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 № P3324

Преподаватель:
Лаздин Артур Вячеславович

Санкт-Петербург
2025

Оглавление

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`

Полый 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)

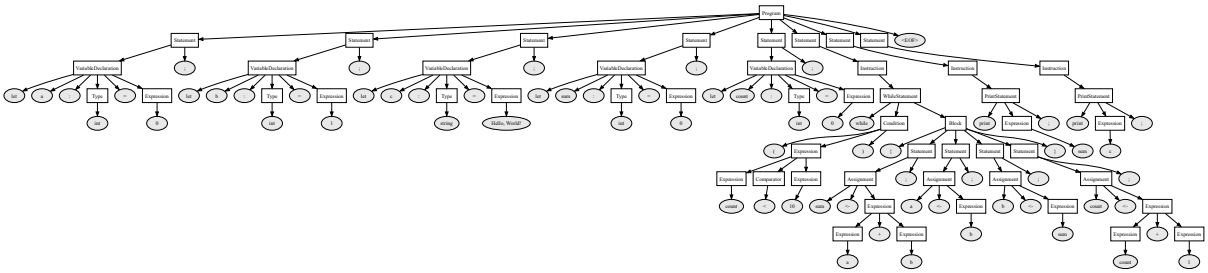
```
let a: int = 0;
let b: int = 1;
let c: string = "Hello, World!"; // single line comment

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;
```

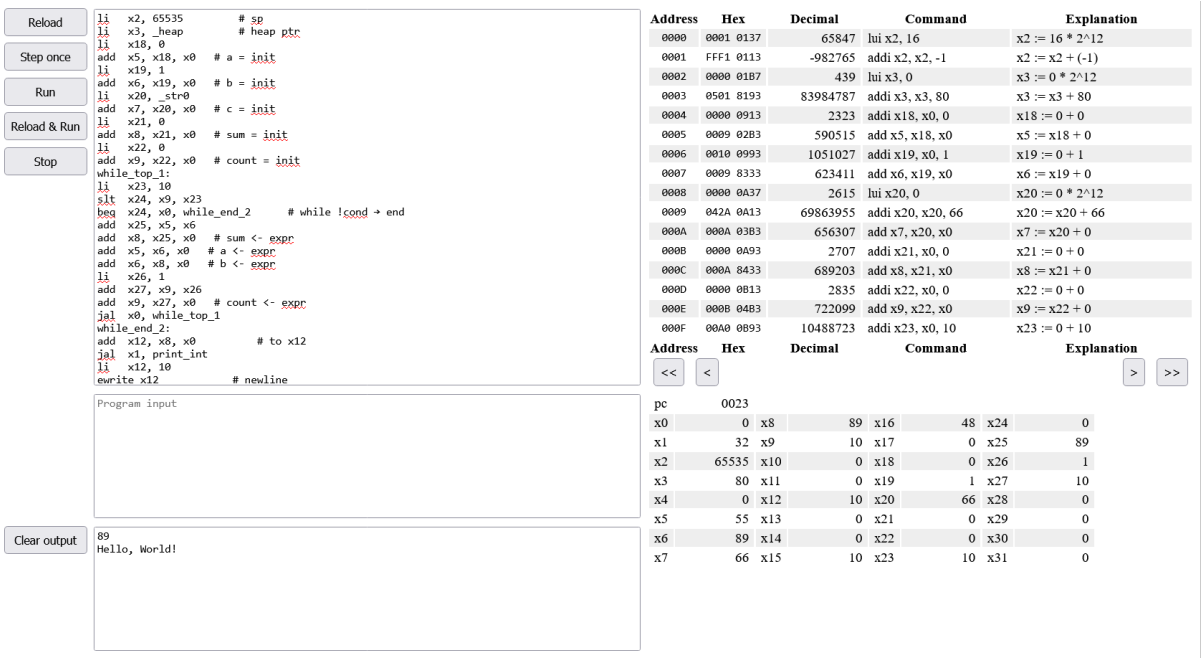
Результат работы

```
89
Hello, World!
```

Построенное дерево



Результат компиляции



НОД

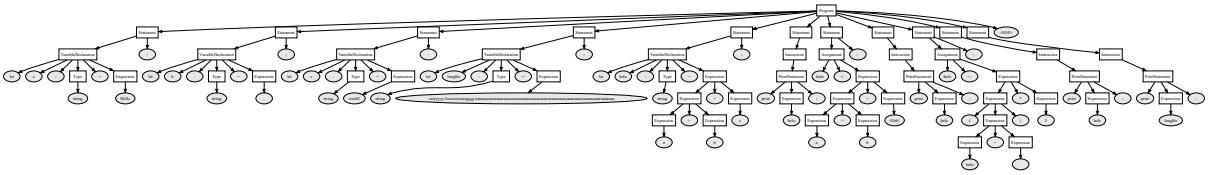
Исходный код программы для поиска наибольшего общего делителя

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

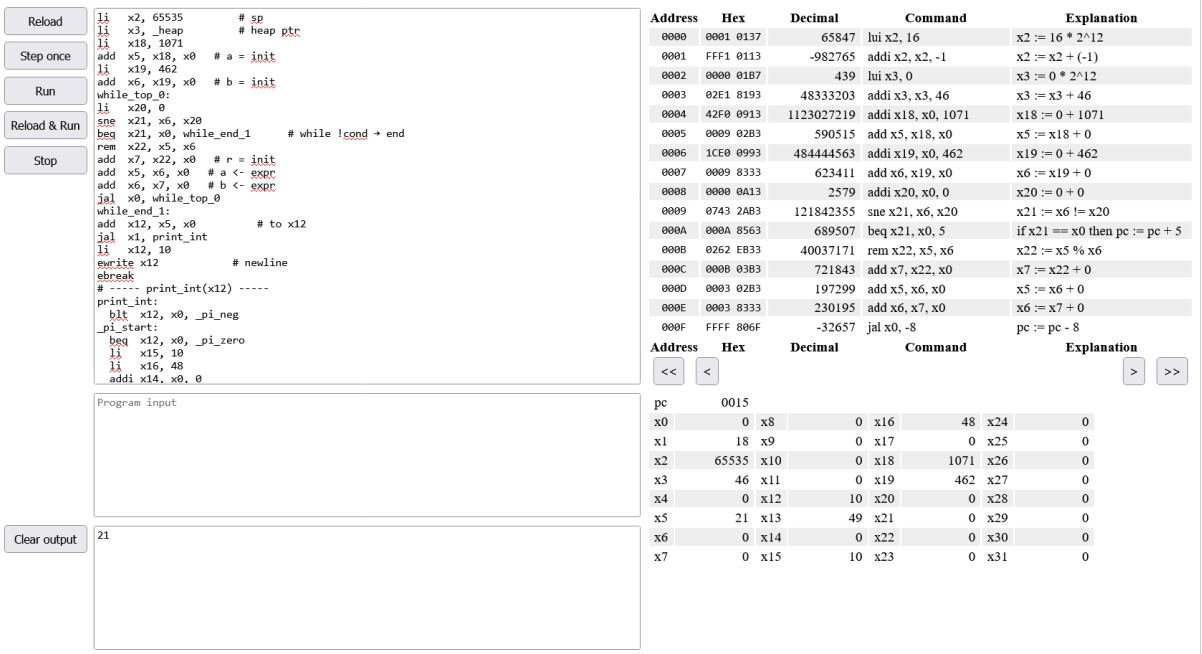
Результат работы

```
21
```

Построенное дерево



Результат компиляции



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.