корректными) и поддерживать все возможные расширения компилятора GCC (the GNU Compiler Collection), установленного на сервере тестирования.

1 Постановка задачи

В рамках выпускной квалификационной работы ставится задача реализовать программу для проверки корректности расстановки отступов в программах на языке C.

Приложение должно выявлять *плохо отформатированные* программы и выдавать описание найденных ошибок.

***Определение* 1.** *В рамках данной задачи исходный код программы считается плохо отформатированным, если:*

1. *На одном уровне вложенности строки имеют разный отступ (за исключением строк, которые являются продолжением выражения, а также меток, включая метки* `case` *и* `default` *в теле оператора* `switch`*).*
2. *Строки кода на бóльшем уровне вложенности располагаются левее кода на меньшем уровне вложенности. Это правило относится к не относится к меткам (за исключением меток* `case` *и* `default`*, которые должны быть не левее объявления блока* `switch`*) и директивам препроцессора, так как они могут не иметь отступа.*
3. *Строка, являющаяся продолжением выражения (перенос), располагается не левее предыдущей строки.*

Основные требования к программе:

1. Программа должна быть реализована в виде консольного приложения, которой подается на вход исходный код программы на языке C.
2. Программа должна выдавать сообщения об обнаруженных ошибках расстановки отступов.
3. Должна быть реализована возможность управлять тем, насколько подробно выводятся сообщения (т.е. поддерживать различные режимы диагностики).
4. Программа не должна реализовывать слишком жесткие требования к оформлению кода.
5. Программа должна запускаться на платформе GNU/Linux.
6. Должна быть реализована поддержка использования любых расширений компилятора GCC, которые могут использоваться в исходных кодах программ.

Пример, иллюстрирующий различия между кодом с допустимым форматированием и плохо отформатированным, приведен на рисунке 1.

```
1  void ok(void) {
2      int x = 1,
3      n = 5, i;
4      for(i = 0;
5          i < n; i++) {
6        x += x;
7      }
8      if (x % 2 == 0
9           || x > 0) {
10       printf("Ok!\n");
11     }
12 }
13
```

а) Допустимое форматирование.

```
1  void bad(void) {
2      int x = 1,
3    n = 5, i;
4      for(i = 0;
5          i < n; i++) {
6        x += x;
7      }
8        if (x % 2 == 0
9             || x > 0) {
10     printf("Bad!\n");
11      }
12 }
13
```

б) *Плохое* форматирование.

Рисунок 1 — Примеры кода с допустим и плохим форматированием.

2 Руководство пользователя

2.1 Системные требования

Программа распространяется в виде исходного кода на платформе GitHub [5]. Для запуска необходимо иметь следующее ПО:

1. ОС GNU/Linux,
2. Компилятор GCC версии не ниже 7 (либо другой компилятор, поддерживающий стандарт C++17),
3. CMake версии от 3.10.2.

2.2 Установка и запуск

В папке с исходным кодом `c-format-checker` находится скрипт `install.bash`, запускающий сборку проекта. Из папки `c-format-checker` в терминале необходимо выполнить следующие команды (листинг 1):

Листинг 1 — Сборка проекта.

```
$ chmod +x install.bash  % Сделать файл исполняемым.
$ ./install.bash         % Запустить сборку.
```

В результате выполнения скрипта в папке появится директории `build` (содержит временные файлы `CMake`) и `bin`. В папке `bin` находится исполняемый файл `c-format-checker`.

Это консольное приложение. Оно имеет следующие параметры вызова (листинг 2):

Листинг 2 — Формат вызова приложения.

```
$ ./c–format–checker [–f] ФАЙЛ [–q | –v | –d] [–l ФАЙЛ]
```

Программе подается на вход один обязательный аргумент: имя файла для анализа и несколько необязательных параметров:

-f, - -file Файл исходного текста программы на языке C (ключ `-f` не требуется, если этот аргумент идет первым).

-q, - -quiet Выводить кратко описание ошибок.

-v, - -verbose Выводить развернутые описания ошибок (по умолчанию).

-d, - -debug Выводить отладочную информацию.

- -help Помощь.

-l, - -logfile Имя файла для записи отладочной информации (по умолчанию: `c-format-checker.log`).

Ключи -q, -v, -d, указывающие, насколько подробным должен быть вывод приложения, не могут встречаться в списке аргументов более одного.

2.2.1. Скрипт тестирования

В каталоге проекта размещен также скрипт для пакетного тестирования `testing_script.bash`. На вход скрипт получает два аргумента: путь к расположению программы (напр. `/bin/c-format-checker`) и путь к папке, содержащей текстовые файлы с программами на языке C.

Каждый файл из указанной папки обрабатывается программой. Логи выполнения записываются в файл `out/<имя файла>.log`. Если для какого-то файла программа вернула код 5 (ошибка на этапе лексического или синтаксического разбора), имя этого файла вносится в файл `parse_errors.log`. Если код 1 — имя обработанного файла записывается в файл `conclusion.txt`. Файлы, для которых программа вернула код 0 (ошибок нет) или 2 (обнаружено использование табуляций в отступах), игнорируются.

В результате выполнения скрипта появится папка `out`, содержащая отчеты о выполнении программы для всех файлов, а также, в случае обнаружения соответствующих ошибок, появятся документы `conclusion.txt` и `parse_errors.log`, со списками файлов, в которых были обнаружены ошибки.

2.3 Вывод приложения и сообщения об ошибках

Программа выделяет два вида ошибок при расстановке отступов:

1. Строка имеет иной отступ, чем строки выше на том же уровне вложенности.
2. Использование табуляций (так как нет единого стандарта о ширине символа при отображении).

Если программа обнаружила только ошибки первого вида, выдается соответствующее сообщение (в зависимости от режима вывода) и программа завершается с кодом 1.

Если в процессе анализа найдена ошибка второго вида, анализ прекращается, выдается соответствующее сообщение и программа завершается с кодом 2.

В режиме `quiet` в стандартный поток выводится краткая сводка об обнаруженных некорректных отступах, а также сообщения об ошибках. Об ошибках отступов сообщается в следующем формате: «`[Анализ отступов] строка <номер_строки>: некорректный отступ.`» Пример вывода приложения приведен в листинге 3.

Листинг 3 — Запуск приложения в режиме `quiet`.

```
> ./c-format-checker ../tests/test.c -q
Использование: ./c-format-checker [-f] ФАЙЛ [-q | -v | -d] [-l ФАЙЛ].

Запустите «./c-format-checker --help» для более подробного описания.

Файл для вывода лога не указан, использован файл по умолчанию
«c-format-checker.log».

[Анализ отступов] строка 13: некорректный отступ.
[Анализ отступов] строка 24: некорректный отступ.

Process finished with exit code 1
```

Если ошибок не обнаружено, программа ничего не выводит и завершается с кодом 0.

Режим `verbose` является форматом вывода по умолчанию. В этом режиме в стандартный поток выводятся развернутые сообщения о некорректных отступах и сообщения об ошибках. Сообщения имеют следующий вид: «`[Анализ отступов] строка <номер_строки>: ошибка: отступ ширины <число>: <образ>. <подсказка>`». Образ — печатное представление символов отступа: «␣» для пробела и «`\t`» для табуляции. При вычислении величины отступа ширина табуляции в программе берется равной одному пробелу. Подсказка — это сообщение, описывающее ошибку. Если в отступе в начале строки встретился символ табуляции, выдается сообщение «Использование пробелов и табуляций в одном отступе» или «Использование табуляций (ранее использовались пробелы)». Если ошибка связана с тем, что какая-то строка имеет отступ, не равный таковому для строки выше на том же уровне вложенности, под-

сказка имеет вид: «Ранее на том же уровне вложенности в строке <номер сроки выше> отступ ширины <число>: <образ отступа>.» Если при переносе выражения на следующую строку продолжение оказалось левее начала выражения, дополнительно выводится сообщение «Продолжение выражения левее предыдущей строки.» Пример вывода приложения в режиме verbose приведен в листинге 4.

Листинг 4 — Запуск приложения в режиме verbose.

```
> ./c-format-checker ../tests/test.c -v

Использование: ./c-format-checker [-f] ФАЙЛ [-q | -v | -d] [-l ФАЙЛ].

Запустите «./c-format-checker --help» для более подробного описания.

Файл для вывода лога не указан, использован файл по умолчанию
«c-format-checker.log».

[Анализ отступов] строка 13: ошибка: отступ ширины 5: <␣␣␣␣␣>.
Продолжение выражения левее предыдущей строки.
[Анализ отступов] строка 24: ошибка: отступ ширины 4: <␣␣␣␣>.
Ранее на том же уровне вложенности  в строке 23 отступ ширины 8:
<␣␣␣␣␣␣␣␣>.

Process finished with exit code 1
```

В режиме debug в стандартный поток вывода выводится вся отладочная информация.

Эти данные содержат список токенов и вывод правил грамматики, полученные в результате лексического и синтаксического разбора, а также промежуточные таблицы анализатора, содержащие уровень вложенности, на котором находится каждая строка файла. Во всех режимах эти данные записываются в лог-файл. Сообщения о найденных ошибках отступов выводятся в таком же формате, как в режиме verbose.

2.4 Критерии корректности форматирования кода

Программа считается корректной, если на одном уровне вложенности все строки имеют одинаковый отступ. Программа не должна содержать символов табуляции в начале строки (при этом не накладывается никаких ограничений на их использование в середине или конце строки).

Следующим уровнем вложенности считаются:

1. Блок.
2. Тело функции.
3. Тело конструкций условных переходов `if`, `else`, `switch`.
4. Тело цикла `for`, `while`, `do`.
5. Пользовательская метка, метки `case` и `default`.
6. Блок кода, следующий за метками `case` и `default` внутри тела `switch`.
7. Тело объявления структуры `struct`, объединения `union` и перечисления `enum`.

Не считаются следующим уровнем вложенности:

1. Список инициализации.
2. Перенос выражения на следующую строку.
3. Выражения в скобках.
4. Условные выражения в конструкциях `if`, `switch`, `for`, `while`.

Случаи, в которых ослабляется требование равенства всех отступов на одном уровне вложенности:

1. Метки могут располагаться левее или правее основного блока кода.
2. При переносе выражения на другую строку ослабляется требование равенства отступов. Перенос считается корректным, если новая строка начинается не левее предыдущей.
3. Внутри блока `switch` метки `case` и `default` могут располагаться левее прочего кода в том же блоке, но на одном уровне друг с другом и не левее самого оператора `switch`.
4. Внутри блока `switch` код, который предшествует самой первой метке `case`, имеет собственный уровень вложенности и не обязан совпадать по ширине отступа с метками или последовательностью кода после них.

Примеры корректого и некорректного форматирования тела оператора `switch` приведены на рисунке 2.

```
switch(x)
{
    int i=4;
    f(i);
case 0:
    i=17;
default:
    i=2;
}
```

а) Корректно (стиль из стандарта ISO/IEC9899:2017 [6, с. 109])

```
switch(x)
{
    int i=4;
    f(i);
    case 0:
        i=17;
    default:
        i=2;
}
```

б) Корректно (форматирование среды разработки CLion)

```
switch(x)
{
    int i=4;
f(i);
    case 0:
          i=17;
    default:
        i=2;
}
```

в) Некорректно

Рисунок 2 — Пример корректного (а, б) и некорректного (в) форматирования тела оператора `switch`.

Заключение

Список литературы

[1] Lee T., Lee J. B., In H. P. A Study of Different Coding Styles Affecting Code Readability // International Journal of Software Engineering and Its Applications. Gothenburg, Sweden, 2013. Vol. 7, no. 5. P. 413–422. Access: https://pdfs.semanticscholar.org/1727/4fef400424fe4876cd23edb8318e4944b203.pdf.

[2] Dos Santos R. M., Gerosa M. A. Impacts of Coding Practices on Readability // ICPC'18: 26th IEEE/ACM International Conference on Program Comprehension. Gothenburg, Sweden, 2018. 9 p. Access: https://www.ime.usp.br/ gerosa/papers/ICPC2018-Legibility.pdf.

[3] Google Style Guides [Электронный ресурс]. Режим доступа: http://google.github.io/styleguide/.

[4] Qt Coding Style [Электронный ресурс]. URL: https://wiki.qt.io/Qt_Coding_Style (дата обращения: 30.05.2019).

[5] Исходный код приложения c-format-checker [Электронный ресурс]. Режим доступа: https://github.com/anastasiarazb/c-format-checker.

[6] INTERNATIONAL STANDARD. Programming languages – C. ISO/IEC9899:2017 [Электронный ресурс]. Режим доступа: https://web.archive.org/web/20181230041359if_/http://www.open-std.org/jtc1/sc22/wg14/www/abq/c17_updated_proposed_fdis.pdf. (дата обращения: 2.06.2019).