**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**

Пояснительная записка

На 22 листах

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

**Владимир 2021**

**Оглавление**

[**Аннотация** 3](#_Toc72675939)

[**1 ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА** 4](#_Toc72675940)

[1.1 Основные требования 4](#_Toc72675941)

[1.2 Лексический анализатор 5](#_Toc72675942)

[1.3 Синтаксический анализатор 6](#_Toc72675943)

[1.4 Таблица символов 13](#_Toc72675944)

[1.5 Промежуточный код 14](#_Toc72675945)

[1.6 Генерация кода ассемблера 15](#_Toc72675946)

[**2** **ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ** 16](#_Toc72675947)

# **Аннотация**

В данном программном документе приведён текст компилятора подмножества процедурно-ориентированного языка. Компилятор реализован на языке Python с использованием библиотеки ply. Основная функция компилятора – проверка принадлежности исходной цепочки входному языку и генерация выходной цепочки символов виде кода для MIPS-ассемблера.

Разработка компилятора подмножества процедурного языка в ассемблер состоит из следующих стадий:

* построение лексического анализатора;
* построение синтаксического анализатора;
* построение генератора MIPS-кода

# **1 ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА**

## Основные требования

Разработка будет производиться в соответствии со следующими требованиями:

* 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. Арифметика (\*, /, +, -, >, <, =);
3. Логические операторы (И, ИЛИ, НЕ);
4. Условный оператор (ЕСЛИ);
5. Оператор цикла (while);
6. Базовый вывод (строковой литерал, переменная);
7. Типы (целочисленный, вещественный).

## Лексический анализатор

Лексический анализатор является первой фазой компилятора. Он преобразует входной поток символов в поток токенов.

Грамматика языка реализована с использованием библиотеки ply.lex. Грамматика языка включает в себя элементы из популярных языков, таких как Pascal, Java, Python и др.

Список зарезервированных слов:

* def
* return
* and
* or
* not
* print
* while
* do
* if
* then
* break
* continue
* int
* real
* var
* str

А также прочие символы на подобии *“{}”, “=”, “>”* и т.д.

## Синтаксический анализатор

Второй стадией компилятора является синтаксический анализ. На вход синтаксическому анализатору подаётся набор токенов из лексического анализатора. На основе грамматики языка строится дерево разбора грамматики.

Синтаксический анализатор реализован с использованием библиотеки ply.yacc.

КС-грамматика языка:

S' -> prog

prog -> VAR dec\_list OPEN\_FIG stmt\_list CLOSE\_FIG

prog -> VAR dec\_list def\_list OPEN\_FIG stmt\_list CLOSE\_FIG

def\_list -> def

def\_list -> def\_list SEMI\_COLON def

def -> DEF ID OPEN\_PAREN dec\_list CLOSE\_PAREN OPEN\_FIG stmt\_list\_def CLOSE\_FIG

def -> DEF ID OPEN\_PAREN dec\_list CLOSE\_PAREN OPEN\_FIG VAR dec\_list stmt\_list\_def CLOSE\_FIG

defstmt -> ID OPEN\_PAREN args CLOSE\_PAREN

args -> arg

args -> args SEMI\_COLON arg

arg -> ID

arg -> NUMBER\_INT

arg -> NUMBER\_REAL

arg -> OPEN\_PAREN exp CLOSE\_PAREN

dec\_list -> dec

dec\_list -> dec\_list SEMI\_COLON dec

dec -> id\_list DOUBLE\_POINT type

type -> INT

type -> REAL

type -> STRING

id\_list -> ID

id\_list -> id\_list COMA ID

stmt\_list -> stmt

stmt\_list -> stmt\_list SEMI\_COLON stmt

stmt -> assign

stmt -> print

stmt -> while

stmt -> if

stmt\_list\_if -> stmt\_if

stmt\_list\_if -> stmt\_list\_if SEMI\_COLON stmt\_if

stmt\_if -> assign

stmt\_if -> print

stmt\_if -> while

stmt\_if -> if

stmt\_if -> CONTINUE

stmt\_if -> BREAK

stmt\_list\_def -> stmt\_def

stmt\_list\_def -> stmt\_list\_def SEMI\_COLON stmt\_def

stmt\_def -> assign

stmt\_def -> print

stmt\_def -> while

stmt\_def -> if

stmt\_def -> return

return -> RETURN exp

assign -> ID PRISV exp

assign -> ID PRISV STRING

exp -> term

exp -> exp SUM term

exp -> exp SUB term

term -> factor

term -> term MUL factor

term -> term DIV factor

factor -> defstmt

factor -> ID

factor -> NUMBER\_INT

factor -> NUMBER\_REAL

factor -> OPEN\_PAREN exp CLOSE\_PAREN

print -> PRINT OPEN\_PAREN exp CLOSE\_PAREN

print -> PRINT OPEN\_PAREN STRING CLOSE\_PAREN

while -> WHILE bool\_exp DO OPEN\_FIG stmt\_list CLOSE\_FIG

if -> IF bool\_exp THEN OPEN\_FIG stmt\_list\_if CLOSE\_FIG

bool\_exp -> bool\_exp OR bool\_exp\_term

bool\_exp -> bool\_exp\_term

bool\_exp -> NOT bool\_exp

bool\_exp -> bool

bool\_exp\_term -> bool\_exp\_term AND bool

bool\_exp\_term -> bool

bool -> OPEN\_PAREN exp RAVNO exp CLOSE\_PAREN

bool -> OPEN\_PAREN exp MORE exp CLOSE\_PAREN

bool -> OPEN\_PAREN exp LESS exp CLOSE\_PAREN

Правила использования грамматики представлены ниже.

**Объявление переменных**

Блок объявления переменных представляется следующим правилом КС-грамматики:

dec\_list -> dec

dec\_list -> dec\_list SEMI\_COLON dec

dec -> id\_list DOUBLE\_POINT type

type -> INT

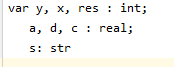
type -> REAL

type -> STRING

id\_list -> ID

id\_list -> id\_list COMA ID

Пример:



**Присваивание**

Блок присвоения представляется следующим правилом КС-грамматики:

assign -> ID PRISV exp

assign -> ID PRISV STRING

Для присваивания необходимо указать имя переменной и выражение, которое ей присваивается, либо строку, если переменная имеет тип str.

Пример:

****

**Условный оператор**

Условие определяется следующим правилом КС-грамматики:

if -> IF bool\_exp THEN OPEN\_FIG stmt\_list\_if CLOSE\_FIG

Блок обязательно содержит ключевое слово *if*, условие ветвления в скобках, затем ключевое *then* и тело в фигурных скобках.

Пример:



**Циклы**

Блок цикла определяется следующим правилом КС-грамматики:

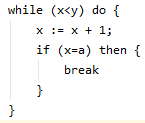
while -> WHILE bool\_exp DO OPEN\_FIG stmt\_list CLOSE\_FIG

Блок цикла содержит ключевое слово *while*, условие цикла в скобках, ключевое слово *do* 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:



**Объявление функций**

Пользовательские функции задаются после блока объявления переменных и перед основным телом программы. Объявление функции начинается с ключевого слова *def,* затем в скобках задаются аргументы с указанием типов, и далее в фигурных скобках тело функции.

Объявление функций определяется следующим правилом КС-грамматики:

def -> DEF ID OPEN\_PAREN dec\_list CLOSE\_PAREN OPEN\_FIG stmt\_list\_def CLOSE\_FIG

def -> DEF ID OPEN\_PAREN dec\_list CLOSE\_PAREN OPEN\_FIG VAR dec\_list stmt\_list\_def CLOSE\_FIG

Пример:



**Вызов функций**

В теле программы функции можно вызвать, указав название функции, и в скобках указав аргументы.

Пример:



**Стандартный вывод**

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

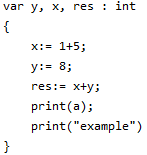
Пример:



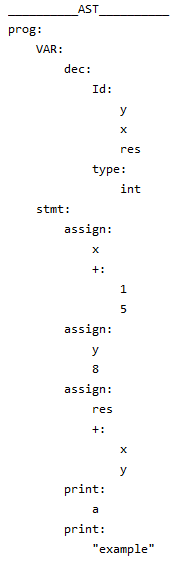
На выходе синтаксический анализатор возвращает дерево синтаксического разбора.

Пример:

Программа:



Дерево синтаксического разбора:



## Таблица символов

Следующим этапом является генерация таблицы символов. Таблица символов содержит в себе все переменные, их типы, область видимости, а также используемые регистры в MIPS-коде.

Для генерации таблицы символов используется рекурсивный обход синтаксического дерева обхода.

Пример:

Программа:

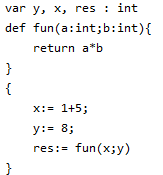
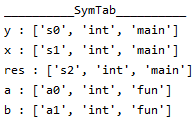


Таблица символов:



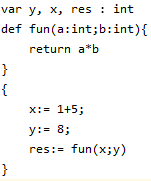
## Промежуточный код

Следующим этапом является генерация промежуточного кода. Промежуточный код строится следующим образом:

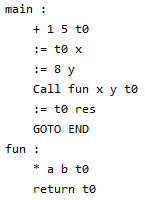
[операция] [первый аргумент] [второй аргумент] [результат]

Промежуточный код генерируется с помощью рекурсивного обхода дерева синтаксического обхода.

Пример:



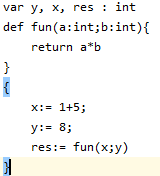
Промежуточный код:



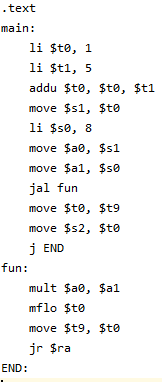
## Генерация кода ассемблера

Целевым языком является язык ассемблера MIPS. Промежуточный код построчно транслируется в код ассемблера MIPS.

Пример для программы:



Код ассемблера MIPS:



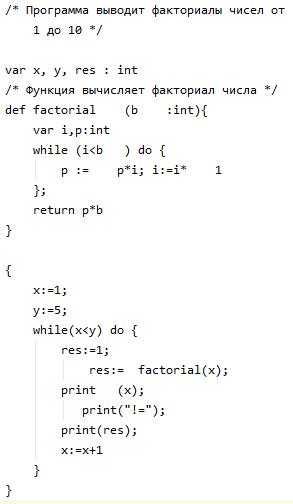
Для запуска сгенерированного кода используется эмулятор MIPS QtSpim.

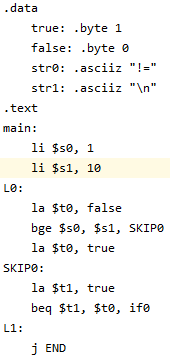
# **ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ**

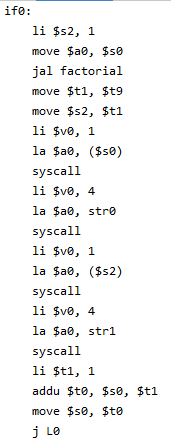
При проектировании компилятора к основному языку были установлены следующие минимальные требования: наличие операторных скобок, игнорирование индентации программы, поддержка комментариев любой и вызова функций. Наличие операторов присваивания, условных, цикла, арифметических, логических, базового вывода. Должны присутствовать два типа данных – целочисленный и с плавающей запятой.

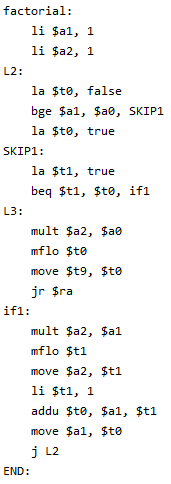
Далее приведено тестирование компилятора.

Проверка на игнорирование индентации, комментариев, проверка вызова функций, оператора присваивания, арифметики, оператора цикла, базового вывода:

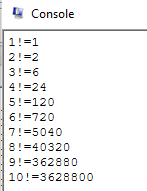




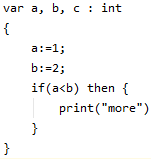


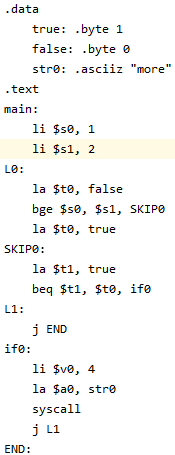


Результат запуска:

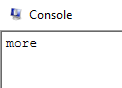


Проверка условного оператора:

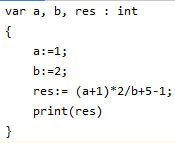


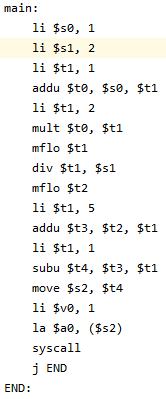


Результат запуска:

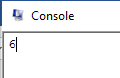


Проверка арифметики:

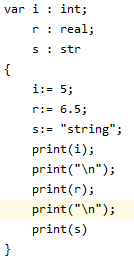


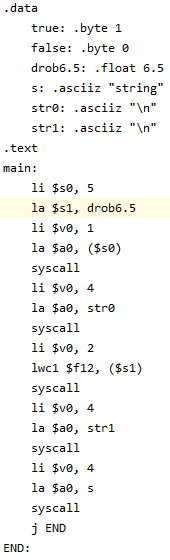


Результат при запуске:

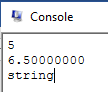


Проверка типов данных:





Результат запуска:



Реквизиты к курсовой работе:

<https://github.com/Gogopel/compilator>