# Проверка корректности расстановки отступов в программе на языке C

Студент: Разборщикова А В.
Руководитель: Скоробогатов С. Ю.

Москва, 2019

# Постановка задачи

Необходимо реализовать компонент системы автоматической системы тестирования, выполняющую проверку корректности расстановки отступов в программе на языке C.

**Цель:** уменьшить нагрузку на проверяющего, допуская к проверке только хорошо отформатированные и легко читаемые программы.

**Входные данные:** один или несколько файлов исходного кода.

**Выходные данные:** список сообщений об ошибках форматирования.

# Когда форматирование влияет на понимание кода

**Распространенная ошибка:** при отсутствии фигурных скобок неочевидно, к какому оператору относится выражение.

```
1  int main(){
2
3  int a = 1;       // Ошибка: отступ отличается на одном и том же
4      if (a ==2)   // уровне вложенности
5      {
6   return 2;       // Ошибка: вложенное выражение расположено левее
7      }            // объемлющего
8      if (1)
9          if (2) return 2;  // Ошибка: ветвь else относится
10     else                 // к вложенному оператору if
11         return 3;
12     return 0;
13 }
```

Пример файла, в котором ошибки форматирования могут приводить к ошибочному пониманию алгоритма.

# Критерии некорректного форматирования

Отступ в строке считается *некорректным*, если:

- он отличается от предыдущего отступа на том же уровне вложенности;
- при переносе инструкции новая строка начинается левее предыдущей;
- строка начинается левее, чем объемлющий блок (например, код в блоке оператора левее фигурных скобок и т.п.).

# Пример некорректного форматирования

```
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 }
```

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 }
```

Некорректное форматирование.

# Обзор существующих решений

Инструменты автоформатирования кода (clang-format и др.):

- выдают заново отформатированный код, а не сообщения об ошибках $\rightarrow$ не подходят для обучения

Линтеры — инструменты проверки стиля оформления кода (Cpplint, Splint, clang-tidy):

- проверяют код на соответствие стилям форматирования, но в них редко учитывается взаимное расположение строк
- ни один из проверенных инструментов не выявил ошибок в тестовых примерах $\rightarrow$ не подходят для использования

Статические анализаторы (Clang Static Analyzer, Cppcheck)

- выполняют поиск возможных ошибок времени выполнения
- могут выдавать сообщения об ошибках форматирования

# Обзор существующих решений (продолжение)

Таблица: Сравнение рассмотренных статических синтаксических анализаторов на предмет выявления ошибок в тестовом файле `dummy.c`

| Анализатор | Разные отступы в блоке | Несоответствие отступа уровню вложенности | Разный отступ ветвей оператора `if-else` |
|---|---|---|---|
| clang-tidy | — | — | Предупреждение об отсутствии скобок. Указание на двусмысленность выражения. |
| Clang Static Analyzer | — | — | — |
| cpplint | — | Предупреждение о «странном» отступе. Причина неясна. | Сообщение об ошибке отступа. Напоминание о скобках. |
| Cppcheck | — | — | — |
| Splint | — | — | Предупреждение об отсутствии скобок. |

# Методы анализа текста. Пример: классическая схема компиляции

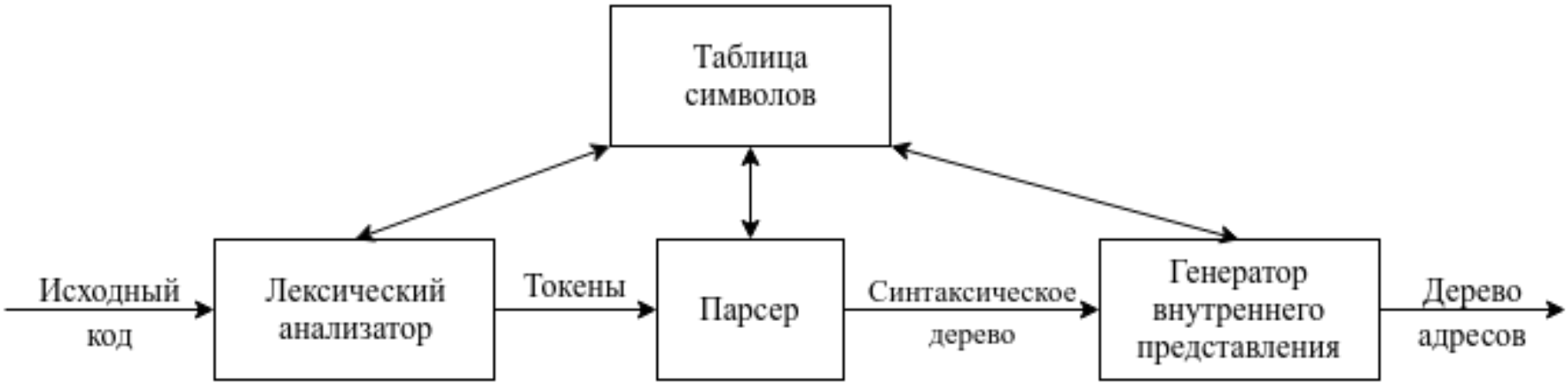
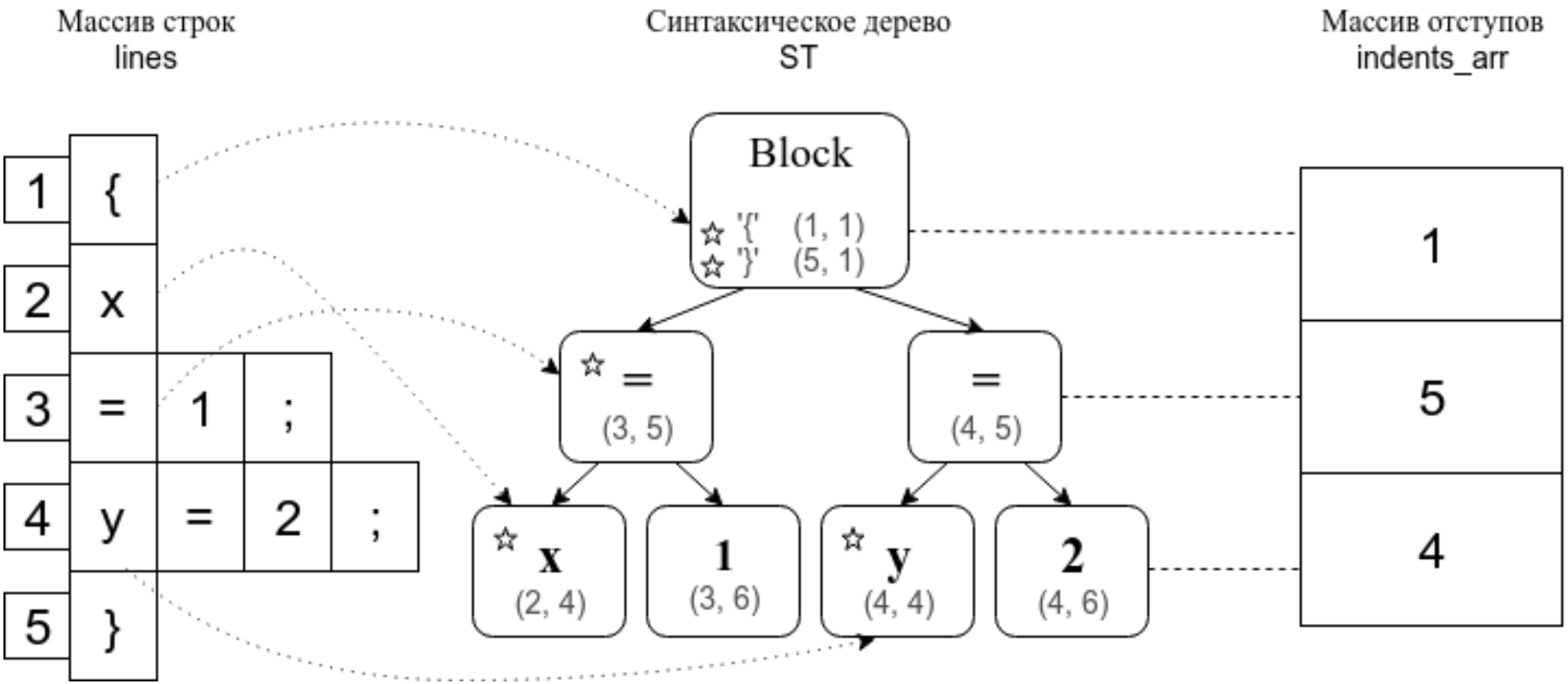



Схема стадии анализа в процессе компиляции

(Compilers: principles, techniques, and tools. Second Edition / A.V. Aho, M.S. Lam, etc.)

- Из классической схемы компиляции возьмем стадии ликсического и синтаксического анализа.

# Разработка алгоритма: первый этап

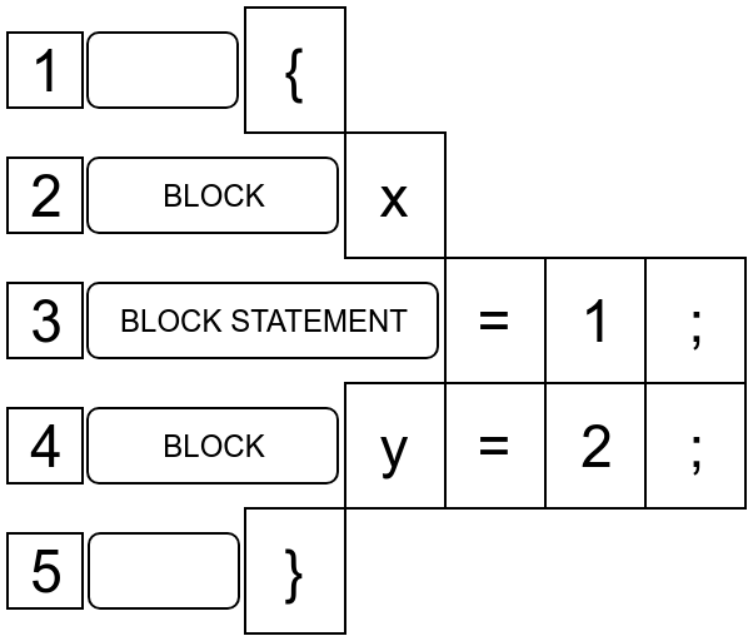



Структуры данных для первой реализации алгоритма.

- Слишком сложная синтаксическая структура усложняет анализ.
- Скомбинируем синтаксическое дерево и массив строк.

# Разработка алгоритма: окончательный вариант

- В окончательном варианте парсер строит не синтаксическое дерево, а таблицу строк с сохранением уровня вложенности в виде вектора состояний (*метасостояние*).




Структура данных, используемая для анализа.

# Алгоритм анализа отступов

1. Проверка корректности отступов относительно уровней вложенности:
   - проверяем, что у строк с большим числом элементов в метасостоянии отступ не меньше, чем у строк с меньшим метасостоянием.
2. Проверка равенства величины отступа на одном уровне вложенности:
   - сравниваем величину отступа для строк с одним метасостоянием.
3. Проверка корректности переносов:
   - сравниваем строки, последнее состояние в которых равно `STATEMENT` с предыдущей строкой.

# Повышаем абстракцию грамматики

Цель:

- поддержка нестандартных расширений языка;
- остановка синтаксического разбора на уровне инструкции (выражение, оканчивающееся точкой с запятой, условный оператор, цикл и т.п.): избавление от лишних сущностей, не влияющих на результат анализа;
- поддержка различных стилей форматирования и их комбинаций (а не одного стандарта, как реализовано в существующих анализаторах).

# Новая грамматика

- 70 правил $\longrightarrow$ 12 правил.
- Все, что является простым выражением представляется как набор слов `WORD-SEQUENCE`.

Таблица: Пример преобразования правила(жирным шрифтом выделен замененный нетерминал, который будет разобран как простая последовательность слов).

| Исходная грамматика | Преобразованная грамматика |
|---|---|
| `selection-statement:`<br>`  if (`**`expression`**`) statement`<br>`  if (`**`expression`**`) statement`<br>`  else statement`<br>`  switch (`**`expression`**`)`<br>`  statement` | `SELECTION-STATEMENT =`<br>`'if' '(' `**`WORD-SEQUENCE`**` ')'`<br>`  STATEMENT ('else' STATEMENT)?`<br>`\|`<br>`'switch' '(' `**`WORD-SEQUENCE`**` ')'`<br>`  STATEMENT` |

# Использование

- Программа реализована на языке C++ в виде консольного приложения
- Формат запуска приложения:

  `c-format-checker [-f] ФАЙЛЫ [-q | -v | -d]`

  (Ключи `-q`, `-v`, `-d` обозначают уровень детализированности сообщений об ошибках: `quiet` — краткая сводка, `verbose` — подробные сообщения, `debug` — включить отладочный вывод)

- Имена файлов, в которых была найдена ошибка, записываются в файл `cfc-statistics.txt`. Если в файле обнаружена ошибка табуляции, рядом с именем ставится пометка `(tab.)`.

# Сообщения об ошибках

Таблица: Сопоставление сообщений об ошибках в режимах `quiet` и `verbose`.

| Режим `quiet` | Режим `verbose` |
|---|---|
| ”Вложенное выражение левее объемлющего.” | ”Вложенное выражение левее объемлющего.” |
| ”Ошибка отступа при переносе выражения.” | ”Продолжение выражения левее предыдущей строки.” |
| ”Использование табуляции.” | ”Использование пробелов и табуляций в одном отступе.” |
| | ”Использование табуляций (ранее использовались пробелы).” |
| ”Различные отступы на одном уровне вложенности.” | Ранее на том же уровне вложенности в строке <номер сроки выше> отступ ширины <число>: <образ отступа>. |

# Тестирование

Тестирование проводилось в несколько этапов:

1. Тестирование на искусственно созданных простых примерах в режиме `debug`.
2. Тестирование на большом объеме реальных данных.
3. Анализ программы сторонними инструментами на предмет ошибок и утечек памяти.

Программа была проанализирована сторонними инструментами:

- Clang Static Analyzer: показал наличие одной неиспользуемой переменной;
- clang-tidy, cppcheck, valgrind: не выявили проблем или утечек памяти;
- SonarQube: 0 ошибок , уровень безопасности A (уязвимостей не обнаружено), 0% дублирования кода.

# Тестирование на реальных данных

Кафедрой были предоставлены следующие архивы с решениями студентов. Время замерялось утилитой командной строки `time`.

Таблица: Результат тестирования программы на реальных примерах.

| Задача | Файлов всего | Обнаружено файлов с ошибкой | Из них с ошибкой использования табуляции | Время выполнения, с |
|---|---|---|---|---|
| 011 Подсчёт слов в строке | 989 | 619 | 453 | 0,342 |
| 091 Фибоначчиевы строки | 353 | 207 | 136 | 0,284 |
| 094 Наибольший простой делитель | 397 | 260 | 160 | 0,328 |
| 095 Кратчайшая суперстрока | 939 | 706 | 633 | 1,208 |
| 148 Полином | 436 | 230 | 157 | 0,254 |

# Заключение

Целью данной работы было решить задачу автоматического анализа корректности расстановки отступов в программах на языке C.

Реализованная программа позволяет:

- автоматически отклонять плохо отформатированные решения, облегчая работу преподавателя;
- настраивать режимы вывода сообщений об ошибках;
- собирать статистику об обнаруженных ошибках;

Анализатор допускает большое количество стилей форматирования, а также их комбинаций.