Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Кафедра вычислительной техники

Отчет по лабораторной работе № 4

по дисциплине «Теория формальных языков и компиляторов»

на тему «Синтаксис языков программирования. Нисходящий синтаксический анализ»

Студент: Шабанов М. Г.

Вариант: 24221342

Группа: АВТ-709

Преподаватель: Малявко А.А.

Новосибирск, 2020

# Цель работы

# Изучение основных идей и понятий нисходящих методов синтаксического анализа, выявление свойств формальных грамматик, необходимых для реализации нисходящего восстановления дерева грамматического разбора, приобретение навыков построения процедурной и различных автоматных реализаций нисходящего анализа, исследование поведения нисходящих синтаксических акцепторов.

# Постановка задачи

* 1. Используя пакет ВебТрансЛаб:
* изучить и освоить проверку принадлежности грамматики к классу **LL(1)**, используя в качестве проверяемых грамматики, полученные при выполнении работы №4;
* построить конечный автомат со стековой памятью и несколькими состояниями (шаблон …SyntAsMultiFSM…), разобраться в структуре управляющей таблицы автомата, уяснить способы формирования и использования всех полей;
* построить конечный автомат со стековой памятью и одним состоянием, управляемый входным символом и символом, снятым с верхушки стека (шаблон …SyntAsSingleFSM…), разобраться в структуре управляющей таблицы автомата, уяснить способы формирования и использования клеток таблицы;
* построить процедурную реализацию рекурсивного спуска (шаблон …SyntAsRD…), уяснить способы формирования функций этого акцептора;
  1. Выполнить трассировку процессов нисходящего синтаксического акцепта, изучить поведение всех построенных синтаксических акцепторов при разборе как правильных предложений, так и предложений с намеренно внесенными синтаксическими ошибками.
  2. Проанализировать и сравнить между собой все полученные тексты программ и результаты выполнения пункта 3.2. Оценить степень пригодности изученных вариантов реализации нисходящих синтаксических акцепторов для выполнения курсовой работы.
  3. Подготовить, сдать и защитить отчет к лабораторной работе.

Вариант:

Идентификаторы: *<бБ><пЦ><бБ > (т0т, r2R,…)*

Константы: *целые по основанию 10 и указанному в константе; вещественные; символьные.*

Типы: *int\_u] float letter*

Оператор присваивания: *<И> let <В>;*

Условный оператор: *at <ЛВ> do <ОБ> [ else do <ОБ> ]*

Оператор цикла: *cycle([<ОП>];[<ЛВ>];[<О>]) <ОБ>*

Оператор переключателя: **?? <В> { ? <К> : <ОБ> [gout;] … [ ?~ : <ОБ> ] }**

Формат псевдокода: *пентады<М><Код><Оп><Оп><Р>*

# Теоретическая часть

*LL(1)-грамматикой* называется такая контекстно-свободная грамматика, у которой множества выбора правил с одинаковой нетерминалом в левой част и попарно не пересекаются.

Принято считать, что символы в названии класса *LL(1)-грамматик* обозначают следующее.

*Первая буква L (сокращение слова left - левый)* - чтение слов анализируемого предложения производится слева направо.

*Вторая буква L (сокращение слова leftmost - самый левый)* - на каждом шаге принимается решение для замены самого левого нетерминала из текущего уровня восстанавливаемого дерева.

*Цифра 1 в скобках* обозначает количество символов из начала остатка предложения, необходимых для принятия решения о выборе правила на каждом шаге детерминированного нисходящего восстановления дерева грамматического разбора.

Функционирование *конечного автомата со стековой памятью и несколькими состояниями* определяется управляющей таблицей. Предполагается, что автомат при запуске оказывается в особом начальном состоянии, на каждом такте по входному символу и текущему состоянию определяет и выполняет операции над входным потоком символов, стековой памятью и собственным состоянием.

Каждому символу каждого правила грамматики должно быть поставлено в соответствии в точности одно состояние автомата. С каждым состоянием должно быть связано множество выбора и два адреса перехода. Под адресом перехода понимается номер состояния.

При соблюдении определенных правил нумерации состояний и введении операции управления остановом по ошибке можно обойтись только одним адресом перехода.

С каждым состоянием должны быть также связаны операции управления стековой памятью и чтением следующего входного символа. Все операции управления могут задаваться булевскими значениями *true / false*, которые называются флажками. Обозначения для флажков управления операциями:

* + *флаг a* управляет чтением следующего входного символа;
  + *флаг s* управляет занесением адреса точки возврата в стек;
  + *флаг r* обеспечивает переключение автомата в состояние, номер которого снимается с верхушки стека возвратов;
  + *флаг e* запрещает останов по ошибке, когда состояние соответствует нетерминалу из левой части и есть еще хотя бы одно правило для такого нетерминала.

*LL(1)-грамматику* легко можно преобразовать в *конечный автомат с единственным состоянием* и стековой памятью, управляемый:

* + текущим входным символом;
  + символом, находящимся на верхушке стека.

Поведение такого автомата определяется управляющей таблицей, столбцы которой соответствуют входным символам, строки - символам, которые могут находиться в стеке, а в клетках указана некоторая последовательность операций над стеком, входным потоком и состоянием автомата.

# Ход работы

**Доработки и правки с учетом замечаний по предыдущим работам:**

Исправлено правило лексики:

includeFiles ['][a-z]+[.d-z]\*['] → ['][a-z]+[.]?[a-z]\*['].

Добавлены правила синтаксиса:

restStartExpr "[" arrayIndexes "]"

arrayIndexes int

restStartExpr CallFunArgs

CallFunArgs "(" Expression [ "," Expression ]? ")"

Исправлены правила синтаксиса:

Block Operator Limiter → Block Operator

Block "{" [ Operator Limiter ]\* "}" → Block "{" [ Operator ]\* "}"

Asing Indetifier "let" Expression → Asing Indetifier "let" Expression Limiter

Declaration [ "const" ]? VarType Indetifier [ "let" Expression ]? →

Declaration [ "const" ]? VarType Indetifier [ "let" Expression ]? Limiter

В следствии исправлений, связанных с Limiter, потребовалось изменить правила описывающие условный оператор, чтобы грамматика принадлежала классу LL(1). Теперь правила заданы так:

Operator "at" Expression "do" Block RestAt

RestAt "else""do" Block

RestAt ~~else~~

Изменены правила синтаксиса, связанные с типами переменных, добавлены типы функций.

Изменено правило общего вида программы, временное правило Code удалено, теперь код программы обязан содержать хотя бы одну функцию, даже пустую.

Добавлено описание функций. Функция выглядит следующим образом:

*тип имя\_функции (список\_переменных)*

*{*

*тело\_функции*

*}*

В теле функции может быть любое количество операторов, return может не быть.

*Не получилось сделать вызов функции, как процедуры, т.е без присваивания ее результата переменной.*

Доработана тестовая программа:

#include 'studio.h'

void b0e(int a0r[], int s1z){

int i0j;

int j0j;

cycle(i0j let 0 ; i0j < s1z - 1; i0j let (i0j + 1)) {

cycle(j0j let 0 ; j0j < s1z - j0j - 1 ; j0j let ( j0j + 1)){

int t9p let a0r[j0j];

a0r[j0j] let a0r[j0j +1] ;

a0r[j0j +1] let t9p;

}

}

}

int R4N(){

int n0n let 10;

int a0a[10] let {12, 432, 342, 23523, 34, 342, 34, 234, 234, 234};

int c0d;

c0d let S44f(); //scanf

int r3s let 0;

?? c0d {

? 11 : {

r3s let 0;

} gout;

? 20 : {

r3s let b0e(a0a, n0n);

} gout;

?~ : {

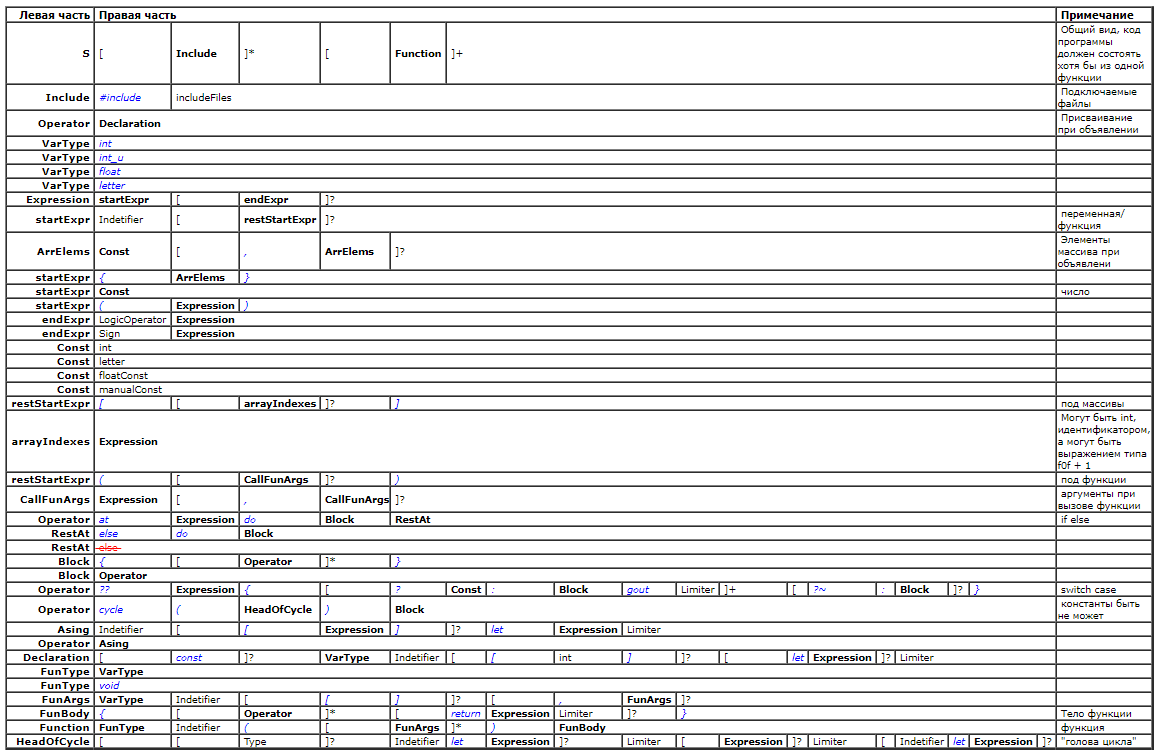
r3s let 0;

}

}

return 0;}

Обновленные правила показаны на рисунках 1 и 2.



*Рис. 1 - Обновленные правила синтаксиса*



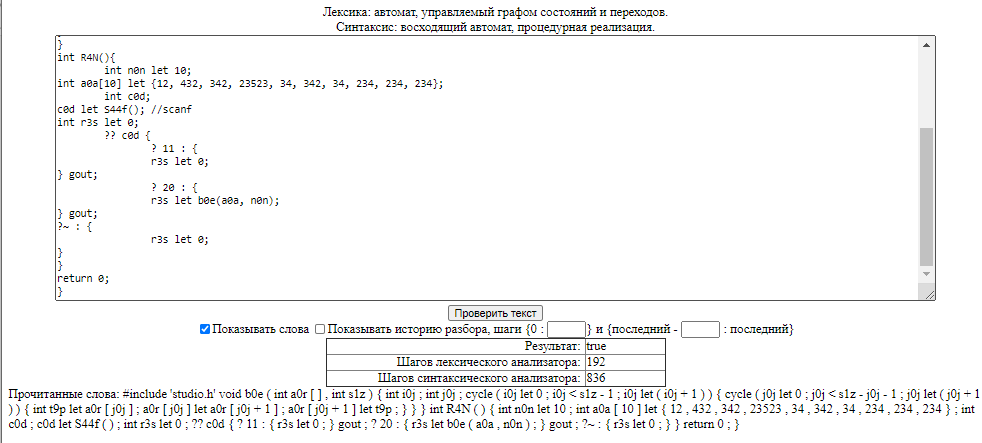
*Рис. 2 - Обновленные правила лексики*

Множества выбора правил для данной грамматики показаны в Приложении 1. Видно, что для каждого нетерминала множества выбора его правил попарно не пересекаются. Значит грамматика принадлежит классу *LL(1)-грамматик*.

Правила грамматики показаны в Приложении 2.

Управляющая таблица нисходящего автомата с несколькими состояниями показана в Приложении 3.

Тестирование обновлённых правил синтаксиса показано на Рис. 2.

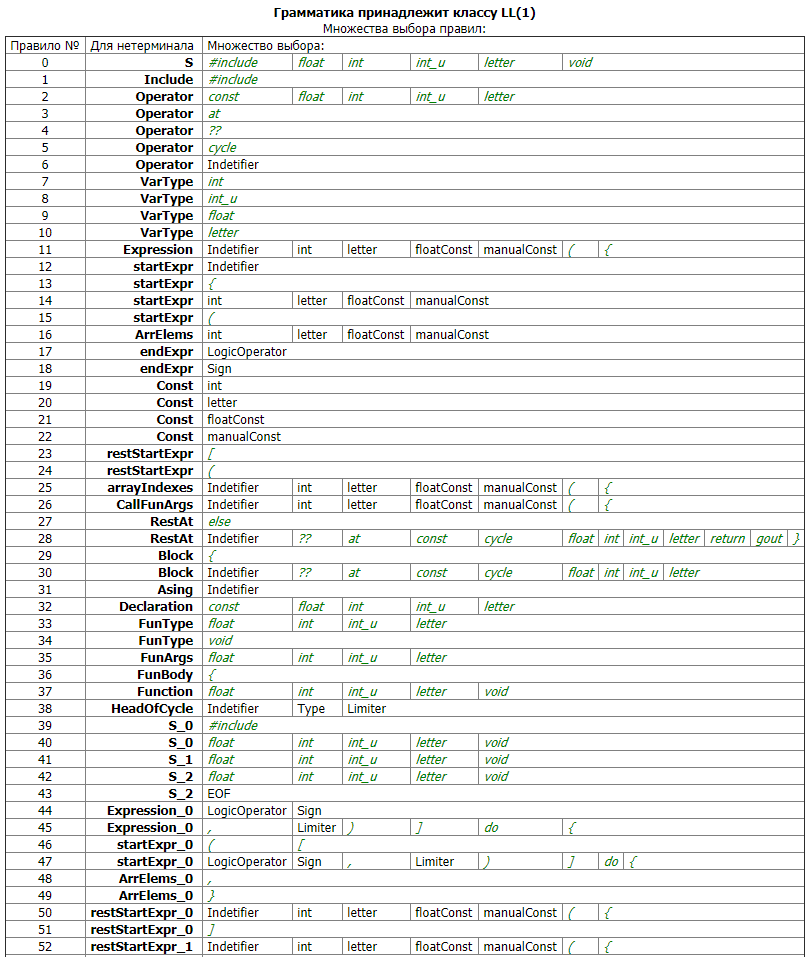


*Рис. 3 - Проверка обновленных правил синтаксиса*

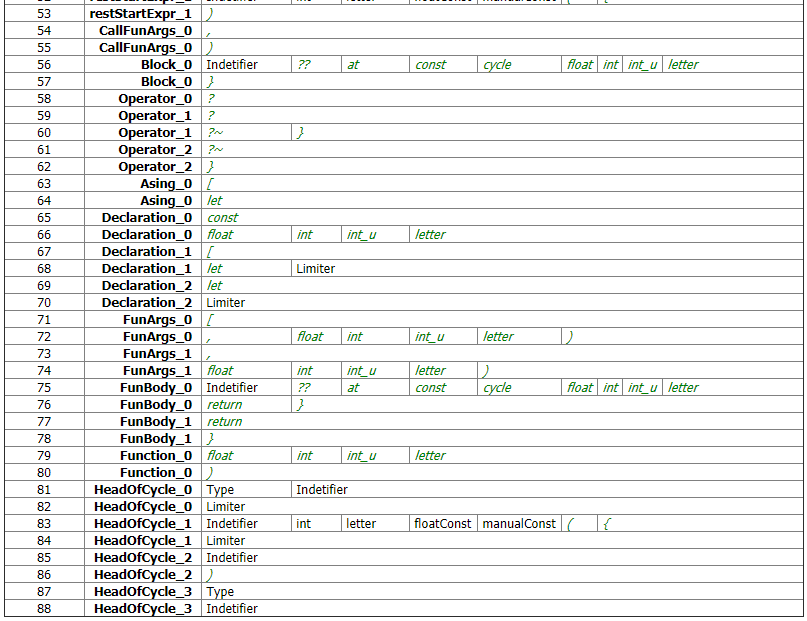
# Вывод

В ходе лабораторной работы были изучены основных идеи и понятия нисходящих методов синтаксического анализа. Была доработана грамматика для языка, в соответствии с заданным вариантом курсовой работы*.*

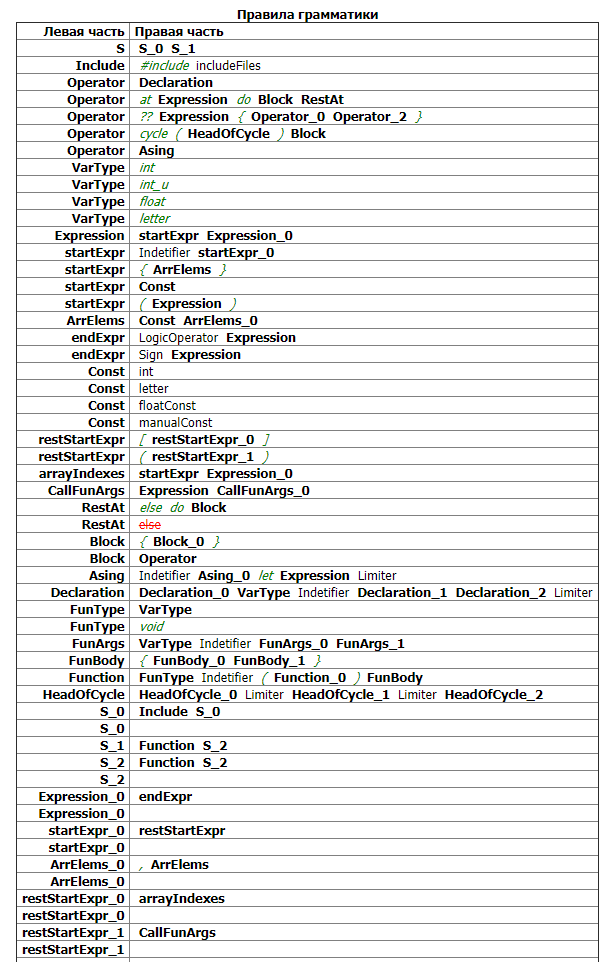
# Приложение 1



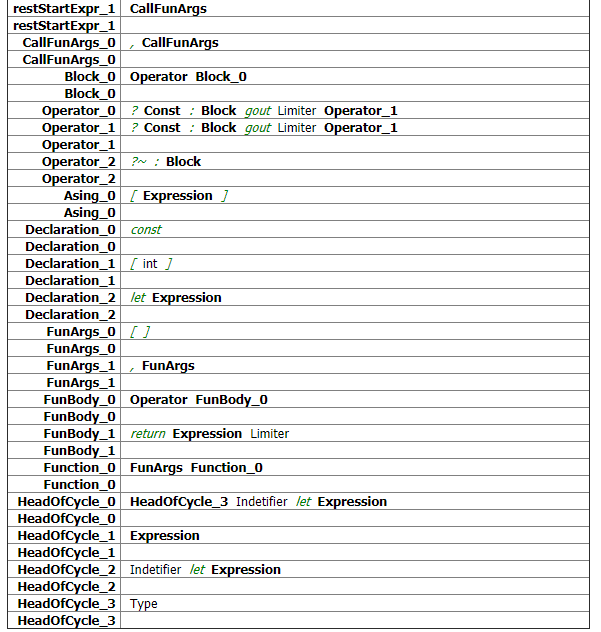
# Приложение 1



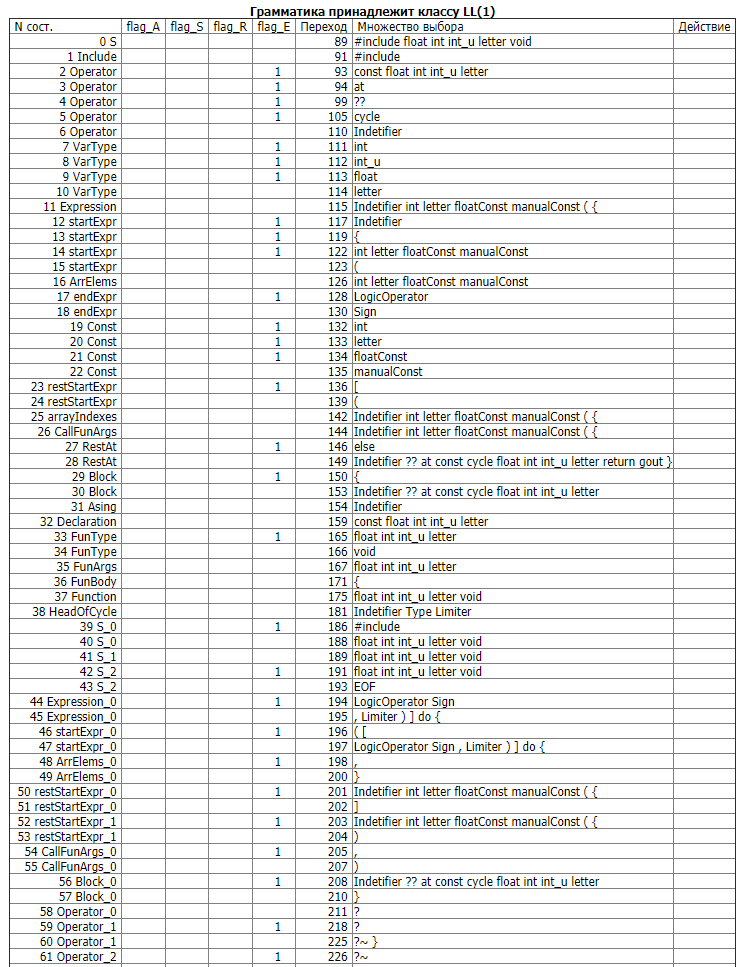
# Приложение 2



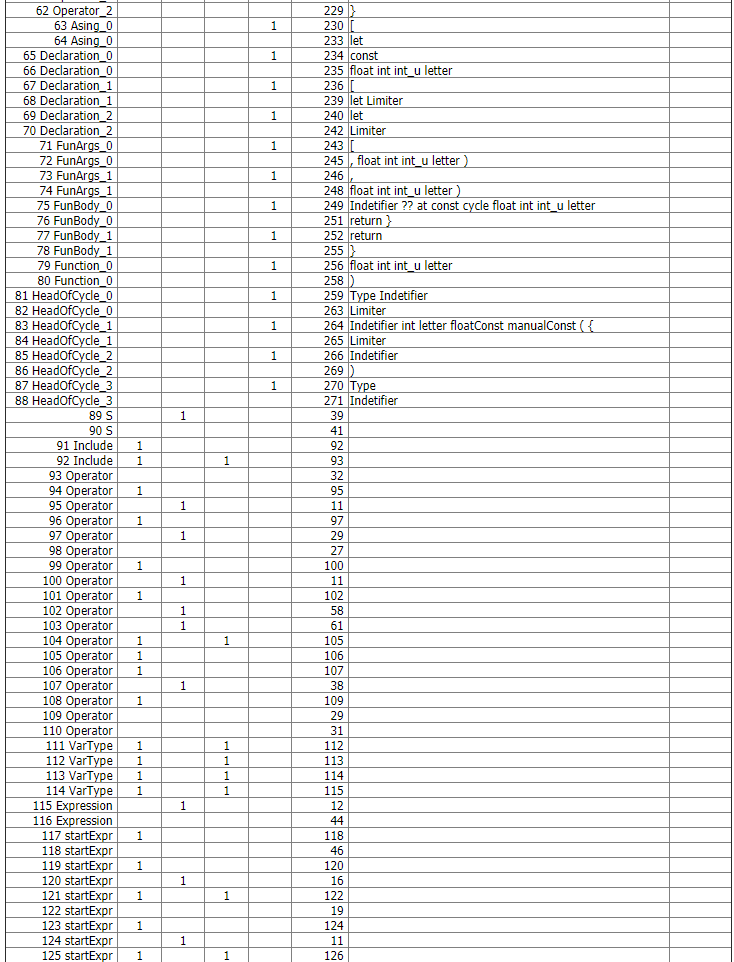
# Приложение 2



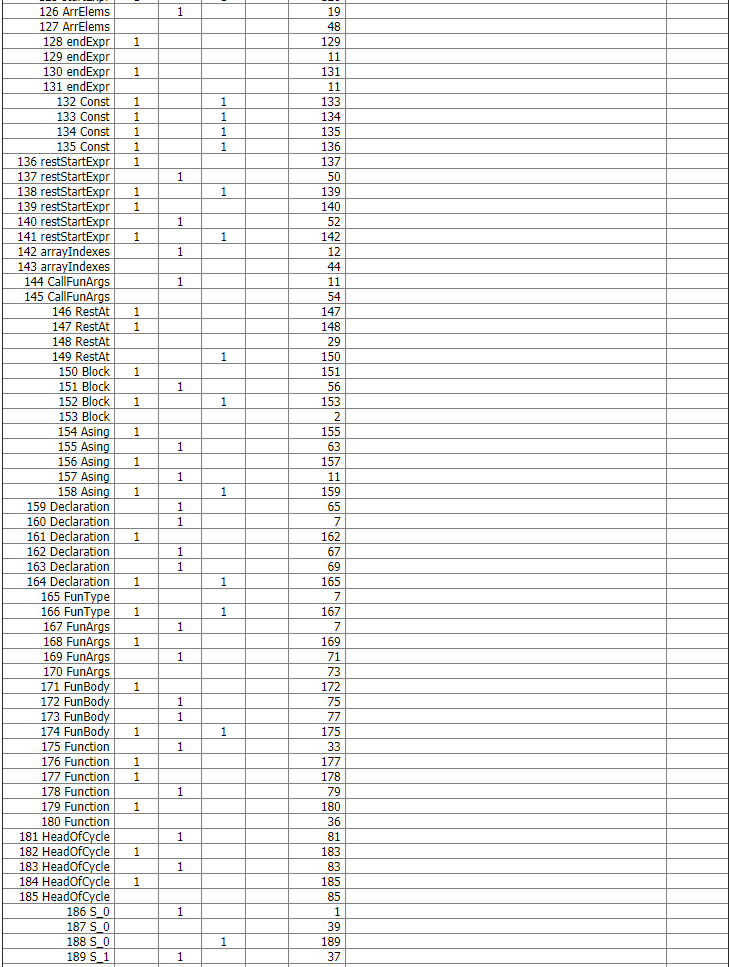
# Приложение 3



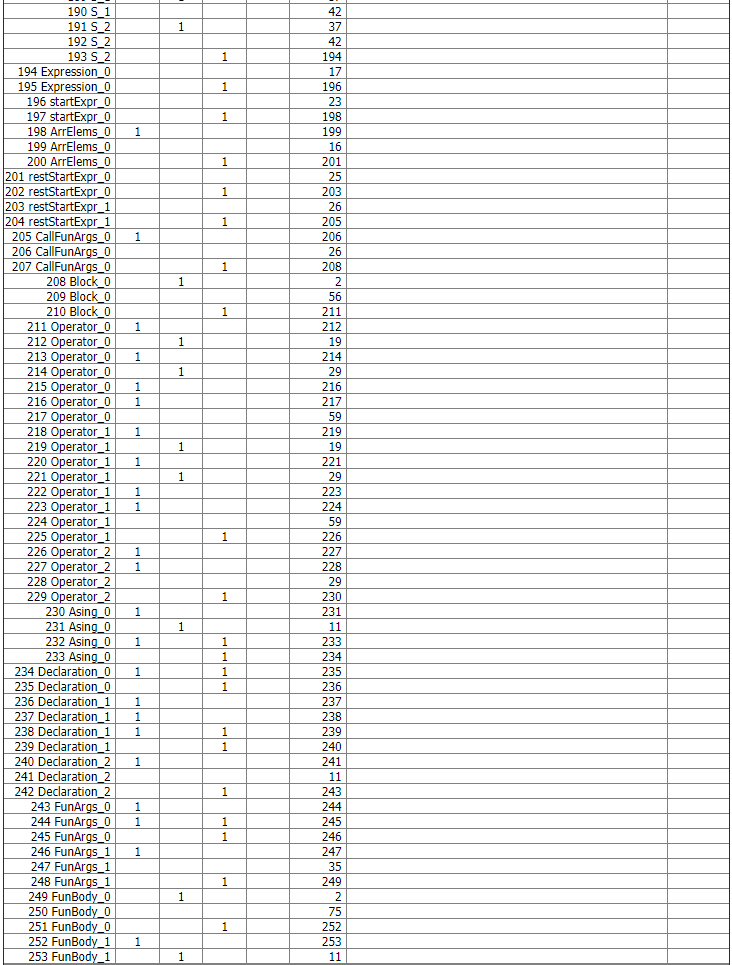
# Приложение 3



# Приложение 3



# Приложение 3



# Приложение 3

