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.

Критерии некорректного форматирования

Отступ в строке считается *некорректным*, если:

- он отличается от предыдущего отступа на том же уровне вложенности;

```
1  int x = 1;
2    int y = 2;
```

- при переносе инструкции новая строка начинается левее предыдущей;

```
1   int x
2  = 1;
```

- вложенная инструкция начинается левее, чем объемлющий блок.

```
1   if(x) {
2  z=2;
3  }
```

Когда форматирование влияет на понимание кода

- **Распространенная ошибка:** при отсутствии фигурных скобок неочевидно, к какому оператору относится следующая инструкция.

```
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 }
```

Обзор существующих решений

Автоформаттеры:

- [×] не показывают сообщения об ошибках → *не подходят для обучения*

Линтеры — инструменты проверки стиля оформления кода:

- ▷ редко учитывают взаимное расположение строк
- [×] не выявили ошибок в тестовых примерах → *не подходят*

Статические анализаторы — выполняют поиск логических ошибок:

- ▷ могут выдавать сообщения об ошибках форматирования
- [×] не выявили ошибок в тестовых примерах → *не подходят*

Обзор существующих решений (продолжение)

Анализатор	1. Разные отступы в блоке	2. Несоответствие отступа уровню вложенности	3. Разный отступ ветвей оператора `if-else`
Clang-Tidy	—	—	Предупреждение об отсут- ствии 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.
Реализованный анализатор	*Сообщение об ошибке. Указание строк с различиями.*	*Описание ошибки.*	*Сообщение об ошибке. Указание строк с различиями.*

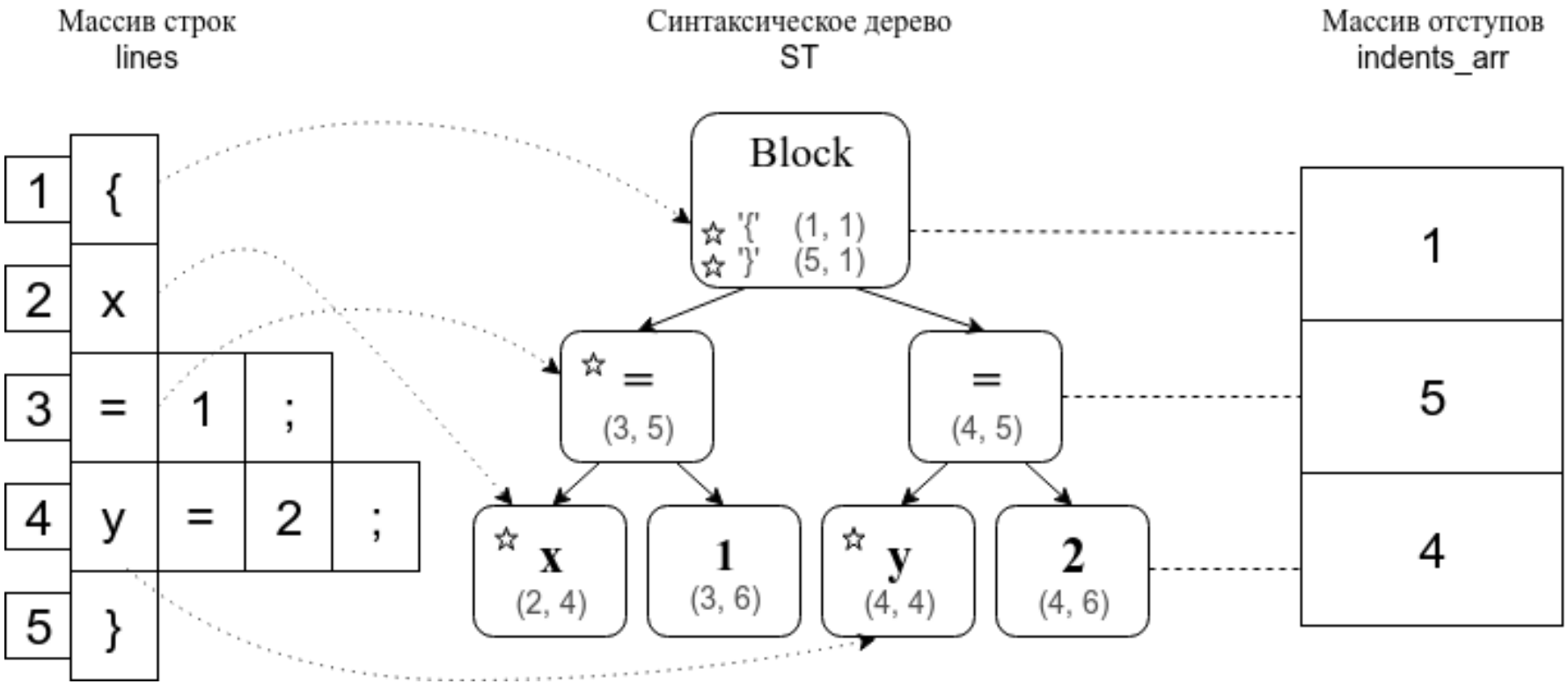
Методы анализа текста. Пример: классическая схема компиляции

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

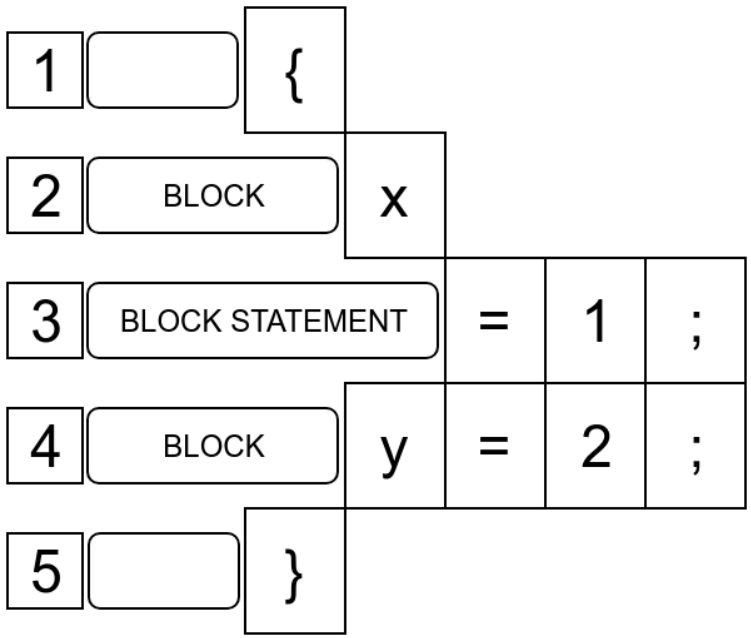


Структуры данных для первой реализации алгоритма.

- 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 при взаимной вложенности** («блок не левее `if`»)
   - проверяем, что у строк с большим числом элементов в метасостоянии отступ не меньше, чем у строк с меньшим метасостоянием.

2. **Проверка равенства отступов на одном уровне вложенности**
   - сравниваем величину отступа для строк с одним метасостоянием.

3. **Проверка корректности переносов**
   - сравниваем с предыдущей строки, последнее состояние в которых равно `STATEMENT`.

Повышение уровня абстракции грамматики

Удаляем из грамматики языка C правила ниже уровня инструкций.

```
declaration-specifiers:
    storage-class-specifier declaration-specifiersopt
    type-specifier declaration-specifiersopt
    type-qualifier declaration-specifiersopt
    function-specifier declaration-specifiersopt
    alignment-specifier declaration-specifiersopt
```

$\longrightarrow$ `WORD-SEQUENCE = WORD*`

Цель:

- поддержка нестандартных расширений языка;
- избавление от лишних сущностей, не влияющих на результат анализа;
- поддержка различных стилей форматирования и их комбинаций.

Новая грамматика

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

Таблица: Пример преобразования правила (жирным шрифтом выделено упрощаемое правило).

Исходная грамматика	Преобразованная грамматика

`selection-statement:`
`  if (`**`expression`**`) statement`
`  if (`**`expression`**`) statement`
`  else statement`
`  switch (`**`expression`**`)`
`  statement`
	`SELECTION-STATEMENT =`
	`'if' '(' `**`WORD-SEQUENCE`**` ')'`
	`  STATEMENT ('else' STATEMENT)?`
	`|`
	`'switch' '(' `**`WORD-SEQUENCE`**` ')'`
	`  STATEMENT`

Использование приложения

- Формат запуска приложения из консоли:

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

`-q`, `--quiet` — краткая сводка,
`-v`, `--verbose` — подробные сообщения,
`-d`, `--debug` — включить отладочный вывод.

- Статистика записывается в файл:

`cfc-statistics.txt`

```
...
src/filename1.c
src/filename2.c   (tab.)
...
```

← имя файла с ошибкой отступа
← файл содержит табуляцию

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;
2. тестирование на большом объеме реальных данных;
3. анализ программы сторонними инструментами на предмет ошибок и утечек памяти.

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.

Планируется интеграция приложения в систему автоматического тестирования кафедры.