

**Федеральное государственное автономное образовательное  
учреждение высшего образования «Национальный исследовательский  
университет ИТМО»**

**Факультет программной инженерии и компьютерной техники**

**Разработка компиляторов**

**Проект**

**Выполнили:**

**Кобик Никита Алексеевич**

**Маликов Глеб Игоревич**

**Группа № Р3324**

**Преподаватель:**

**Лаздин Артур Вячеславович**

**Санкт-Петербург**

**2025**

## Оглавление

Задание .....	3
Отчет .....	5
Описание языка .....	5
Операторы .....	5
Выражения.....	6
Примечания .....	6
Грамматика .....	6
Примеры работы.....	8
Фибоначчи.....	8
НОД .....	9
Обработка несуществующих переменных .....	10
Обработка ошибки в типах данных .....	10
Работа с областями видимости .....	11
Заключение .....	12

## Задание

Необходимо разработать компилятор для учебного языка программирования

Вход компилятора: программа, написанная на учебном языке программирования

Выход компилятора: текст программы на языке ассемблера

Предлагается использовать эмулятор RISC-V процессора

Характеристики учебного языка программирования

Императивный Тьюринг полный язык, поддерживающий следующий набор операторов и выражений:

- Операторы
  1. Определения переменных. Поддерживаемые типы данных: целые числа со знаком, строки
  2. Присваивание (может быть заменено выражением в случае определения оператора-выражения)
  3. Ветвление (if) с факультативным else
  4. Цикл while
  5. Блок (составной оператор)
  6. Вывода (типа print)
- Выражения
  1. Арифметические (минимум + \* - /)
  2. Логические not, and, or
  3. Для чего поддерживать операции (operator) арифметические, логические сравнения

Допускается использовать ключевые слова для определения операторов и операций отличные от общепринятых. Полный список операторов, операций и ключевых слов должен быть приведен в отчете

Для разработанного учебного языка программирования необходимо:

- Определить лексический состав языка и описать правила определения лексем для генератора лексических анализаторов flex. Учесть обработку комментариев: однострочных и многострочных.
- Определить разработанный язык в терминах КС грамматики в формате входного файла для GNU Bison.
- Для каждого правила грамматики (в рамках синтаксически управляемой трансляции) определить семантические действия, определяющие правила построения AST.
- Разработать функцию обхода AST для его визуализации. Для построения AST использовать примеры корректных программ (небольших) для учебного языка.

Второй этап предполагает генерацию кода в процессе обхода AST

Можно построить промежуточное представление программы в виде трехадресного кода, но это не является целью проекта

Цель второго этапа – получение ассемблерной программы эквивалентной тексту входной программы

## Отчет

Исходный код проекта в репозитории на GitHub

### Описание языка

Референсом является синтаксис Rust, претерпевший сильные упрощения

определение переменных использует ключевое слово `let`

типовая аннотация через двоеточие после идентификатора

блочная структура в фигурных скобках

набор операторов

Используются только два типа данных `int` со знаком и `string`

Полный набор ключевых слов

```
let
int
string
if
else
while
print
not
and
or
```

### Операторы

`let` – объявление и инициализация переменной

`<-` – присваивание

`if / else` – ветвление, с опциональным `else`

`while` – цикл с предусловием

`{ }` – блок

`print` – вывод в `stdout`

## Выражения

- Первичные  
NUMBER  
STRING  
IDENTIFIER  
скобки ( expr )
- Унарные  
-expr  
not expr
- Мультипликативные  
\*, /, %
- Аддитивные  
-, +
- Логические  
and  
or
- Сравнения  
<, >, <=, >=, ==, !=

## Примечания

Приоритеты стандартные, слева направо

Переменные могут быть локальными если объявлены внутри блоков

Есть возможность добавления однострочных и многострочных комментариев с помощью // и /\*\*/ соответственно

## Грамматика

```
grammar nlang;
```

```
@lexer::members {  
    private static String unescape(String raw) {  
        raw = raw.substring(1, raw.length() - 1);  
        return raw  
            .replace("\\\\\\\\", "\\")  
            .replace("\\\\\"", "\"")  
            .replace("\\b", "\b")  
            .replace("\\f", "\f")  
    }  
}
```

```

        .replace("\\n", "\n")
        .replace("\\r", "\r")
        .replace("\\t", "\t");
    }
}

program: statement* EOF;

statement
    : variableDeclaration ';'
    | assignment ';'
    | instruction
    ;

variableDeclaration: 'let' IDENTIFIER ':' type '=' expression;
type: 'int' | 'string';

assignment: IDENTIFIER '<-' expression;

instruction
    : ifStatement
    | whileStatement
    | printStatement
    ;

ifStatement: 'if' condition block ('else' block)?;
whileStatement: 'while' condition block;
printStatement: 'print' expression ';;';

condition: '(' expression ')';
block: '{' statement* '}';

expression
    : 'not' expression
    | '-' expression
    | expression 'and' expression
    | expression 'or' expression
    | expression op=('*' | '/') expression
    | expression op=('+' | '-') expression
    | expression '%' expression
    | expression comparator expression
    | '(' expression ')'
    | NUMBER
    | STRING
    | IDENTIFIER
    ;

comparator: '<' | '>' | '<=' | '>=' | '==' | '!=';

IDENTIFIER: [a-zA-Z_][a-zA-Z0-9_]*;
NUMBER: [0-9]+;

```

```

STRING
: '"' ( ESC | ~["\\r\n] )* '"'
  { setText(unescape(getText())); }
;

fragment ESC
: '\\\' [btnfr"\\]          //  \b \t \n \f \r \" \\
| '\\\' 'u' HEX HEX HEX HEX //  \uXXXX
;

fragment HEX : [0-9a-fA-F] ;

LINE_COMMENT  : '//' ~[\r\n]*          -> skip ;
BLOCK_COMMENT : '/*' .*? '*/'          -> skip ;

WS: [ \t\r\n]+ -> skip;

```

## Примеры работы

В директории `examples/` репозитория находятся исходные файлы тестовых программ на языке

Все получившиеся программы генерируют файлы на ассемблере в `output/asm/` и файлы с визуализацией AST в `output/svg/`

## Фибоначчи

Исходный код программы для расчета числа на определенной позиции из последовательности Фибоначчи (+ добавлены комментарии и работа со строками для полной проверки функционала)

```

let a: int = 0;
let b: int = 1;
let c: string = "Hello, World!"; // single line comment

let sum: int = 0;
let count: int = 0;

/* Multi line
comment *
aaaa? :)
*/
while (count < 10) {
  sum <- a + b;
  a <- b;
  b <- sum;
  count <- count + 1;
};

```

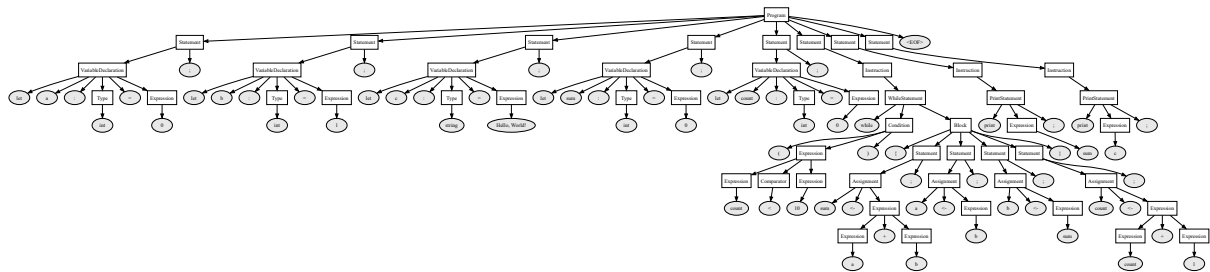


```
print sum;
print c;
```

## Результат работы

89  
Hello, World!

## Построенное дерево



## Результат компиляции

Reload
Step once
Run
Reload & Run
Stop

```

li x2, 65535      # sp
li x3, _heap     # heap ptr
li x18, 0
add x5, x18, x0  # a = init
li x19, 1
add x6, x19, x0  # b = init
li x20, _str0
add x7, x20, x0  # c = init
li x21, 0
add x8, x21, x0  # sum = init
li x22, 0
add x9, x22, x0  # count = init
while_top_1:
li x23, 10
slt x24, x9, x23
beq x24, x0, while_end_2 # while !cond -> end
add x25, x5, x6
add x8, x25, x0 # sum <- expr
add x5, x6, x0 # a <- expr
add x6, x8, x0 # b <- expr
li x26, 1
add x27, x9, x26
add x9, x27, x0 # count <- expr
jal x0, while_top_1
while_end_2:
add x12, x8, x0 # to x12
jal x1, print_int
li x12, 10
ewrite x12      # newline

```

Program input

89  
Hello, World!

Address	Hex	Decimal	Command	Explanation
0000	0001 0137	65847	lui x2, 16	x2 := 16 * 2^12
0001	FFF1 0113	-982765	addi x2, x2, -1	x2 := x2 + (-1)
0002	0000 01B7	439	lui x3, 0	x3 := 0 * 2^12
0003	0501 8193	83984787	addi x3, x3, 80	x3 := x3 + 80
0004	0000 0913	2323	addi x18, x0, 0	x18 := 0 + 0
0005	0009 02B3	590515	add x5, x18, x0	x5 := x18 + 0
0006	0010 0993	1051027	addi x19, x0, 1	x19 := 0 + 1
0007	0009 8333	623411	add x6, x19, x0	x6 := x19 + 0
0008	0000 0A37	2615	lui x20, 0	x20 := 0 * 2^12
0009	042A 0A13	69863955	addi x20, x20, 66	x20 := x20 + 66
000A	000A 03B3	656307	add x7, x20, x0	x7 := x20 + 0
000B	0000 0A93	2707	addi x21, x0, 0	x21 := 0 + 0
000C	000A 8433	689203	add x8, x21, x0	x8 := x21 + 0
000D	0000 0813	2835	addi x22, x0, 0	x22 := 0 + 0
000E	0008 0483	722099	add x9, x22, x0	x9 := x22 + 0
000F	00A0 0893	10488723	addi x23, x0, 10	x23 := 0 + 10

Address	Hex	Decimal	Command	Explanation			
pc	0023						
x0	0	x8	89	x16	48	x24	0
x1	32	x9	10	x17	0	x25	89
x2	65535	x10	0	x18	0	x26	1
x3	80	x11	0	x19	1	x27	10
x4	0	x12	10	x20	66	x28	0
x5	55	x13	0	x21	0	x29	0
x6	89	x14	0	x22	0	x30	0
x7	66	x15	10	x23	10	x31	0

## НОД

### Исходный код программы для поиска наибольшего общего делителя

```

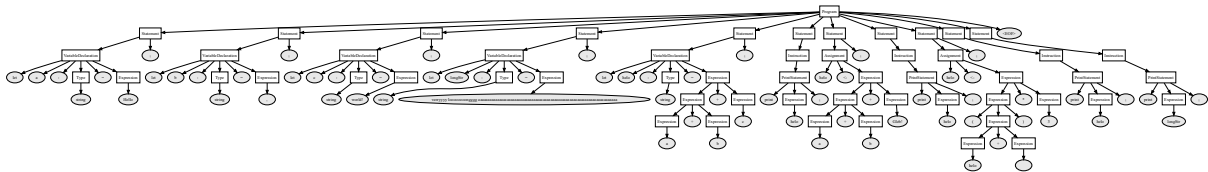
let a:int = 1071;
let b:int = 462;
while (b != 0) {
    let r:int = a % b;
    a <- b;
    b <- r;
}
print a;    // 21

```

## Результат работы

21

## Построенное дерево



## Результат компиляции

Reload

Step once

Run

Reload & Run

Stop

```
li x2, 65535      # sp
li x3, _heap      # heap ptr
li x18, 1071
add x5, x18, x0    # a = init
li x19, 462
add x6, x19, x0    # b = init
while_top_0:
li x20, 0
sne x21, x6, x20
beq x21, x0, while_end_1 # while !cond -> end
rem x22, x5, x6
add x7, x22, x0    # r = init
add x5, x6, x0    # a <- expr
add x6, x7, x0    # b <- expr
jal x0, while_top_0
while_end_1:
add x12, x5, x0    # to x12
jal x1, print_int
li x12, 10
write x12          # newline
ebreak
# ----- print_int(x12) -----
print_int:
blt x12, x0, _pi_neg
_pi_start:
beq x12, x0, _pi_zero
li x15, 10
li x16, 48
addi x14, x0, 0
```

Program input

21

Clear output

Address	Hex	Decimal	Command	Explanation
0000	0001 0137	65847	lui x2, 16	x2 := 16 * 2^12
0001	FFF1 0113	-982765	addi x2, x2, -1	x2 := x2 + (-1)
0002	0000 01B7	439	lui x3, 0	x3 := 0 * 2^12
0003	02E1 8193	48333203	addi x3, x3, 46	x3 := x3 + 46
0004	42F0 0913	1123027219	addi x18, x0, 1071	x18 := 0 + 1071
0005	0009 02B3	590515	add x5, x18, x0	x5 := x18 + 0
0006	1CE0 0993	484444563	addi x19, x0, 462	x19 := 0 + 462
0007	0009 8333	623411	add x6, x19, x0	x6 := x19 + 0
0008	0000 0A13	2579	addi x20, x0, 0	x20 := 0 + 0
0009	0743 2A83	121842355	sne x21, x6, x20	x21 := x6 != x20
000A	000A 8563	689507	beq x21, x0, 5	if x21 == x0 then pc := pc + 5
000B	0262 EB33	40037171	rem x22, x5, x6	x22 := x5 % x6
000C	0008 03B3	721843	add x7, x22, x0	x7 := x22 + 0
000D	0003 02B3	197299	add x5, x6, x0	x5 := x6 + 0
000E	0003 8333	230195	add x6, x7, x0	x6 := x7 + 0
000F	FFFF 806F	-32657	jal x0, -8	pc := pc - 8

<<

<

pc 0015

>

>>

pc	0015
x0	0 x8 0 x16 48 x24 0
x1	18 x9 0 x17 0 x25 0
x2	65535 x10 0 x18 1071 x26 0
x3	46 x11 0 x19 462 x27 0
x4	0 x12 10 x20 0 x28 0
x5	21 x13 49 x21 0 x29 0
x6	0 x14 0 x22 0 x30 0
x7	0 x15 10 x23 0 x31 0

## Обработка несуществующих переменных

```
let a: string = "hola";
print(aa);
```

```
Exception in thread "main" java.lang.RuntimeException Create breakpoint : Undefined variable 'aa'
at org.example.Scope.resolve(Scope.kt:25)
```

## Обработка ошибки в типах данных

```
let x:int = 3;
let y:string = "aaaa";
x <- y; // Ошибка: int <- string
```

```
Type mismatch in assignment to `x`: expected INT but got STRING
```

## Работа с областями видимости

```
let a: int = 0;

if (1) {
    let b: int = 1;
}

print(a);
print(b); // <- out of scope
```

```
Exception in thread "main" java.lang.RuntimeException: Create breakpoint : Undefined variable 'b'
    at org.example.Scope.resolve(Scope.kt:25)
```

## **Заключение**

В ходе выполнения данной работы был создан компилятор NLang на базе ANTLR4. Реализованы лексический и синтаксический анализ, построение AST, статическая типизация и генерация RISC V кода.