ДЗ 4

Студент: Султанов Артур Радикович (367553), группа P3313

Грамматика

Синтаксически язык имеет сходства с Go, Python, C.

```
lexer grammar LangLexer;

// Keywords
IF:    'if';
ELSE:  'else';
WHILE: 'while';
PRINT: 'print';

// Relation operators
EQUALS:     '==';
NOT_EQUALS: '!=';

// Punctuation
L_PAREN:   '(';
R_PAREN:   ')';
L_CURLY:   '{';
R_CURLY:   '}';
ASSIGN:    '=';
SEMICOLON: ';';

// Number literals
DECIMAL_LITERAL: [+-]?[1-9][0-9]* | '0';

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

// Arithmetic operators
PLUS:  '+';
MINUS: '-';
MUL:   '*';
DIV:   '/';

// Whitespaces
WS:  [ \t]+  -> channel(HIDDEN);
EOL: [\r\n]+;
```

```
parser grammar LangParser;

options {
    tokenVocab = LangLexer;
}
```

```
prog
    : stmt (stmt)* EOF
    ;

stmtList
    : (stmt)*
    ;

stmt
    : assignStmt ((EOL | SEMICOLON)+ | EOF)
    | ifStmt ((EOL | SEMICOLON)+ | EOF)
    | whileStmt ((EOL | SEMICOLON)+ | EOF)
    | printStmt ((EOL | SEMICOLON)+ | EOF)
    ;

expr
    : IDENTIFIER                          # ExprIdentifier
    | DECIMAL_LITERAL                     # ExprDecimalLiteral
    | expr op=(EQUALS | NOT_EQUALS) expr  # ExprCompare
    | expr op=(MUL | DIV) expr            # ExprMulDiv
    | expr op=(PLUS | MINUS) expr         # ExprAddSub
    | L_PAREN expr R_PAREN                # ExprParentheses
    ;

block
    : L_CURLY EOL stmtList R_CURLY
    ;

ifStmt
    : IF L_PAREN cond=expr R_PAREN block                              #
IfStmtIf
    | IF L_PAREN cond=expr R_PAREN ifBlock=block ELSE elseBlock=block #
IfStmtIfElse
    ;

assignStmt
    : IDENTIFIER ASSIGN expr
    ;

whileStmt
    : WHILE L_PAREN cond=expr R_PAREN block
    ;

printStmt
    : PRINT L_PAREN expr R_PAREN
    ;
```

Ключевые слова:

- print
- while

- `if/else`

Операции:

- `+`, `-`
- `*`, `/`
- `=`
- `==`, `!=`

Единственный тип данных - целое число (int).

Statement-ы разделяются новыми строками и/или точками с запятой (`;`).

Задание/присвоение переменных неразличимо (подобно python). `=` обновит значение переменной, если она определена в текущей области видимости или одной из областей видимости выше. Если такой переменной нет, он создаст ее в текущей области.

Реализация

Исходный код доступен на Github.

Смысловая часть:

```
std::any EvalVisitor::visitProg(LangParser::ProgContext *ctx) {
    auto res = Result(0);
    this->scopeDepth = 0;
    this->scopes.push_back(std::unordered_map<std::string, int>());
    for (auto stmt : ctx->stmt()) {
        auto stmtRes = visitWithRes(stmt);
        if (!stmtRes) {
            res = stmtRes;
            break;
        }
    }
    this->scopes.pop_back();
    this->scopeDepth = 0;
    return res;
}

std::any EvalVisitor::visitStmtList(LangParser::StmtListContext *ctx) {
    for (auto stmt : ctx->stmt()) {
        auto res = visitWithRes(stmt);
        if (!res) {
            return res;
        }
    }
    return Result(0);
}

std::any EvalVisitor::visitStmt(LangParser::StmtContext *ctx) {
    if (auto stmt = ctx->assignStmt()) {
        return visit(stmt);
```

```
    }
    if (auto stmt = ctx->ifStmt()) {
        return visit(stmt);
    }
    if (auto stmt = ctx->whileStmt()) {
        return visit(stmt);
    }
    if (auto stmt = ctx->printStmt()) {
        return visit(stmt);
    }
    return std::unexpected(formatError(ctx->start, "unexpected statement
type"));
}

std::any EvalVisitor::visitBlock(LangParser::BlockContext *ctx) {
    this->scopeDepth++;
    this->scopes.push_back(std::unordered_map<std::string, int>());
    auto res = visit(ctx->stmtList());
    this->scopes.pop_back();
    this->scopeDepth--;
    return res;
}

std::any EvalVisitor::visitExprAddSub(LangParser::ExprAddSubContext *ctx) {
    auto aRes = visitWithRes(ctx->expr(0));
    if (!aRes) {
        return aRes;
    }
    auto a = aRes.value();

    auto bRes = visitWithRes(ctx->expr(1));
    if (!bRes) {
        return bRes;
    }
    auto b = bRes.value();

    Result result;
    switch (ctx->op->getType()) {
        case LangParser::PLUS: {
            result = a + b;
            break;
        }
        case LangParser::MINUS: {
            result = a - b;
            break;
        }
        default: {
            result = std::unexpected(formatError(ctx->op, "unexpected
operation (+, - expected)"));
            break;
        }
    }

    return result;
```

```
}

std::any EvalVisitor::visitExprIdentifier(LangParser::ExprIdentifierContext
*ctx) {
    auto varName = ctx->IDENTIFIER()->getText();

    auto maybeValue = getVariableValue(varName);

    Result result;
    if (maybeValue.has_value()) {
        result = maybeValue.value();
    } else {
        result = std::unexpected(formatError(ctx->getStart(), "undefined
variable: " + varName));
    }
    return result;
}

std::any EvalVisitor::visitExprMulDiv(LangParser::ExprMulDivContext *ctx) {
    auto aRes = visitWithRes(ctx->expr(0));
    if (!aRes) {
        return aRes;
    }
    auto a = aRes.value();

    auto bRes = visitWithRes(ctx->expr(1));
    if (!bRes) {
        return bRes;
    }
    auto b = bRes.value();

    Result result;
    switch (ctx->op->getType()) {
        case LangParser::MUL: {
            result = a * b;
            break;
        }
        case LangParser::DIV: {
            result = a / b;
            break;
        }
        default: {
            result = std::unexpected(formatError(ctx->op, "unexpected
operation (*, / expected)"));
            break;
        }
    }

    return result;
}

std::any EvalVisitor::visitExprCompare(LangParser::ExprCompareContext *ctx)
{
    auto aRes = visitWithRes(ctx->expr(0));
```

```
    if (!aRes) {
        return aRes;
    }
    auto a = aRes.value();

    auto bRes = visitWithRes(ctx->expr(1));
    if (!bRes) {
        return bRes;
    }
    auto b = bRes.value();

    Result result;
    switch (ctx->op->getType()) {
        case LangParser::EQUALS: {
            result = a == b;
            break;
        }
        case LangParser::NOT_EQUALS: {
            result = a != b;
            break;
        }
        default: {
            result = std::unexpected(formatError(ctx->op, "unexpected
operation (==, != expected)"));
            break;
        }
    }
    return result;
}

std::any
EvalVisitor::visitExprParentheses(LangParser::ExprParenthesesContext *ctx)
{
    return visit(ctx->expr());
}

std::any
EvalVisitor::visitExprDecimalLiteral(LangParser::ExprDecimalLiteralContext
*ctx) {
    auto rawVal = ctx->DECIMAL_LITERAL()->getText();
    return Result(std::atoi(rawVal.c_str()));
}

std::any EvalVisitor::visitIfStmtIf(LangParser::IfStmtIfContext *ctx) {
    auto condResRes = visitWithRes(ctx->expr());
    if (!condResRes) {
        return condResRes;
    }
    auto condRes = condResRes.value();

    if (condRes) {
        return visit(ctx->block());
    }
    return Result(0);
```

```
std::any EvalVisitor::visitPrintStmt(LangParser::PrintStmtContext *ctx) {
    auto valueRes = visitWithRes(ctx->expr());
    if (!valueRes) {
        return valueRes;
    }
    auto value = valueRes.value();
    std::cout << value << std::endl;

    return Result(0);
}
```

Примеры использование

Простой пример на `print` и взятие expression в скобки:

```
$ cat examples/test.prog
print((1))
$ ./lang.elf examples/test.prog
1
```

Демонстрация поддержи разделения statement-ов с помощью новых строк и `;`:

```
$ cat examples/silly.prog
a = 4; b = 2
c = a * b + 1;
d = 0
if (c == 9) {
    print(-123)
} else {
    print(456)

    while (c != 0) {
        c = c - 1
        print(c)
    }
}
$ ./lang.elf examples/silly.prog
-123
```

30-ое число Фибоначчи:

```
$ cat examples/fibonacci.prog
n = 30

cntr = 1
a = 0
b = 1
```

```
if (n == 0) {
    print(0)
} else {
    while (cntr != n) {
        tmp = b
        b = a + b
        a = tmp

        cntr = cntr + 1
    }
    print(b)
}
$ ./lang.elf examples/fibonacci.prog
832040
```

Неопределенная переменная:

```
$ cat examples/undeclared_variable.prog
print(a)
$ ./lang.elf examples/undeclared_variable.prog
examples/undeclared_variable.prog:1:7: undefined variable: a
```

Области видимости переменных:

```
$ cat examples/scopes.prog
a = 4
if (1) {
    b = 3
}
print(b)
$ ./lang.elf examples/scopes.prog
examples/scopes.prog:5:7: undefined variable: b
```