**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых» (ВлГУ)

**РАЗРАБОТКА КОМПИЛЯТОРА ПОДМНОЖЕСТВА ПРОЦЕДУРНОГО ЯЗЫКА**

Пояснительная записка

На 23 листах

|  |  |  |
| --- | --- | --- |
| Руководитель |  | к.т.н. доцент кафедры ИЗИ Ю.М. Монахов |
| Исполнитель |  | студент гр. ИБ-118 П.А Селезнева |

**Владимир 2021**

**Оглавление**

**Аннотация** …………………………………………………………………….3

1. ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА……………………………........4
   1. Основные требования………………………………………………….4
   2. Лексический анализатор………………………………………………5
   3. Синтаксический анализатор…………………………………………...6
   4. Построение генератора объектного кода……………………………..9
2. [ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ………………………………………………………….11](#_Toc72369775)

# **Аннотация**

В данном программном документе приведён текст компилятора подмножества процедурного языка. Код программы был реализован на языке программирования Python с помощью библиотеки PLY.

Разработка программа состоит из лексического анализатора, синтаксического анализатора, таблицы символов, промежуточного представления и генератора целевого кода.

Лексический анализатор разбивает программу на поток токенов и передает их синтаксическому анализатору.

Синтаксический анализатор сопоставляет полученный поток токенов с заданной грамматикой, тем самым проверяя на принадлежность данного кода к заданному языку программирования.

При синтаксическом анализе формируется абстрактное синтаксическое дерево (АСТ), после чего по полученному дереву генерируется таблица символов и трехадресный код.

Трёхадресный код – это совокупность записей, приведенная к следующему виду: первый аргумент, второй аргумент и операция, которая проводится над ними.

Из трёхадресного кода генерируется машинный код. Программы, скомпилированные данной программой, могут компилироваться на машинах с архитектурой MIPS.

Для проверки работоспособности машинного кода использовалась программа «QtSpim».

# **1. ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА**

## Основные требования

К данной курсовой работе установлены следующие требования:

* Требования к входному языку:

1. Должны присутствовать операторные скобки;
2. Должна игнорироваться индентация программы;
3. Должны поддерживаться комментарии любой длины;
4. Входная программа должна представлять собой единый модуль, но поддерживать вызов функций.

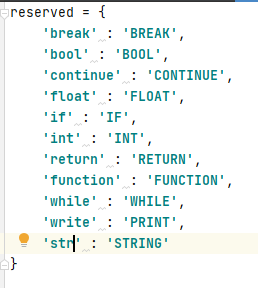
* Требования к операторам:

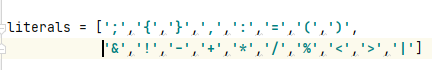
1. Оператор присваивания;
2. Арифметические операторы;
3. Логические операторы (И, ИЛИ, НЕ);
4. Условный оператор (ЕСЛИ);
5. Оператор цикла (while);
6. Базовый вывод (строковой литерал, переменная);
7. Типы (целочисленный 32 бита, с плавающей запятой 32 бита).

## Лексический анализатор

На данном этапе содержимое файла программы разбивается на токены в соответствии с описанными в лексере регулярными выражениями. Результатом работы программы является поток токенов. Лексический анализатор был написан с использованием библиотеки PLY. Каждый токен определяется при помощи регулярного выражения. Кроме токенов, лексер может содержать в себе список зарезервированных слов и литералов.

На рисунке 1 представлен список зарезервированных слов, а также список литералов:





## Синтаксический анализатор

## Второй стадией разработки компилятора является разработка КС-грамматики и синтаксического анализатора. Синтаксический анализатор получает на вход массив токенов, которые в последующем проверяются на принадлежность языку, указанному в грамматике. В соответствии с синтаксическим анализом строится дерево разбора, которое содержит в себе название блока кода в левой части и своих потомков в правой части. В последующем при обходе данного дерева будет формироваться таблица символов. В данной работе используется синтаксический анализатор, предоставленный библиотекой PLY.

КС-грамматика языка представлена ниже.

Rule 0 S' -> start  
Rule 1 start -> body\_programm  
Rule 2 start -> function\_declaration\_list body\_programm  
Rule 3 function\_declaration\_list -> function  
Rule 4 function\_declaration\_list -> function\_declaration\_list function  
Rule 5 body\_programm -> identifier\_list compound\_statement  
Rule 6 identifier\_list -> identifier  
Rule 7 identifier\_list -> identifier\_list ; identifier  
Rule 8 identifier -> type declaration\_list  
Rule 9 type -> INT  
Rule 10 type -> FLOAT  
Rule 11 type -> STRING  
Rule 12 declaration\_list -> IDENTIFIER  
Rule 13 declaration\_list -> declaration\_list , IDENTIFIER  
Rule 14 function -> FUNCTION IDENTIFIER ( identifier\_list ) { function\_statement\_list }  
Rule 15 function -> FUNCTION IDENTIFIER ( identifier\_list ) { identifier\_list function\_statement\_list }  
Rule 16 function\_calling -> IDENTIFIER ( arguments )  
Rule 17 arguments -> argument  
Rule 18 arguments -> arguments , argument  
Rule 19 argument -> IDENTIFIER  
Rule 20 argument -> INT\_N  
Rule 21 argument -> FLOAT\_N  
Rule 22 argument -> ( expression )  
Rule 23 compound\_statement -> { statement\_list }  
Rule 24 statement\_list -> statement  
Rule 25 statement\_list -> statement\_list ; statement  
Rule 26 statement -> assigment\_statement  
Rule 27 statement -> print\_statement  
Rule 28 statement -> while\_statement  
Rule 29 statement -> if\_statement  
Rule 30 statement\_list\_if -> statement\_if  
Rule 31 statement\_list\_if -> statement\_list\_if ; statement\_if  
Rule 32 statement\_if -> if\_statement  
Rule 33 statement\_if -> assigment\_statement  
Rule 34 statement\_if -> print\_statement  
Rule 35 statement\_if -> while\_statement  
Rule 36 statement\_if -> CONTINUE  
Rule 37 statement\_if -> BREAK  
Rule 38 function\_statement\_list -> function\_statement  
Rule 39 function\_statement\_list -> function\_statement\_list ; function\_statement  
Rule 40 function\_statement -> if\_statement  
Rule 41 function\_statement -> assigment\_statement  
Rule 42 function\_statement -> print\_statement  
Rule 43 function\_statement -> while\_statement  
Rule 44 function\_statement -> return\_statement  
Rule 45 return\_statement -> RETURN expression  
Rule 46 assigment\_statement -> IDENTIFIER = expression  
Rule 47 assigment\_statement -> IDENTIFIER = STRING\_LITERAL  
Rule 48 expression -> multiplication  
Rule 49 expression -> expression + multiplication  
Rule 50 expression -> expression - multiplication  
Rule 51 multiplication -> item  
Rule 52 multiplication -> multiplication \* item  
Rule 53 multiplication -> multiplication / item  
Rule 54 item -> function\_calling  
Rule 55 item -> IDENTIFIER  
Rule 56 item -> INT\_N  
Rule 57 item -> FLOAT\_N  
Rule 58 item -> ( expression )  
Rule 59 print\_statement -> PRINT ( expression )  
Rule 60 print\_statement -> PRINT ( STRING\_LITERAL )  
Rule 61 if\_statement -> IF logical\_expression { statement\_list\_if }  
Rule 62 while\_statement -> WHILE logical\_expression { statement\_list }  
Rule 63 logical\_expression -> logical\_or\_expression  
Rule 64 logical\_expression -> ! logical\_or\_expression  
Rule 65 logical\_or\_expression -> logical\_and\_expression  
Rule 66 logical\_or\_expression -> logical\_or\_expression | logical\_and\_expression  
Rule 67 logical\_and\_expression -> logical\_and\_expression & boolean  
Rule 68 logical\_and\_expression -> boolean  
Rule 69 boolean -> ( expression EQ\_OP expression )  
Rule 70 boolean -> ( expression > expression )  
Rule 71 boolean -> ( expression < expression )

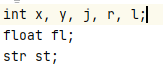
Привила использования грамматики представлены ниже.

**Объявление переменных**

Объявление переменных определятся следующими правилами КС-грамматики:

identifier\_list -> identifier  
Rule 7 identifier\_list -> identifier\_list ; identifier  
Rule 8 identifier -> type declaration\_list  
Rule 9 type -> INT  
Rule 10 type -> FLOAT  
Rule 11 type -> STRING

В программе это выглядит следующим образом:



**Присваивание**

Блок присваивания определяется следующими правилами грамматики:

Rule 46 assigment\_statement -> IDENTIFIER = expression  
Rule 47 assigment\_statement -> IDENTIFIER = STRING\_LITERAL

В программе это выглядит следующим образом:



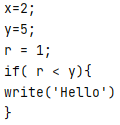
**Условие**

Условия определяются следующими правилами КС-грамматики:

Rule 61 if\_statement -> IF logical\_expression { statement\_list\_if }

Условный оператор содержит в себе ключевое слово if и состоит из выражения, по которому определяется истинность или ложность высказывания, и блока команд, которые выполняются в случаи правильности условия.

В программе это выглядит следующим образом:



**Циклы**

Блок цикла определяется следующим правилом КС-грамматики:

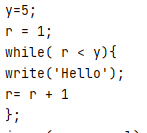
Rule 62 while\_statement -> WHILE logical\_expression { statement\_list }

Аналогично блоку условия, блок цикла содержит ключевое слово (*while*), условие цикла в скобках и тело цикла в фигурных скобках.

Тело цикла будет выполняться до тех пор, пока условие цикла истинно.

Кроме того, для данного блока имеются 2 специфичных оператора: *break* и *continue*. Оператор *break* позволяет досрочно выйти из тела цикла и продолжить выполнение команд, идущих за ним, а оператор *continue* позволяет прервать текущую итерацию и начать выполнение тела цикла заново.

Пример:



**Вызов встроенных функций**

В языке есть встроенные функции, которые определяются в грамматике следующим образом:

Rule 16 function\_calling -> IDENTIFIER ( arguments )

Пример:



**Стандартный вывод**

Встроенная функция *write(a)* позволяет выводить на стандартный вывод выражение *a.*

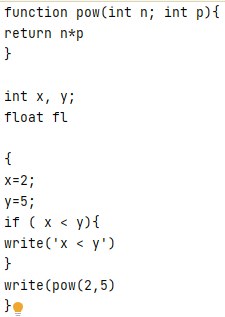
Пример:

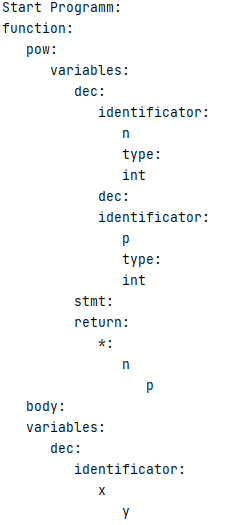


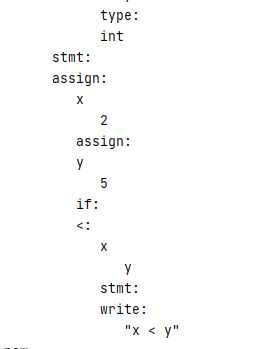
## Построение генератора объектного кода

Паралельно с синтаксическим анализом происходит построение абстрактного синтаксического дерева, которое представляет собой набор узлов, которые содержат в себе операторы, а листьями являются операнды.

Ниже представлен пример программы и составленного по нему АСТ:





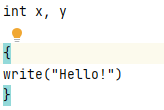


# **ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ**

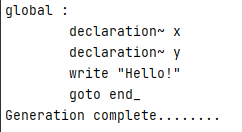
При проектировании компилятора к основному языку были установлены следующие минимальные требования: наличие операторных скобок, игнорирование пробелов и идентации программы, поддержка многострочных комментариев и вызова функций. Наличие операторов присваивания, условных, цикла, арифметических, логических. Должны присутствовать два типа данных – целочисленный и вещественный.

Далее приведено тестирование компилятора.

Программа «Hello!»:

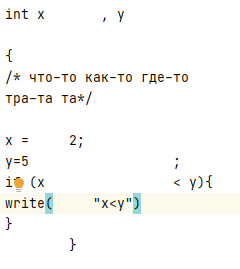


Итог работы программы:

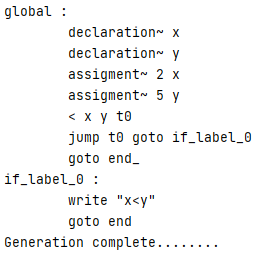


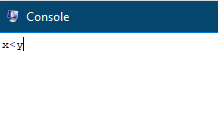


Проверка на игнорирование пробелов и индентацию, а также проверка многострочных комментариев:

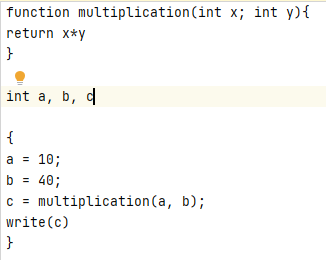


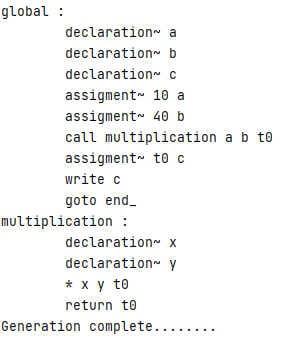
Итог работы программы:





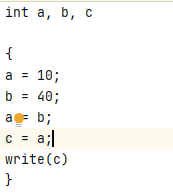
Проверка вызова функций

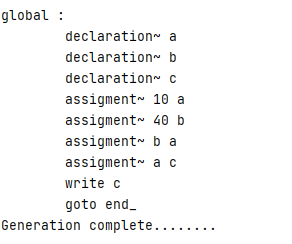






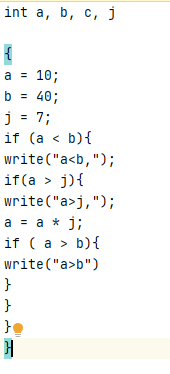
Проверка оператора присваивания:

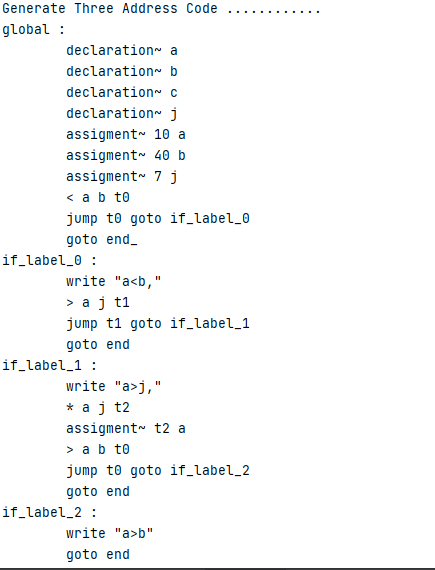


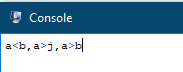




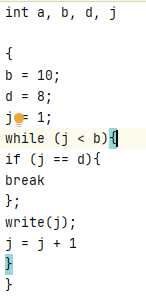
Проверка условного оператора:

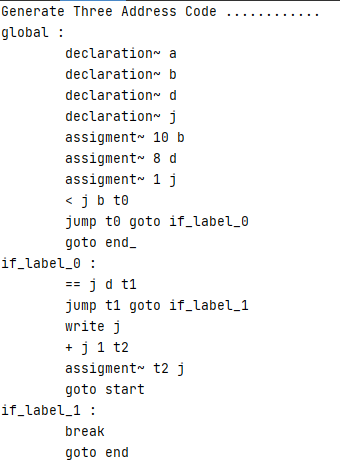


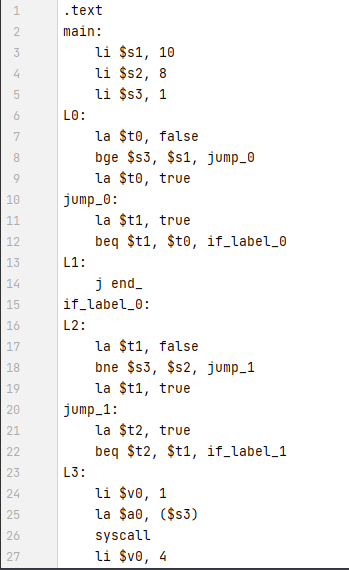


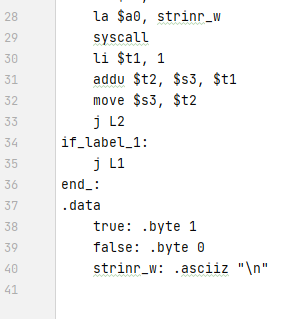


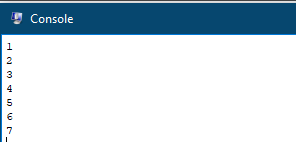
Проверка оператора цикла:



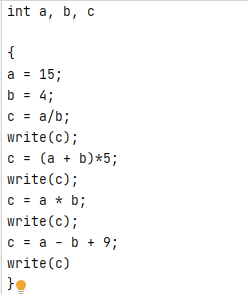


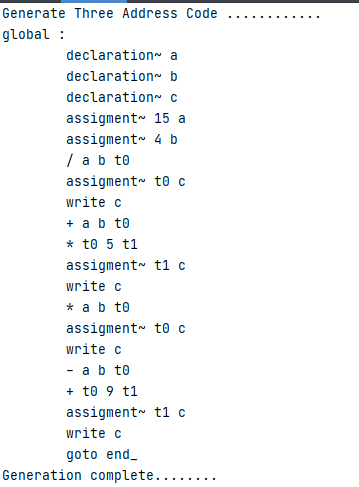


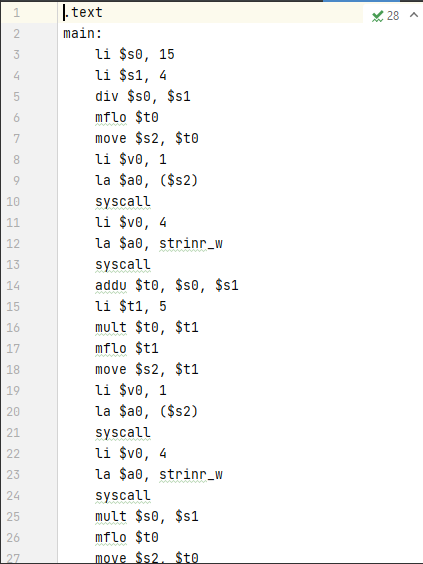


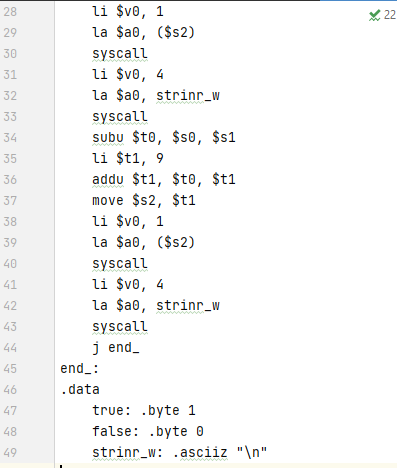


Проверка арифметики:

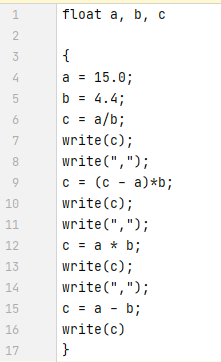


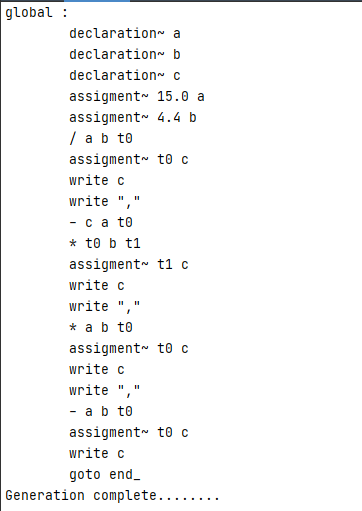














Реквизиты к курсовой работе: https://github.com/sind-coder/ps/