

Министерство науки и высшего образования Российской Федерации
Национальный научно-исследовательский университет ИТМО
Факультет программной инженерии и компьютерной техники

Курсовая проектная работа
по дисциплине
“Разработка компиляторов”.

Работу выполнил:
Афанасьев Кирилл Александрович.
студент группы Р3306.
Преподаватель:
Лаздин Артур Вячеславович.

Санкт-Петербург, 2025

Содержание

Содержание.....	2
Задание.....	3
Описание работы.....	5
Особенности реализации.....	6
Структура проекта.....	6
CLI.....	6
Пример использования CLI.....	8
Примеры работы программ Wakizashi.....	8
test0.wak - Hello World!.....	8
test1.wak - Работа с функциями и бинарными операциями.....	9
test2.wak - Факториал!.....	9
test3.wak - Задача 1000 с сайта по программированию Timus Online Judge. https://acm.timus.ru/problem.aspx?space=1&num=1000	10
test4.wak - Задача 1607 с сайта по программированию Timus Online Judge. https://acm.timus.ru/problem.aspx?space=1&num=1607	10
Негативные сценарии работы Wakizashi.....	12
neg_test0.wak - Декларация строкового литерала в целочисленную переменную.....	12
neg_test1.wak - Попытка положить в целочисленный параметр строку.....	12
neg_test2.wak - Недостаточно переданных аргументов в функцию.....	13
neg_test3.wak - Попытка вытащить переменную из другой области видимости.....	13
Лексическое описание Wakizashi.....	15
Синтаксическое описание Wakizashi.....	17
Семантическое описание Wakizashi.....	24
Немного про родную среду исполнения и генерацию кода.....	25
Выводы.....	29

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 (ссылка на README в репозитории эмулятора <https://github.com/asurkis/risc-emulator>)~~. По согласованию с преподавателем язык целевой машины может быть изменен.

Язык целевой машины изменен - <https://github.com/Zerumi/csa3-140324-asm-stack>

Характеристики учебного языка программирования (императивный тыюринг полный язык), поддерживающий следующий набор операторов и выражений:

● Операторы:

1. Определения переменных. Поддерживаемые типы данных: целые числа со знаком, строки (можно ограничиться набором ASCII символов из первой половины таблицы коды от 32 до 127 и управляющими символами переноса строки и пр.).
2. ~~Присваивание~~ (может быть заменено выражением в случае определения оператора-выражения). *В языке нет присваивания, есть только декларирование.*
3. Ветвление (if) с факультативным else.
4. ~~Цикл while~~. *Вместо этого есть рекурсия.*
5. ~~Блок (составной оператор)~~. *Вместо этого есть функции.*
6. ~~Вывода (тип print)~~. *Вместо этого есть стандартная библиотека.*

● 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.
- 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.

На этом завершается первый этап, его результаты можно обсудить со мной, если возникнут проблемы.

Второй этап предполагает генерацию кода в процессе обхода AST.

Можно построить промежуточное представление программы в виде трехадресного кода, если вам это будет удобно. Трехадресный код не является целью проекта.

Цель второго этапа - получение ассемблерной программы эквивалентной тексту входной программы.

~~Проект можно выполнять в парах. Полученная на защите оценка ставится обоим участникам.~~ Выбор языка реализации оставляю за вами.

По итогам будет нужно оформить отчет с описанием лексического и синтаксического состава, демонстрация работы компилятора для корректных и ошибочных программ. (вопросы обработки ошибок обсудим).

Описание работы

Вместо тысячи слов, лучше это все-таки один раз увидеть:

<https://github.com/Zerumi-ITMO-Related/Wakizashi>

По формальностям:

- Язык **Wakizashi**: полнотьюринговый язык с элементами императивного и функционального программирования:
 - Имеется поддержка *деклараций значений*
 - Имеется поддержка *функций первого порядка*
 - Имеется поддержка *рекурсии*
 - Имеется поддержка *оператора if/else*
 - Реализована *стандартная библиотека*, присутствует простая interoperability с ассемблером
 - Имеются *типы данных*: *Int*, *String*, *Unit (void)*.
 - Строки представляются *указателем* на нуль-терминированный массив символов.
 - Остальные типы данных передаются *по значению*.
 - Имеется поддержка *бинарных операторов* для целочисленных данных
 - *Компилятор* для этого языка *написан в функциональном стиле!*
- В языке отсутствуют следующие компоненты:
 - *Нет присваивания*. Декларированные идентификаторы невозможно изменить в коде программы
 - *Нет циклов*. Вместо этого подразумевается использовать рекурсию
- На данный момент для упрощения действуют следующие ограничения:
 - Имя переменной должно быть уникальным для всей программы. Обращение к разным областям видимости предотвращается за счет проверки AST, но генерация кода не делает уникальных имен, поэтому потенциально, такое поведение считается опасным. То же самое справедливо для пересечений с именами из стандартной библиотеки.
 - В текст программы нужно добавлять много ненужной информации (например, `return Unit` в конце функции `main`). Да, вывод типов в языке есть, однако, как это уже случилось, весь бюджет ушел на реализацию этого самого вывода типов. Поэтому фронтенд компилятора не построит AST без явно выведенных типов.

Особенности реализации

- Стек:
 - *Фронтенд*: C/YACC (a.k.a. Flex/Bison)
 - *Бекенд*: Kotlin/JVM
 - *Эмулятор процессора*: Kotlin/JVM
- Фронтенд и бекенд компилятора обмениваются AST, сериализованным в JSON формате.
- Далее сгенерированный ассемблерный код статически линкуется со стандартной библиотекой, отдается транслятору
- Для удобства интеграции разработана еще одна вспомогательная программа
- Все это можно попробовать завести у себя:
<https://github.com/Zerumi-ITMO-Related/Wakizashi>

Структура проекта

Проект доступен на GitHub, и содержит в себе 6 основных модулей:

- lang-frontend
- lang-backend
- lang-compiler
- asm
- comp
- isa

Подробнее (с картинкой!) - доступно в [README.md](#) в репозитории

CLI

Чтобы максимально просто работать с Wakizashi, для него был разработан простой текстовый интерфейс:

```
waki [-h | --help] | [--with-input="String1 String2..."]  
[--show-ast] [--show-sasm] [--export-ast <filename>]  
[--export-sasm <filename>] [--export-machine <filename>]  
[--to-ast | --to-sasm] [--from-ast | --from-sasm] <input_file>
```

- Возможность просмотра промежуточных артефактов компиляции:
--show-*
- Возможность их экспорта в отдельный файл: --export-*
- Возможность регулировать точки входа и выхода. Например, можно начать компиляцию с ранее сгенерированного AST.

Пример использования CLI

```
Apple | ~/gitClone/Wakizashi/build/distribution | git develop !1 ?1 | ./waki /Users/zerumi/gitClone/Wakizashi/lang-frontend/input/test0.wak | 07:04:17
Hello, world!

Apple | ~/gitClone/Wakizashi/build/distribution | git develop !1 ?1 | ./waki --from-ast /Users/zerumi/gitClone/Wakizashi/lang-frontend/output/test0.ast | 07:06:52
Hello, world!
```

```
Apple | ~/gitClone/Wakizashi/build/distribution | git develop !1 | ./waki -h
Usage: waki-compiler [<options>] <file>

Compilation pipeline start:

Choose entry point of compiler

--from-ast                Start from AST
--from-assembly, --from-sasm Start from Stack Assembly

Compilation pipeline stop:

Choose leave point of compiler

-ast, --to-ast=<value>          Stop after generating AST
-assembly, -sasm, --to-assembly, --to-sasm=<value> Stop after generating SASM

Options:
--show-ast                Print the AST after parsing
--show-assembly, --show-sasm Print the generated stack assembly
--export-ast=<path>        Write AST to a file
--export-sasm=<path>       Write stack assembly to a file
--export-machine=<path>    Write machine code to a file
--with-input=<text>        Input some data to CPU I/O
-h, --help                Show this message and exit
```

Примеры работы программ Wakizashi

test0.wak - Hello World!

```
fun main() : Unit {
    print("Hello, world!");

    return Unit;
}
```

Результат работы программы:

```
Apple | ~/gitClone/Wakizashi/build/distribution | git develop !1 | ./waki /Users/zerumi/gitClone/Wakizashi/lang-frontend/input/test0.wak | 10:08:42
Hello, world!

Apple | ~/gitClone/Wakizashi/build/distribution | git develop | 10:27:01
```


test1.wak - Работа с функциями и бинарными операциями

```
fun is_equal(a: Int, b: Int) : Int {
    return a == b;
}

fun main() : Unit {
    val a1: Int = 40 - 20;
    val b1: Int = 20;
    val c: Int = is_equal(a1, b1);
    if (c && (a1 > 10)) {
        print("Equals and a1 more than 10");
    } else {
        print("Not Equals or a1 less/equal than 10");
    }
    return Unit;
}
```

Результат работы программы:



```
~/gitClone/Wakizashi/build/distribution | gk develop ./waki /Users/zerumi/gitClone/Wakizashi/lang-frontend/input/test1.wak | 10:27:01
Equals and a1 more than 10

~/gitClone/Wakizashi/build/distribution | gk develop | 10:28:39
```

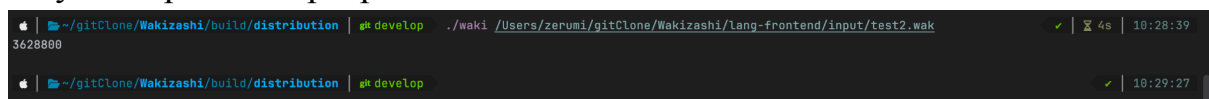
test2.wak - Факториал!

```
fun factorial(n : Int) : Int {
    if (n == 1) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }

    return 0;
}

fun main() : Unit {
    print_number(factorial(10));
    return Unit;
}
```

Результат работы программы:



```
~/gitClone/Wakizashi/build/distribution | gk develop ./waki /Users/zerumi/gitClone/Wakizashi/lang-frontend/input/test2.wak | 10:28:39
3628800

~/gitClone/Wakizashi/build/distribution | gk develop | 10:29:27
```

test3.wak - Задача 1000 с сайта по программированию Timus Online Judge.
<https://acm.timus.ru/problem.aspx?space=1&num=1000>

```
fun main() : Unit {
    val a: Int = parse_int(readln());
    val b: Int = parse_int(readln());

    print_number(a + b);

    return Unit;
}
```

Результат работы программы:



```
Apple | ~/gitClone/Wakizashi/build/distribution | develop | ./waki --with-input="2 2" /Users/zerumi/gitClone/Wakizashi/lang-frontend/input/test3.wak | 10:29:27
4
Apple | ~/gitClone/Wakizashi/build/distribution | develop | | 10:32:08
```

test4.wak - Задача 1607 с сайта по программированию Timus Online Judge.
<https://acm.timus.ru/problem.aspx?space=1&num=1607>

```
fun binarySearch(left: Int, right: Int, a: Int, b: Int, c:
Int, d: Int): Int {
    if ((right - left) < 2) {
        return left;
    } else {
        val mid : Int = (left + right) / 2;
        val taxiPrice : Int = c - (d * mid);
        val petyaPrice : Int = a + (b * mid);

        if (taxiPrice > petyaPrice) {
            return binarySearch(mid, right, a, b, c, d);
        } else {
            return binarySearch(left, mid, a, b, c, d);
        }
    }
    return 0;
}
```

```
fun main() : Unit {
    val a0: Int = parse_int(readln());
    val b0: Int = parse_int(readln());
    val c0: Int = parse_int(readln());
    val d0: Int = parse_int(readln());
```

```

val left_1: Int = binarySearch(0, 10000, a0, b0, c0, d0);
val right_1: Int = left_1 + 1;

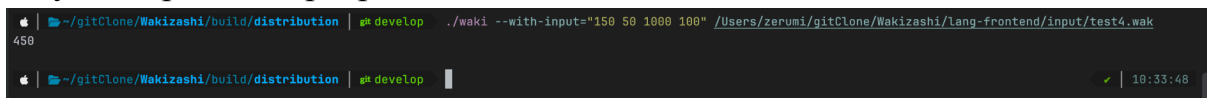
val lp: Int = a0 + (b0 * left_1);
val lt: Int = c0 - (d0 * left_1);
val rp: Int = a0 + (b0 * right_1);
val rt: Int = c0 - (d0 * right_1);

if (rp > lt) {
    if (lp > lt) {
        print_number(lp);
    } else {
        print_number(lt);
    }
} else {
    if (rp > rt) {
        print_number(rp);
    } else {
        print_number(rt);
    }
}

return Unit;
}

```

Результат работы программы:



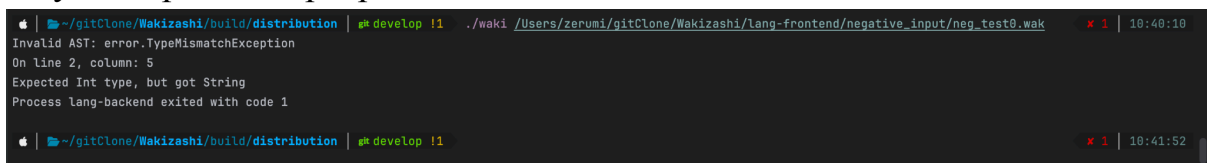
A terminal window showing the execution of a command. The prompt is `~/gitClone/Wakizashi/build/distribution` with the `develop` branch selected. The command executed is `./waki --with-input="150 50 1000 100" /Users/zerumi/gitClone/Wakizashi/lang-frontend/input/test4.wak`. The output is the number `450`. The terminal also shows the time `10:33:48` and a green checkmark icon.

Негативные сценарии работы Wakizashi

neg_test0.wak - Декларация строкового литерала в целочисленную переменную

```
fun main() : Unit {  
    val a: Int = "10";  
    print_number(a);  
  
    return Unit;  
}
```

Результат работы программы:



```
Apple | ~/gitClone/Wakizashi/build/distribution | develop !1 | ./waki /Users/zerumi/gitClone/Wakizashi/lang-frontend/negative_input/neg_test0.wak | 1 | 10:40:10  
Invalid AST: error.TypeMismatchException  
On line 2, column: 5  
Expected Int type, but got String  
Process lang-backend exited with code 1  
  
Apple | ~/gitClone/Wakizashi/build/distribution | develop !1 | 1 | 10:41:52
```

```
Invalid AST: error.TypeMismatchException  
On line 2, column: 5  
Expected Int type, but got String  
Process lang-backend exited with code 1
```

neg_test1.wak - Попытка положить в целочисленный параметр строку

```
fun getString(str_num: Int) : String {  
    if (str_num == 1) {  
        return "Hello, world!";  
    } else {  
        return "Goodbye, world!";  
    }  
  
    return "";  
}  
  
fun main() : Unit {  
    print(getString("1"));  
  
    return Unit;  
}
```

Результат работы программы:

```
~/gitClone/Wakizashi/build/distribution | gh develop !1 | ./waki /Users/zerumi/gitClone/Wakizashi/lang-frontend/negative_input/neg_test1.wak x 1 | 10:41:52
Invalid AST: error.TypeMismatchException
On line 12, column: 21
Expected Int type, but got String
Process lang-backend exited with code 1

~/gitClone/Wakizashi/build/distribution | gh develop !1 | x 1 | 10:42:21
```

Invalid AST: error.TypeMismatchException
On line 12, column: 21
Expected Int type, but got String
Process lang-backend exited with code 1

neg_test2.wak - Недостаточно переданных аргументов в функцию

```
fun sum(a: Int, b: Int) : Int {
    return a + b;
}

fun main() : Unit {
    print_number(sum(5));
}
```

Результат работы программы:

```
~/gitClone/Wakizashi/build/distribution | gh develop !1 | ./waki /Users/zerumi/gitClone/Wakizashi/lang-frontend/negative_input/neg_test2.wak x 1 | 10:42:21
Invalid AST: error.FunctionArgumentsCountMismatchException
On line 6, column: 18
During function call: sum | Expected: 2 arguments, actual: 1
Process lang-backend exited with code 1

~/gitClone/Wakizashi/build/distribution | gh develop !1 | x 1 | 10:42:47
```

Invalid AST: error.FunctionArgumentsCountMismatchException
On line 6, column: 18
During function call: sum | Expected: 2 arguments, actual: 1
Process lang-backend exited with code 1

neg_test3.wak - Попытка вытащить переменную из другой области видимости

```
fun val_a_holder() : Unit {
    val a: Int = 10;
    return Unit;
}

fun main() : Unit {
    print_number(a);
}
```

Результат работы программы:

```
🍏 | ~/gitClone/Wakizashi/build/distribution | g# develop | ./waki /Users/zerumi/gitClone/Wakizashi/lang-frontend/negative_input/neg_test3.wak | 10:33:48
Invalid AST: error.MissedDeclarationException
On line 7, column: 18
Declaration of a is not defined
Process lang-backend exited with code 1

🍏 | ~/gitClone/Wakizashi/build/distribution | g# develop !1 | 10:40:10
```

Invalid AST: error.MissedDeclarationException
On line 7, column: 18
Declaration of a is not defined
Process lang-backend exited with code 1

Лексическое описание Wakizashi

lexer.l:

```
%{
#include "ast.h"
#include "parser.h"
#include <stdlib.h>
#include <string.h>

extern YYLTYPE yylloc;

extern int yylineno;
int yycolumn = 0;

void update_location() {
    yylloc.first_line = yylineno;
    yylloc.first_column = yycolumn;
    yylloc.last_line = yylineno;
    yylloc.last_column = yycolumn + yyleng - 1;
    yycolumn += yyleng;
}
%}

%x COMMENT

%%

"val"          { update_location(); return VAL; }
"fun"          { update_location(); return FUN; }
"if"           { update_location(); return IF; }
"else"         { update_location(); return ELSE; }
"return"       { update_location(); return RETURN; }

"Int"          { update_location(); return INT; }
"Boolean"      { update_location(); return BOOLEAN; }
"String"       { update_location(); return STRING; }
"Unit"         { update_location(); return UNIT; }

[0-9]+         {
                update_location();
                yylval.sval = strdup(yytext);
                return LIT_INT;
            }

\"[^\"]*\"      {
                update_location();
                yylval.sval = strdup(yytext);
```

```

        return LIT_STRING;
    }

    "true"|"false" {
        update_location();
        yylval.sval = strdup(yytext);
        return LIT_BOOLEAN;
    }

    [a-zA-Z_][a-zA-Z0-9_]* {
        update_location();
        yylval.sval = strdup(yytext);
        return IDENT;
    }

    "=" { update_location(); return ASSIGN; }
    ":" { update_location(); return COLON; }
    ";" { update_location(); return SEMICOLON; }
    "," { update_location(); return COMMA; }
    "(" { update_location(); return LPAREN; }
    ")" { update_location(); return RPAREN; }
    "{" { update_location(); return LBRACE; }
    "}" { update_location(); return RBRACE; }
    "+" { update_location(); return PLUS; }
    "-" { update_location(); return MINUS; }
    "*" { update_location(); return MUL; }
    "/" { update_location(); return DIV; }
    "<" { update_location(); return LESS; }
    ">" { update_location(); return MORE; }
    "==" { update_location(); return EQUAL; }
    "!=" { update_location(); return NOTEQUAL; }
    "&&" { update_location(); return AND; }
    "||" { update_location(); return OR; }

    [ \t\r]+ {
        int i;
        for (i = 0; i < yyleng; i++) {
            yycolumn += (yytext[i] == '\t') ? 4 : 1;
        }
    }

    \n {
        yylineno++;
        yycolumn = 0;
    }

    . {
        update_location();
    }

```



```

        fprintf(stderr, "Ошибка: неожиданный символ '%s'
на строке %d, колонке %d\n", yytext, yylineno, yycolumn);
        exit(1);
    }

```

```
%%
```

```

int yywrap() {
    return 1;
}

```

Синтаксическое описание Wakizashi

parser.y:

```
%locations
```

```

%{
#include "ast.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

```

extern int yylex();
void yyerror(const char *s);

```

```
#define YYERROR_VERBOSE 1
```

```

extern int yylineno;
extern char* yytext;

```

```

extern ASTNode* ast_root;
%}

```

```

%union {
    char* sval;
    ASTNode* node;

    struct {
        char **names;
        char **types;
        size_t count;
    } params;
}

```

```

%token <sval> IDENT
%token <sval> LIT_INT LIT_STRING

```

```

%token <sval> LIT_BOOLEAN UNIT

%token VAL FUN RETURN
%token INT BOOLEAN STRING
%token IF ELSE
%token COLON SEMICOLON COMMA
%token LPAREN RPAREN LBRACE RBRACE LESS MORE EQUAL NOTEQUAL
%token ASSIGN PLUS MINUS MUL DIV AND OR

%type <sval> type
%type <params> param param_list

%type <node> program statements statement declaration_statement
%type <node> expression if_statement function_call return_statement
%type <node> expression_list block

%right '='
%left '+' '-'
%left '*' '/' '%'
%left '.'

%start program

%%

program
: statements { ast_root = $1; }
;

statements
: statement statements
{
    ASTNode *program;
    if ($1→type == NODE_PROGRAM) {
        program = $1;
    } else {
        program = create_program_node(@1.first_line,
@1.first_column);
        add_child(program, $1);
    }
    if ($2) {
        for (size_t i = 0; i < $2→block.children.size; ++i) {
            add_child(program, $2→block.children.items[i]);
        }
    }
    $$ = program;
}
| statement

```

```

    {
        ASTNode *program = create_program_node(@1.first_line,
@1.first_column);
        add_child(program, $1);
        $$ = program;
    }
;

/* fun main() : Unit { ... } */
statement
    : declaration_statement      { $$ = $1; }
    | if_statement               { $$ = $1; }
    | function_call SEMICOLON    { $$ = $1; }
    | return_statement SEMICOLON { $$ = $1; }
;

/* factorial(2 + 2) */
function_call
    : IDENT LPAREN expression_list RPAREN
    {
        $$ = create_function_call($1, $3→function_call.arguments,
$3→function_call.arg_count, @1.first_line, @1.first_column);
        free($3);
    }
;

/* factorial(2 + 2, 3) */
expression_list
    : expression
    {
        $$ = create_expression_list($1);
    }
    | expression_list COMMA expression
    {
        $$ = append_expression_list($1, $3);
    }
    | /* nothing */
    {
        $$ = create_empty_expression_list();
    }
;

/* return 2 + 2 */
return_statement
    : RETURN expression      { $$ = create_return_statement($2,
@1.first_line, @1.first_column); }
;

```

```

/* val a: Int = 4 */
declaration_statement
    : VAL IDENT COLON type ASSIGN expression SEMICOLON {
        $$ = create_variable_declaration($2, $4, $6, @1.first_line,
@1.first_column);
    }
    | FUN IDENT LPAREN param_list RPAREN COLON type block
    {
        $$ = create_function_declaration(
            $2,
            $4.names,
            $4.types,
            $4.count,
            $7,
            $8,
            @1.first_line,
            @1.first_column
        );
    }
;

/* if (bool) { ... } else { ... } */
if_statement
    : IF LPAREN expression RPAREN block
    {
        $$ = create_if_node($3, $5, NULL, @1.first_line,
@1.first_column);
    }
    | IF LPAREN expression RPAREN block ELSE block
    {
        $$ = create_if_node($3, $5, $7, @1.first_line,
@1.first_column);
    }
;

/* 2 + 2 */
expression
    : expression PLUS expression
    {
        $$ = create_binary_operation("+", $1, $3, @2.first_line,
@2.first_column);
    }
    | expression MINUS expression
    {
        $$ = create_binary_operation("-", $1, $3, @2.first_line,
@2.first_column);
    }

```

```

    | expression MUL expression
    {
        $$ = create_binary_operation("*", $1, $3, @2.first_line,
@2.first_column);
    }
    | expression DIV expression
    {
        $$ = create_binary_operation("/", $1, $3, @2.first_line,
@2.first_column);
    }
    | expression LESS expression
    {
        $$ = create_binary_operation("<", $1, $3, @2.first_line,
@2.first_column);
    }
    | expression MORE expression
    {
        $$ = create_binary_operation(">", $1, $3, @2.first_line,
@2.first_column);
    }
    | expression EQUAL expression
    {
        $$ = create_binary_operation("=", $1, $3, @2.first_line,
@2.first_column);
    }
    | expression NOTEQUAL expression
    {
        $$ = create_binary_operation("≠", $1, $3, @2.first_line,
@2.first_column);
    }
    | expression AND expression
    {
        $$ = create_binary_operation("&&", $1, $3, @2.first_line,
@2.first_column);
    }
    | expression OR expression
    {
        $$ = create_binary_operation("||", $1, $3, @2.first_line,
@2.first_column);
    }
    | LPAREN expression RPAREN
    {
        $$ = $2; // просто возвращаем выражение внутри скобок
    }
    | function_call
    {
        $$ = $1; // возвращаем вызов функции
    }

```

```

    | LIT_INT
    {
        $$ = create_literal($1, "Int", @1.first_line,
@1.first_column);
    }
    | LIT_STRING
    {
        $$ = create_literal($1, "String", @1.first_line,
@1.first_column);
    }
    | LIT_BOOLEAN
    {
        $$ = create_literal($1, "Boolean", @1.first_line,
@1.first_column);
    }
    | UNIT
    {
        $$ = create_literal($1, "Unit", @1.first_line,
@1.first_column); // создаем узел для Unit
    }
    | IDENT
    {
        $$ = create_identifier_node($1, @1.first_line,
@1.first_column); // создаем узел переменной
    }
;

/* a: Int, b: String */
param_list
: // пустой список
{
    $$ .names = NULL;
    $$ .types = NULL;
    $$ .count = 0;
}
| param
{
    $$ .names = malloc(sizeof(char*));
    $$ .types = malloc(sizeof(char*));
    $$ .names[0] = $1.names[0];
    $$ .types[0] = $1.types[0];
    $$ .count = 1;
}
| param_list COMMA param
{
    size_t n = $1.count + 1;
    $$ .names = realloc($1.names, sizeof(char*) * n);

```

```

        $$types = realloc($1types, sizeof(char*) * n);
        $$names[n - 1] = $3names[0];
        $$types[n - 1] = $3types[0];
        $$count = n;
    }
;

/* a: Int */
param
: IDENT COLON type
{
    $$names = malloc(sizeof(char*));
    $$types = malloc(sizeof(char*));
    $$names[0] = strdup($1);
    $$types[0] = strdup($3);
    $$count = 1;
}
;

/* { ... } */
block
: LBACE statements RBACE
{
    $$ = create_block_node(@1.first_line, @1.first_column);
    if ($2 && ($2->type == NODE_PROGRAM || $2->type ==
NODE_BLOCK)) {
        NodeList* children = &$2->block.children;
        for (size_t i = 0; i < children->size; i++) {
            add_child($$, children->items[i]);
        }
        children->size = 0;
        free_node($2);
    }
}
;

/* Int */
type
: INT      { $$ = strdup("Int"); }
| BOOLEAN { $$ = strdup("Boolean"); }
| STRING  { $$ = strdup("String"); }
| UNIT    { $$ = strdup("Unit"); }
;

%%

// Получение корня AST
ASTNode* get_ast_root() {

```

```

        return ast_root;
    }

void yyerror(const char *s) {
    fprintf(stderr, "Syntax error at line %d: %s near '%s'\n",
        yylineno, s, yytext);
}

```

Семантическое описание Wakizashi

Обходим AST дерево по шаблону “Посетитель”. Обход производится дважды

- Сначала для проверки валидности предоставленного нам дерева
- Затем для генерации ассемблерного кода .sasm

Пример

<https://github.com/Zerumi-ITMO-Related/Wakizashi/blob/master/lang-backend/src/main/kotlin/semantic/SemanticASTVisitor.kt>

```

fun checkASTSemantic(ast: ASTNode): Result<SemanticContext> =
    ASTVisitor(
        visitProgramNode = ::visitProgramNode,
        visitFunctionDeclarationNode = ::visitFunctionDeclarationNode,
        visitReturnNode = ::visitReturnNode,
        visitBlockNode = ::visitBlockNode,
        visitValueDeclarationNode = ::visitValueDeclarationNode,
        visitBinaryOperationNode = ::visitBinaryOperationNode,
        visitFunctionCallNode = ::visitFunctionCallNode,
        visitIdentNode = ::visitIdentNode,
        visitIfNode = ::visitIfNode,
        visitLiteralNode = ::visitLiteralNode,
        visitUnknownNode = ::visitUnknownNode
    ).visitAST(ast, stdlibContext())

fun visitFunctionCallNode(
    ast: ASTNode.FunctionCallNode, state: SemanticContext,
    astVisitor: ASTVisitor<SemanticContext>
): Result<SemanticContext> {
    ast.args.fold(Result.success(state)) { ctx, child →
        ctx.fold(onSuccess = { astVisitor.visitAST(child, it) },
            onFailure = { ctx })
    }.onFailure { return Result.failure(it) }
    val callingFunction = state.functions.find { it.name == ast.name }
    return Result.failure(
        MissedDeclarationException(

```



```

        ast.line,
        ast.column,
        ast.name,
    )
)
if (ast.args.size != callingFunction.params.size) return
Result.failure(
    FunctionArgumentsCountMismatchException(
        ast.line,
        ast.column,
        ast.name,
        callingFunction.params.size,
        ast.args.size,
    )
)
ast.args.zip(callingFunction.params).map { (arg, param) →
    val actualType = inferType(arg, state).getOrElse { return
Result.failure(it) }
    val expectedType = param.type
    if (!expectedType.equals(actualType, ignoreCase = true))
return Result.failure(
        TypeMismatchException(
            arg.line,
            arg.column,
            expectedType,
            actualType
        )
    )
}
return Result.success(state)
}

```

Немного про родную среду исполнения и генерацию кода

Это самописный эмулятор стекового процессора. Имеет два стека: стек данных и стек возврата из функций. Язык Wakizashi в первой итерации транслируется в ассемблер стекового процессора. Вот пример генерации функции (по тому же шаблону Посетитель):

```

fun generateFunctionBody(
    node: ASTNode,
    codegenState: CodegenContext,
    currentFunction: ASTNode.FunctionDeclarationNode,
    state: List<String> = emptyList()
): Result<List<String>> {
    return when (node) {

```

```

        is ASTNode.BlockNode → {
            return node.children.fold(Result.success(state)) { ctx,
child →
                ctx.fold(
                    onSuccess = { generateFunctionBody(child,
codegenState, currentFunction, it) },
                    onFailure = { Result.failure(it) }
                )
            }
        }

        is ASTNode.BinaryOperationNode → {
            val left = generateFunctionBody(node.left, codegenState,
currentFunction).fold(
                onSuccess = { it },
                onFailure = { return Result.failure(it) }
            )
            val right = generateFunctionBody(node.right,
codegenState, currentFunction).fold(
                onSuccess = { it },
                onFailure = { return Result.failure(it) }
            )
            val op = when (node.op) {
                "+" → "lit add"
                "-" → "lit sub"
                "*" → "lit mul"
                "/" → "lit div"
                "=" → "lit equals"
                "≠" → "lit not_equals"
                ">" → "lit more"
                "<" → "lit less"
                "&&" → "lit and"
                "||" → "lit or"
                else → return
Result.failure(UnknownOperationException(node.line, node.column))
            }
            Result.success(state.plus(listOf(left,
right).flatten().plus(op).plus("call")))
        }

        is ASTNode.FunctionCallNode → {
            val args =
node.args.reversed().fold(Result.success(emptyList<String>())) {
ctx, child →
                ctx.fold(
                    onSuccess = { generateFunctionBody(child,
codegenState, currentFunction, it) },
                    onFailure = { Result.failure(it) }
                ).fold(

```

```

        onSuccess = { it },
        onFailure = { return Result.failure(it) }
    )
    val functionDeclaration = "lit ${node.name}"
    val call = "call"

Result.success(state.plus(args.plus(functionDeclaration).plus(call))
)

    }

    is ASTNode.IdentNode → Result.success(state.plus(listOf("lit
${node.name}", "load")))
    is ASTNode.IfNode → {
        val thenBlock = generateFunctionBody(node.then,
codegenState, currentFunction).fold(
            onSuccess = { it },
            onFailure = { return Result.failure(it) }
        )
        val elseBlock = node.`else`?.let {
            generateFunctionBody(it, codegenState,
currentFunction).fold(
                onSuccess = { it },
                onFailure = { return Result.failure(it) }
            )
        } ?: emptyList()
        val condition = generateFunctionBody(node.condition,
codegenState, currentFunction).fold(
            onSuccess = { it },
            onFailure = { return Result.failure(it) }
        )

        val uuid = UUID.randomUUID()
        val elseLabel = "else-$uuid"
        val continueLabel = "continue-$uuid"

        Result.success(
            state.plus(
                listOf("lit $elseLabel")
                    .asSequence()
                    .plus(condition)
                    .plus(
                        listOf(
                            "jz",
                        )
                    )
                )
            .plus(thenBlock)
            .plus(
                listOf(

```

```

        "lit $continueLabel",
        "jump",
        "$elseLabel:"
    )
    )
    .plus(elseBlock)
    .plus(
        listOf(
            "$continueLabel:",
            "nop"
        )
    )
    .toList()
    )
    )
}

is ASTNode.ReturnNode → {
    val initReturn = generateFunctionBody(node.value,
codegenState, currentFunction).fold(
        onSuccess = { it },
        onFailure = { return Result.failure(it) }
    )
    Result.success(state.plus(initReturn).plus("ret"))
}

is ASTNode.LiteralNode → {
    val literal = codegenState.literals.find { it.value ==
node.value } ?: return Result.failure(
        MissedLiteralException(
            node.line,
            node.column
        )
    )
    Result.success(state.plus(listOf("lit ${literal.label}",
"load"))))
}

is ASTNode.ValueDeclarationNode → {
    val identifier = "lit init_${node.name}"
    Result.success(state.plus(listOf(identifier, "call")))
}

is ASTNode.UnknownNode →
Result.failure(UnknownNodeInASTException(node.line, node.column))

```

```
        is ASTNode.ProgramNode →  
Result.failure(UnsupportedOperationException(node.line,  
node.column))  
        is ASTNode.FunctionDeclarationNode →  
Result.failure(UnsupportedOperationException(node.line,  
node.column))  
    }  
}
```

Выводы

Во время выполнения данной проектной работы я ознакомился с принципами построения компиляторов. Я изучил этапы трансляции исходного кода, включая лексический и синтаксический анализ, построение абстрактного синтаксического дерева, семантический анализ и генерацию кода. В результате работы был получен компилятор учебного языка программирования, способный даже решать некоторые серьезные алгоритмические задачи.