Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых» (ВлГУ)

Кафедра «Информатики и защиты информации»

КУРСОВАЯ РАБОТА НА ТЕМУ:

РАЗРАБОТКА КОМПИЛЯТОРА ПОДМНОЖЕСТВА ПРОЦЕДУРНОГО ЯЗЫКА В АССЕМБЛЕР

СПЕЦИАЛЬНОСТЬ: 10.03.01 – ИНФОРМАЦИОННО-АНАЛИТИЧЕСКИЕ СИСТЕМЫ БЕЗОПАСНОСТИ

Выполнил:
ст. гр. ИБ-118
Жерихов Алексей Владимирович

Цель курсовой работы

- Разработать компилятор, выполняющий функцию перевода исходного текста программы в формат машинного кода языка Assembler.

Техническое задание

- 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 также должна быть поддержка вызова функций.
- Операторы:
  - 1. Оператор присваивания.
  - 2. Арифметика (*, /, +, -, >, <, =).
  - 3. Логические операторы (И, ИЛИ, НЕ).
  - 4. Условный оператор (ЕСЛИ).
  - 5. Операторы цикла (while, break, continue).
  - 6. Базовый вывод (строковый литерал, переменная).
  - 7. Типы (целочисленный 32 бита, с плавающей запятой 32 бита).
- Требования к выходному языку:
  - 1. В ассемблере.

Компилятор

- **Компилятор** – это программа, которая переводит текст, написанный на языке программирования, в набор машинных кодов. В процессе работы данной программы происходит процесс **компиляции** - сборка программы, включающая трансляцию всех модулей программы, написанных на одном или нескольких исходных языках программирования высокого уровня в эквивалентные программные модули на низкоуровневом языке.

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.

Пример:

```
var i,j : int
{
    i:= 5;
    j:= 8;
}
```

```
LexToken(VAR,'var',1,1)
LexToken(ID,'i',1,5)
LexToken(COMA,',',1,6)
LexToken(ID,'j',1,7)
LexToken(DOUBLE_POINT,':',1,9)
LexToken(INT,'int',1,11)
LexToken(OPEN_FIG,'{',1,15)
LexToken(ID,'i',1,21)
LexToken(PRISV,':=',1,22)
LexToken(NUMBER_INT,'5',1,25)
LexToken(SEMI_COLON,';',1,26)
LexToken(ID,'j',1,32)
LexToken(PRISV,':=',1,33)
LexToken(NUMBER_INT,'8',1,36)
LexToken(SEMI_COLON,';',1,37)
LexToken(CLOSE_FIG,'}',1,39)
```

Синтаксический анализатор

- Второй фазой 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.

Пример:

```
var x, y, res : int
{
    x:= 1+5;
    y:= 8;
    res:= x+y;
    print(res);
    print("example")
}
```

```
_____AST_____
prog:
    VAR:
        dec:
            Id:
                x
                y
                res
            type:
                int
    stmt:
        assign:
            x
            +:
                1
                5
        assign:
            y
            8
        assign:
            res
            +:
                x
                y
        print:
            res
        print:
            "example"
```

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:

```
var x, y, res : int;
    r: real;
    s: str

def fun(a:int; b:int){
    return a*b
}
{
    x:= 1+5;
    y:= 8;
    res:= fun(x;y)
}
```

```
__________SymTab__________
x : ['s0', 'int', 'main']
y : ['s1', 'int', 'main']
res : ['s2', 'int', 'main']
r : ['s3', 'real', 'main']
s : ['s4', 'str', 'main']
a : ['a0', 'int', 'fun']
b : ['a1', 'int', 'fun']
```

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:

```
var y, x, res : int
def fun(a:int;b:int){
    return a*b
}
{
    x:= 1+5;
    y:= 8;
    res:= fun(x;y)
}
```

```
main :
    + 1 5 t0
    := t0 x
    := 8 y
    Call fun x y t0
    := t0 res
    GOTO END
fun :
    * a b t0
    return t0
```

Генерация кода ассемблера

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

Пример:

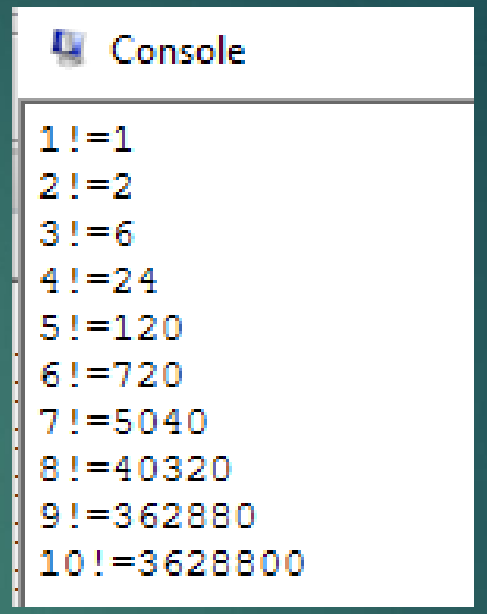
```
var y, x, res : int
def fun(a:int;b:int){
    return a*b
}
{
    x:= 1+5;
    y:= 8;
    res:= fun(x;y)
}
```

```
.text
main:
    li $t0, 1
    li $t1, 5
    addu $t0, $t0, $t1
    move $s1, $t0
    li $s0, 8
    move $a0, $s1
    move $a1, $s0
    jal fun
    move $t0, $t9
    move $s2, $t0
    j END
fun:
    mult $a0, $a1
    mflo $t0
    move $t9, $t0
    jr $ra
END:
```

Примеры программ

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

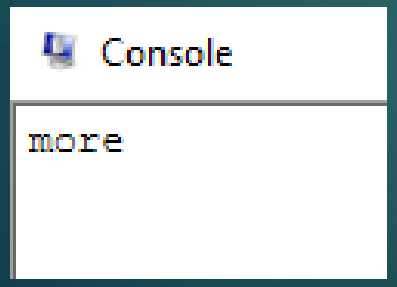
```
/* Программа выводит факториалы чисел от
    1 до 10 */

var x, y, res : int
/* Функция вычисляет факториал числа */
def factorial    (b    :int){
    var i,p:int
    while (i<b   ) do {
        p :=    p*i; i:=i*    1
    };
    return p*b
}

{
    x:=1;
    y:=5;
    while(x<y) do {
        res:=1;
            res:=  factorial(x);
        print   (x);
          print("!=");
        print(res);
        x:=x+1
    }
}
```



Примеры программ

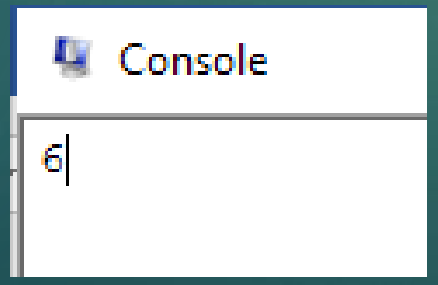
Проверка условного оператора

```
var a, b, c : int
{
    a:=1;
    b:=2;
    if(a<b) then {
        print("more")
    }
}
```



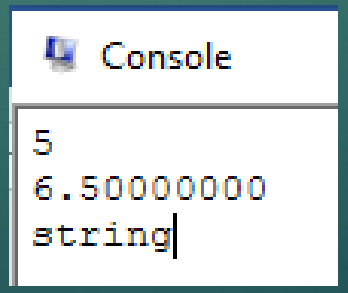
Проверка арифметики

```
var a, b, res : int
{
    a:=1;
    b:=2;
    res:= (a+1)*2/b+5-1;
    print(res)
}
```



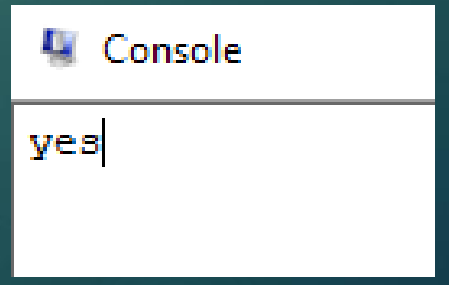
Проверка типов данных

```
var i : int;
    r : real;
    s : str
{
    i:= 5;
    r:= 6.5;
    s:= "string";
    print(i);
    print("\n");
    print(r);
    print("\n");
    print(s)
}
```



Проверка логических операторов

```
var x, y, z : int
{
    x:=1;
    y:=2;
    z:=3;
    if (x>y) and (z>y) then {
        print("yes")
    }
}
```



Спасибо за внимание!