

**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)**

**ОТЧЕТ**

По курсовому проекту

По курсу «Конструирование компиляторов»

Тема: «Компилятор языка apalang подмножества языка программирования Ruby»

Студент:

Апальков Ф.С.

Группа:

ИУ7-23М

Преподаватель:

Ступников А.А.

Москва 2022

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой \_\_\_\_\_\_\_\_\_\_

(Индекс)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_И.В.Рудаков\_\_\_\_

(И.О.Фамилия)

« \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_\_ г.

**ЗАДАНИЕ**

**на выполнение курсового проекта**

по дисциплине \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Конструирование компиляторов\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент группы ИУ7-23М\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Апальков Фёдор Станиславович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Тема курсовой работы Компилятор языка apalang подмножества языка программирования Ruby

Направленность КР: учебная\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Источник тематики: кафедра\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Задание*** \_\_Разработать компилятор языка apalang используя кодогенерацию в Javascript для запуска в браузере\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Оформление курсового проекта:***

Расчетно-пояснительная записка на 20-30 листах формата А4.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Расчетно-пояснительная записка должна содержать постановку, введение, полное описание всех стадий компиляции, заключение, список использованной литературы,\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Перечень графического (иллюстративного) материала\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания « \_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

**Руководитель курсового проекта**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_Ступников А.А.

(Подпись, дата) (И.О.Фамилия)

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_**Апальков Ф.С.

(Подпись, дата) (И.О.Фамилия)

Оглавление

[Постановка задачи 4](#_Toc106746163)

[Введение 4](#_Toc106746164)

[Грамматика языка apalang 6](#_Toc106746165)

[Создание парсера 14](#_Toc106746166)

[Создание дерева разбора 16](#_Toc106746167)

[Кодогенерация 18](#_Toc106746168)

[Создание исполняемого файла 20](#_Toc106746169)

[Тестирование и запуск 20](#_Toc106746170)

[Список использованной литературы 23](#_Toc106746171)

# Постановка задачи

Разработать компилятор языка apalang подмножества языка программирования Ruby.

Описать грамматику языка.

Создать Парсер файла в токены языка apalang.

Создать по полученному списку токенов дерево разбора.

По дереву разбора сгенерировать JavaScript код.

Встроить полученный JavaScript код в html разметку.

Создать исполняемый файл, запускающий браузер по умолчанию с параметром автоматически загружаемой страницы, полученной на предыдущем этапе.

# Введение

Грамматика языка apalang (описанная в бнф формате далее) является КСГ. В грамматике отсутствует левая рекурсия, что позволяет создать дерево ее разбора нисходящим LL анализом. Для правил в которых присутствует возможность выбора разных вариантов, алгоритм будет строить все возможные деревья сохраняя процесс, если возникнет ошибка в разборе алгоритм вернется к предыдущему состоянию и попробует сделать разбор по другим вариантам правил.

Контекстно-свободная грамматика (КС-грамматика, бесконтекстная грамматика) — частный случай формальной грамматики (тип 2 по иерархии Хомского), у которой левые части всех продукций являются одиночными нетерминалами (объектами, обозначающими какую-либо сущность языка (например, формула, арифметическое выражение, команда) и не имеющими конкретного символьного значения). Смысл термина «контекстно-свободная» заключается в том, что есть возможность применить продукцию к нетерминалу, причём независимо от контекста этого нетерминала (в отличие от общего случая неограниченной грамматики Хомского).

Язык, который может быть задан КС-грамматикой, называется контекстно-свободным языком или КС-языком.

По сути КС-грамматика — другая форма БНФ. [1]

Синтаксический LL-анализатор (LL parser) — в информатике нисходящий синтаксический анализатор для некоторого подмножества контекстно-свободных грамматик, известных как LL-грамматики. При этом не все контекстно-свободные грамматики являются LL-грамматиками.

Буквы L в выражении «LL-анализатор» означают, что входная строка анализируется слева направо (left to right), и при этом строится её левосторонний вывод (leftmost derivation).

LL-анализатор называется LL(k)-анализатором, если данный анализатор использует предпросмотр на k токенов (лексем) при разборе входного потока. Грамматика, которая может быть распознана LL(k)-анализатором без возвратов к предыдущим символам, называется LL(k)-грамматикой. Язык, который может быть представлен в виде LL(k)-грамматики, называется LL(k)o-языком. Существуют LL(k+n)-языки, которые не являются LL(k)-языками.

Далее описывается анализатор, в основе которого лежит построение таблиц; альтернативой может служить анализатор, построенный методом рекурсивного спуска, который обычно пишется вручную (хотя существуют и исключения, например, генератор синтаксических анализаторов ANTLR для LL(\*) грамматик).

LL(1)-грамматики очень распространены, потому что соответствующие им LL-анализаторы просматривают поток только на один символ вперед при принятии решения о том, какое правило грамматики необходимо применить. Языки, основанные на грамматиках с большим значением k, традиционно считались трудными для распознавания, хотя при широком распространении генераторов синтаксических анализаторов, поддерживающих LL(k) грамматики с произвольным k, это замечание уже неактуально.

LL-анализатор называется LL(\*)-анализатором, если нет строгого ограничения для k и анализатор может распознавать язык, если токены принадлежат какому-либо регулярному множеству (например, используя детерминированные конечные автоматы).

Существуют противоречия между так называемой «Европейской школой» построения языков, которая основывается на LL-грамматиках, и «Американской школой», которая предпочитает LR-грамматики. Такие противоречия обусловлены традициями преподавания и деталями описания различных методов и инструментов в конкретных учебниках; кроме того, своё влияние оказал Н. Вирт из ETHZ, чьи исследования описывают различные методы оптимизации LL(1) распознавателей и компиляторов. [2]

# Грамматика языка apalang

Ниже описана грамматика языка apalang.

Terminals:

+ - \* / == < > <= >= !=

. ) ( = ; : , ' " [

] пока выполнять все стоп функция вернуть если иначе для

до внутри шаг целое дробное строка список начало конец получить

вставить продолжить ввод вывод 0 1 2 3 4 5

6 7 8 9 П А Б В Г Д

Е Ё Ж З И Й К Л М Н

О Р С Т У Ф Х Ц Ч Ш

Щ Ъ Ы Ь Э Ю Я \_

Nonterminals:

Программа СписокБлоков Блок Выражение Функция Пока Для Если Стоп Определение

Объявление Присваивание Лево равно т ИМЯ Право Подсчет Вызов СписокБукв

Тип Цел Дроб Строка Список двточ ПОКА ВЫПОЛН ВСЕ СТОП

ФУНКЦИЯ Лскб СписПарам ВЕРНУТЬ Пскб Запятая СписПарамФ ЕСЛИ ИНАЧЕ ДЛЯ

Число ДО ШАГ ВНУТРИ Кавыч Буква ДвКавыч ОпределениеСписка ЛскбКВ ПскбКВ

СписокЭлементовСписка ЭлементСписка Выр множить Авыр сложить СписокСимволов Симв Цифра точка

ЦЕЛ ДРОБ СТРОКА СПИСОК НАЧАЛО КОНЕЦ Строчка СписокЦифр ПОЛУЧИТЬ Получить

Вставить Встав сравнение Продолжить ПРОДОЛЖИТЬ ВВОД ВЫВОД Ввод Вывод ВнутриСписка

ВсеСимволы

Start: Программа

Rules:

Программа -> <НАЧАЛО><СписокБлоков><КОНЕЦ>

НАЧАЛО -> <начало>

КОНЕЦ -> <конец>

СписокБлоков -> <Блок><СписокБлоков>

СписокБлоков -> <Блок>

Блок -> <Функция>

Блок -> <Пока>

Блок -> <Для>

Блок -> <Если>

Блок -> <Вызов><т>

Блок -> <Выражение><т>

Блок -> <Продолжить>

Блок -> <Стоп>

Блок -> <Ввод><т>

Блок -> <Вывод><т>

Блок -> <Вставить><т>

Блок -> <ВЕРНУТЬ><ИМЯ><т>

Вставить -> <Встав><Право><ВНУТРИ><ИМЯ>

Встав -> <вставить>

Ввод -> <ВВОД><ИМЯ>

Вывод -> <ВЫВОД><ИМЯ>

ВВОД -> <ввод>

ВЫВОД -> <вывод>

Выражение -> <Определение>

Выражение -> <Объявление>

Выражение -> <Присваивание>

Присваивание -> <Лево><равно><Право>

Присваивание -> <ВнутриСписка><равно><Право>

ВнутриСписка -> <ИМЯ><ЛскбКВ><ИМЯ><ПскбКВ>

ВнутриСписка -> <ИМЯ><ЛскбКВ><СписокЦифр><ПскбКВ>

равно -> <=>

Лево -> <ИМЯ>

Право -> <Вызов>

Право -> <Строчка>

Право -> <ОпределениеСписка>

Право -> <Подсчет>

Право -> <Получить>

Право -> <ИМЯ>

Получить -> <ПОЛУЧИТЬ><СписокЦифр><ВНУТРИ><ИМЯ>

Получить -> <ПОЛУЧИТЬ><ИМЯ><ВНУТРИ><ИМЯ>

ПОЛУЧИТЬ -> <получить>

Определение -> <Тип><ИМЯ>

т -> <;>

Тип -> <Цел>

Тип -> <Дроб>

Тип -> <Строка>

Тип -> <Список>

Цел -> <ЦЕЛ>

Дроб -> <ДРОБ>

Строка -> <СТРОКА>

Список -> <СПИСОК>

ЦЕЛ -> <целое>

ДРОБ -> <дробное>

СТРОКА -> <строка>

СПИСОК -> <список>

Объявление -> <Тип><ИМЯ><равно><Право>

Пока -> <ИМЯ><двточ><ПОКА><Право><ВЫПОЛН><СписокБлоков><ВСЕ>

ПОКА -> <пока>

ВЫПОЛН -> <выполнять>

ВСЕ -> <все>

двточ -> <:>

Стоп -> <СТОП><ИМЯ><т>

СТОП -> <стоп>

Продолжить -> <ПРОДОЛЖИТЬ><ИМЯ><т>

ПРОДОЛЖИТЬ -> <продолжить>

Функция -> <ФУНКЦИЯ><ИМЯ><Лскб><СписПарам><ВЫПОЛН><СписокБлоков><ВСЕ>

ФУНКЦИЯ -> <функция>

ВЕРНУТЬ -> <вернуть>

СписПарам -> <ИМЯ><Пскб>

СписПарам -> <ИМЯ><Запятая><СписПарам>

СписПарам -> <Пскб>

СписПарамФ -> <ИМЯ><Пскб>

СписПарамФ -> <ИМЯ><Запятая><СписПарамФ>

СписПарамФ -> <Пскб>

Вызов -> <ИМЯ><Лскб><СписПарамФ>

Если -> <ЕСЛИ><Право><ВЫПОЛН><СписокБлоков><ИНАЧЕ><СписокБлоков><ВСЕ>

Если -> <ЕСЛИ><Право><ВЫПОЛН><СписокБлоков><ВСЕ>

ЕСЛИ -> <если>

ИНАЧЕ -> <иначе>

Для -> <ИМЯ><двточ><ДЛЯ><ИМЯ><ДО><ИМЯ><ШАГ><ИМЯ><ВЫПОЛН><СписокБлоков><ВСЕ>

Для -> <ИМЯ><двточ><ДЛЯ><ИМЯ><ВНУТРИ><ИМЯ><ВЫПОЛН><СписокБлоков><ВСЕ>

ДЛЯ -> <для>

ДО -> <до>

ШАГ -> <шаг>

ВНУТРИ -> <внутри>

Строчка -> <Кавыч><СписокБукв><Кавыч>

Строчка -> <ДвКавыч><СписокБукв><ДвКавыч>

Строчка -> <ДвКавыч><ДвКавыч>

Строчка -> <Кавыч><Кавыч>

СписокБукв -> <Буква><СписокБукв>

СписокБукв -> <Буква>

Кавыч -> <'>

ДвКавыч -> <">

ОпределениеСписка -> <ЛскбКВ><ПскбКВ>

ОпределениеСписка -> <ЛскбКВ><СписокЭлементовСписка><ПскбКВ>

ЛскбКВ -> <[>

ПскбКВ -> <]>

СписокЭлементовСписка -> <ЭлементСписка>

СписокЭлементовСписка -> <ЭлементСписка><Запятая><СписокЭлементовСписка>

ЭлементСписка -> <ИМЯ>

ЭлементСписка -> <Строчка>

ЭлементСписка -> <Число>

Подсчет -> <Выр><сравнение><Выр>

Подсчет -> <Выр>

Выр -> <Лскб><Выр><Пскб><множить><Выр>

Выр -> <Лскб><Выр><Пскб>

Выр -> <Авыр><множить><Выр>

Выр -> <Авыр>

Авыр -> <Лскб><Авыр><Пскб><сложить><Выр>

Авыр -> <Лскб><Авыр><Пскб>

Авыр -> <ИМЯ><сложить><Выр>

Авыр -> <ИМЯ>

Авыр -> <Число>

ИМЯ -> <СписокСимволов>

СписокСимволов -> <Симв><Цифра><Цифра>

СписокСимволов -> <Симв><Цифра>

СписокСимволов -> <Симв>

Число -> <сложить><СписокЦифр><точка><СписокЦифр>

Число -> <СписокЦифр><точка><СписокЦифр>

Число -> <сложить><СписокЦифр>

Число -> <СписокЦифр>

СписокЦифр -> <Цифра><СписокЦифр>

СписокЦифр -> <Цифра>

точка -> <.>

Запятая -> <,>

Лскб -> <(>

Пскб -> <)>

сравнение -> <==>

сравнение -> <!=>

сравнение -> <>=>

сравнение -> <<=>

сравнение -> <>>

сравнение -> <<>

множить -> <\*>

множить -> </>

сложить -> <+>

сложить -> <->

ВсеСимволы -> <0> | … | <9>

Симв -> <П>

Цифра -> <0> | … | <9>

Буква -> <А> | … | <Я> | <0> … <9> | <\_>

Можно заметить, что программа всегда состоит из списка блоков, который в свою очередь состоит из Блока или списка блоков. В дальнейшем это позволит сильно ускорить этап создания дерева разбора. Т.К. разбирать длинную строку токенов программы – долгий процесс.

В грамматике строго отсутствует левая рекурсия, правая рекурсия допустима, т.к. необходима в многих конструкциях языка.

# Создание парсера

Парсер читает файл и первым этапом избавляется от всех разделителей – пробелов, переносов строк и возвратов каретки. Далее из полученных элементов нужно получить список токенов. Важно учитывать, что обычно пользователи ставят символы точка с запятой, запятая сразу после строки кода, поэтому их нужно отделить и записать как отдельный токен.

Далее список токенов разбивается на подсписки. Они собираются из отдельных блоков программы. Например, большие блоки кода – функции, циклы, условные конструкции, выделяются в отдельные подсписки токенов. Каждая строка вне этих конструкций так же выделяется в отдельный подсписок. Получить такие подсписки – тривиальная задача. Обычный блок кода всегда заканчивается точкой с запятой, а большие блоки всегда заканчиваются токеном «все» и начинаются токеном «выполнять».

Файл программы на языке apalang имеет формат .ap

Так как все токены не являющиеся именами переменных написаны в нижнем регистре, а имена переменных записаны большими буквами, процесс разбора на токены так же сильно упрощается.

Листинг 1. Получение токенов из файла

def GetTokensByFile(fileName, myrezerved):

    f = open(fileName, 'r')

    stroka = f.read()

    mytokens = ''.join(stroka.split())

    mytokens.encode('cp1251')

    buffer = []

    endind = 1

    sind = 0

    while True:

        if mytokens[sind:endind] in myrezerved:

            buffer.append(mytokens[sind:endind])

            sind = endind

        endind += 1

        if endind == len(mytokens) + 1:

            break

    if sind + 1 != endind: return None

    return buffer

# Создание дерева разбора

Благодаря тому, что грамматика не содержит левой рекурсии, дерево разбора можно создать при помощи алгоритма LL разбора.  
 Модификация алгоритма с сохранением состояния позволяет создавать деревья даже при наличии разных выводов из одного нетерминала. Если не получится создать дерева перебрав все возможные варианты правил, алгоритм выдаст ошибку.

Процесс создания дерева таким способом имеет огромное преимущество перед стандартным LL анализатором – он не зависит от приоритета выбора правил и справляется с построением дерева по любой допустимой строке токенов. Существенным его недостатком является скорость работы. Для больших программ, если пытаться составить дерево не оптимизируя процесс, времени может быть потрачено до получаса.

Однако, как было сказано выше, процесс был сильно (в случаях больших программ до 100 раз) ускорен.

Дерево представляет собой структуру из элементов типа Node (тип описан далее). Дерево имеет методы печати. Все методы для работы с деревом документированы и выполнены на высокой степени абстракции и задокументированы, что позволяет расширять функционал программы.

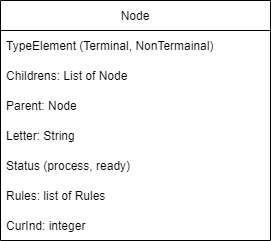


Рис 1. Структура Node

Сам же процесс оптимизации возможен благодаря структуре правил грамматики.

Программа всегда начинается с токена «начало» и заканчивается токеном «конец». Любое дерево разбора любой программы всегда состоит из вершин типа «Блок» и «Список Блоков» это позволяет брать каждый подсписок токенов, полученный на предыдущем этапе и разбирать его отдельно. Единственное что для этого нужно – исправить стартовый символ грамматики языка на «Список Блоков».

В результате такого процесса получится список из готовых деревьев. Если в процессе построения «мини дерева» произойдет ошибка – пользователь узнает точно в каком именно блоке была допущена ошибка и сообщение об этом будет выведено на экран.

Изначально если в грамматике отсутствуют токены начала и конца, так же будет выведена ошибка.

Если ошибок не было обнаружено - сообщает о корректности синтаксиса, будет построено дерево разбора.

Дерево будет строиться таким образом:

Строится корень дерева и от него ответвляются начало и конец, лист «Список Блоков» остается незаполненным.

Далее каждое «мини дерево» будет присоединяться к основному дереву в порядке, в котором оно было получено. Последнее дерево присоединится, не создавая новое ответвление.

Таким образом будет получено дерево.

# Кодогенерация

После создания дерева будет создан JavaScript код.

Процесс основан на использовании стека и обратного обхода в ширину.

Обратный обход в ширину вернет список вершин дерева разбора таким образом:

1. Выбираются вершины, у которых нет потомков (для готового дерева разбора – это будут символы терминалы) и заносятся в стек разобранных вершин.
2. Выбираются вершины все потомки которых находятся в стеке вершин, и они заносятся в стек вершин.
3. Процесс повторяется пока в стеке вершин не окажутся все вершины.

Благодаря такому списку вершин, можно утверждать следующее: если преобразовывать вершину в готовый код строго в порядке, в котором она находится в списке, полученным при обратном разборе в ширину, то если она содержит в себе детей и является Нетерминалом, то все ее дети уже будут разобраны.

Так как все терминалы грамматики можно однозначно закодировать в JavaScript (как и в любой другой язык), а любую вершину типа нетерминал можно так же однозначно закодировать. То процесс кодогенерации гарантированно вернет синтаксически верный код. Кроме того, он будет идентичен коду, из токенов которого было построено дерево.

При разборе очередной вершины, ее сгенерированный код помещается в стек. Если у вершины есть потомки, то из стека будут «вытянуты» все ее разобранные дети (это будут точно ее дети исходя из утверждения выше, и они точно будут разобраны), а сгенерированный для нее код будет опять помещен в стек.

После работы алгоритма в стеке останется один элемент – итоговый код программы.

После получения кода программы, необходимо подключить библиотеки ввода вывода. После их подключения последний оставшийся элемент стека записывается в файл и получается готовый JavaScript код, который уже можно запускать.

# Создание исполняемого файла

Имя готовый код, его нужно поместить в html обертку для запуска в браузере или при помощи node.js собрать из него исполняемый файл. В данной работе был выбран способ с html файлом.

Поместив код в тэг <script> получаем файл, запустив который в браузере будет исполнена исходная программа на языке apalang.

Что бы получить исполняемый файл, нужно вызвать запуск браузера по умолчанию с параметром страницы – файла с кодом.

# Тестирование и запуск

Программа запускается с обязательными параметрами запуска:

1. Имя файла программы на языке apalang
2. Имя создаваемого исполняемого файла



Рис 2. Файлы до компиляции

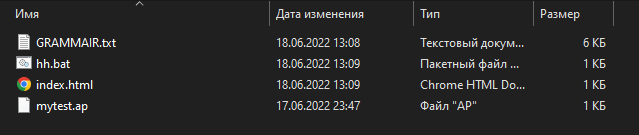


Рис 3. Файлы после компиляции

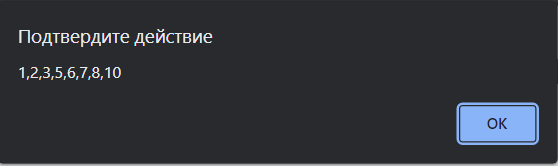


Рис 4. Результат работы программы «bubble sort» для списка [1,5,3,7,6,8,2,10]

Компиляция выполняется за 19.873 секунды. Результат можно ускорить, оптимизируя написанный код используя распределение по блокам меньшего размера.

Если указать флаг –g при запуске программы на экран будет выведено дерево разбора, при указании флага –grammar будет создан файл GRAMMAIR.txt полным описанием грамматики.

Листинг 2. «bubble sort» для списка [1,5,3,7,6,8,2,10] на языке apalang

начало

П = [1,5,3,7,6,8,2,10];

П1 = 8;

П50 = П1 - 1;

П51 = 1;

П2 = 0;

П40: для П2 до П50 шаг П51

выполнять

П3 = 0;

П52 = П1 - 1;

П52 = П52 - П2;

П41: для П3 до П52 шаг П51

выполнять

П71 = П3 + 1;

П61 = получить П3 внутри П;

П62 = получить П71 внутри П;

если П61 > П62

выполнять

П[П3] = П62;

П[П71] = П61;

все

все

все

вывод П;

конец

# Список использованной литературы

1. Контекстно свободная грамматика [Электронный ресурс] https://ru.wikipedia.org/wiki Контекстно-свободная\_грамматика 2020 г.
2. LLанализатор [Электронный ресурс] https://ru.wikipedia.org/wiki/ LL-анализатор. 2018 г.
3. АХО А.В, ЛАМ М.С., СЕТИ Р., УЛЬМАН Дж.Д. Компиляторы: принципы, технологии и инструменты. – М.: Вильямс, 2008.
4. Разработка компиляторов на платформе .NET / А. Терехов, Н. Вояковская, Д. Булычев, А. Москаль.
5. D. Grune, C. H. J. Jacobs "Parsing Techniques - A Practical Guide", Ellis Horwood, 1990. [Электронный ресурс] URL: http://www.cs.vu.nl/~dick/PTAPG.html
6. Introduction to Recursive Descent Parsing. [Электронный ресурс] URL: <http://ag-kastens.uni-paderborn.de/lehre/material/uebi/parsdemo/recintro.html>
7. Parsing Expressions by Recursive Descent. [Электронный ресурс] URL: http://www.engr.mun.ca/~theo/Misc/exp\_parsing.html