

Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ
ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ”



Лабораторная работа по дисциплине
“Основы разработки компиляторов”

ФИО студентов: Железнов Никита Сергеевич

Силантьева Анна Олеговна

Учебная группа: P34212

Направление подготовки (специальность): 09.03.04

ФИО преподавателя: Лаздин Артур Вячеславович

г. Санкт-Петербург, 2024

Задание:

Разработать язык программирования, который должен реализовать следующие компоненты:

- Присваивание (оператор или операция), арифметические и логические операции с возможностью изменения приоритета – скобочные формы.
- Ветвление, включая вариант факультативного `else`.
- Цикл `while`.
- Поддержка целочисленного и логического типа данных.
- Вывод значений переменных и констант;
- Многострочные комментарии в стиле Си-подобных языков.

```
/* **** строки комментариев **** */
/***** */
```

Это минимальные требования (на 4С) – и это до защиты.

Что добавит баллы:

Вместо ветвление `if [then] else` конструируется оператор `if elif [elif]+ else`

Вместо цикла `while` (или в дополнение к нему) конструируется цикл `for`.

Реализуется вывод значений.

Одно дополнение позволяет получить 4В, любые два 5А.

Приветствуется работа со строками и введение дополнительных операций: возведение в степень, получение остатка от деления и другие.

Выполнение:

Весь исходный код находится в репозитории:

<https://github.com/vkidofdarkness/compiler>

Разработка описания языка (ANTLR4).

Лексический состав языка

Для описания лексем языка были реализованы ключевые слова, операторы, идентификаторы, числа, строки и комментарии. Лексемы описаны в файле RusLang.g4:

```
7 // Ключевые слова
8 ЦЕЛ      : 'цел';      // целочисленный тип
9 ЛОГ      : 'лог';      // логический тип
10 СТР     : 'стр';      // строковый тип
11 ЕСЛИ     : 'если';
12 ТОГДА    : 'тогда';
13 ИНАЧЕЕСЛИ : 'иначеесли';
14 ИНАЧЕ    : 'иначе';
15 ПОКА     : 'пока';
16 ВЫВЕСТИ  : 'вывести';
17 И        : 'и';
18 ИЛИ      : 'или';
19 ИСТИНА   : 'истина';
20 ЛОЖЬ     : 'ложь';
21
22 // Символы
23 LPAREN   : '(';
24 RPAREN   : ')';
25 LBRACE   : '{';
26 RBRACE   : '}';
27 SEMICOLON : ';';
28 COMMA    : ',';
29 ASSIGN   : '=';
30 PLUS     : '+';
31 MINUS    : '-';
32 MUL      : '*';
33 DIV      : '/';
34 MOD      : '%';
35 LT       : '<';
36 GT       : '>';
37 LE       : '<=';
38 GE       : '>=';
39 EQ       : '==';
40 NEQ      : '!=';
```

```
47 // Идентификатор: русские/латинские буквы + _
48 IDENT
49   : [a-яA-ЯёЁa-zA-Z_] [a-яA-ЯёЁa-zA-Z_0-9]*
50   ;
51
52 // Целые числа
53 INT_NUMBER
54   : [0-9]+
55   ;
56
57 // Строка в двойных кавычках
58 STRING
59   : '"' (~["\\] | '\\' .)* '"'
60   ;
61
62 // Пропуск пробелов и переводов строк
63 WS
64   : [ \t\r\n]+ -> skip
65   ;
66
67 // Многострочные комментарии
68 COMMENT
69   : '/*' .*? '*/' -> skip
70   ;
71
72 // Однострочные комментарии (C++-подобные)
73 LINE_COMMENT
74   : '//' ~[\r\n]* -> skip
75   ;
```

Синтаксические правила

Грамматика поддерживает базовые конструкции:

1. **Объявление переменных:** цел a, b;
2. **Присваивание:** a = 10;
3. **Сравнение:** a > b;
4. **Условные операторы:** если, иначеесли, иначе.
5. **Циклы:** пока.
6. **Вывод:** вывести(a + b);

Пример описания конструкции “если”:

```
119 // Ветвление: если (expr) тогда { ... } иначеесли { ... } иначе { ... }
120 ifStatement
121 | : ЕСЛИ LPAREN expr RPAREN ТОГДА blockStatement (иначеЕслиPart)* (иначеPart)?
122 | ;
123
124 иначеЕслиPart
125 | : ИНАЧЕЕСЛИ LPAREN expr RPAREN ТОГДА blockStatement
126 | ;
127
128 // Факультативное иначе
129 иначеPart
130 | : ИНАЧЕ blockStatement
131 | ;
```

С помощью ANTLR генерируем парсер и лексер для обработки кода (для Python в нашем случае):

```
java -jar antlr-4.13.2-complete.jar -Dlanguage=Python3 -visitor RusLang.g4
```

Интерпретатор (EvalVisitor).

Основная структура

Интерпретатор реализован на Python и использует механизм обхода дерева разбора, генерируемого ANTLR (Visitor).

Код организован в классе EvalVisitor, который обрабатывает узлы грамматики.

Содержимое eval_visitor.py:

```
from RusLangVisitor import RusLangVisitor
from RusLangParser import RusLangParser

class EvalVisitor(RusLangVisitor):
    def __init__(self):
        super().__init__()
        # Здесь храним переменные и их значения
        self.memory = {}

    # -----
    # Визит "программы" в целом
    # -----
    def visitProgram(self, ctx: RusLangParser.ProgramContext):
        # Перебираем все глобальные операторы
        return self.visitChildren(ctx)

    # -----
    # Объявления переменных
    # -----
    def visitVarDeclaration(self, ctx: RusLangParser.VarDeclarationContext):
        vartype = ctx.getChild(0).getText() # "цел", "лог", "стр"
        # varList -> IDENT (COMMA IDENT)*
        for ident in ctx.varList().IDENT():
            name = ident.getText()
            # Инициализируем значением по умолчанию:
            if vartype == 'цел':
                self.memory[name] = 0
            elif vartype == 'стр':
                self.memory[name] = ""
            elif vartype == 'лог':
                self.memory[name] = False
            else:
                self.memory[name] = None
        return None

    # -----
    # Присваивание: a = expr;
    # -----
    def visitAssignmentStatement(self, ctx: RusLangParser.AssignmentStatementContext):
        var_name = ctx.IDENT().getText()
        val = self.visit(ctx.expr()) # вычислим expr

        # Преобразуем значение к нужному типу (целое, строка или логическое)
        if isinstance(val, int):
            self.memory[var_name] = int(val)
        elif isinstance(val, str):
            self.memory[var_name] = str(val)
        elif isinstance(val, bool):
            self.memory[var_name] = bool(val)
        else:
```

```

        self.memory[var_name] = val # Сохраняем, если это неизвестный тип
    return None

# -----
# Вывод: вывести(expr);
# -----
def visitPrintStatement(self, ctx: RusLangParser.PrintStatementContext):
    val = self.visit(ctx.expr())
    print(val)
    return None

# -----
# ifStatement:
#   если (expr) тогда blockStatement (иначеPart)?
# -----
def visitIfStatement(self, ctx: RusLangParser.IfStatementContext):
    # Проверяем основное условие (если)
    if self.visit(ctx.expr()): # Если основное условие истинно
        self.visit(ctx.blockStatement())
        return None # Завершаем обработку, если условие выполнено
    else:
        # Проверяем все блоки "иначеесли"
        for elifPart in ctx.иначеЕслиPart():
            if self.visit(elifPart.expr()): # Если условие elif истинно
                self.visit(elifPart.blockStatement())
                return None # Прекращаем выполнение, если найдено истинное условие

        # Проверяем блок "иначе", если он существует
        if ctx.иначеPart():
            self.visit(ctx.иначеPart().blockStatement())

    return None

# -----
# иначеPart: иначе blockStatement
# -----
def visitИначеPart(self, ctx: RusLangParser.ИначеPartContext):
    # просто заходим в блок
    self.visit(ctx.blockStatement())
    return None

# -----
# whileStatement:
#   пока (expr) blockStatement
# -----
def visitWhileStatement(self, ctx: RusLangParser.WhileStatementContext):
    # Выполняем цикл, пока условие истинно
    while self.visit(ctx.expr()):
        self.visit(ctx.blockStatement())
    return None

```

```

# -----
# blockStatement: { statement* }
# -----

def visitBlockStatement(self, ctx: RusLangParser.BlockStatementContext):
    # Просто обходим всех детей (statement)
    return self.visitChildren(ctx)

# -----
# Выражения (expr)
# -----

def visitAtom(self, ctx: RusLangParser.AtomContext):
    # atom : INT_NUMBER | ИСТИНА | ЛОЖЬ | IDENT | LPAREN expr RPAREN
    if ctx.INT_NUMBER():
        return int(ctx.INT_NUMBER().getText())
    elif ctx.STRING():
        return ctx.STRING().getText()[1:-1]
    elif ctx.ИСТИНА():
        return True
    elif ctx.ЛОЖЬ():
        return False
    elif ctx.IDENT():
        var_name = ctx.IDENT().getText()
        return self.memory.get(var_name, 0) # если нет в памяти, возвращаем 0
    elif ctx.expr():
        return self.visit(ctx.expr())
    return None

def visitRelationalExpr(self, ctx: RusLangParser.RelationalExprContext):
    # Вычисляем левый операнд
    left = self.visit(ctx.additiveExpr(0))

    # Проверяем наличие второго операнда
    if ctx.additiveExpr(1):
        right = self.visit(ctx.additiveExpr(1)) # Вычисляем правый операнд
        operator = ctx.getChild(1).getText() # Оператор сравнения

        if operator == '<':
            return left < right
        elif operator == '>':
            return left > right
        elif operator == '<=':
            return left <= right
        elif operator == '>=':
            return left >= right
        elif operator == '==':
            return left == right
        elif operator == '!=':
            return left != right

    # Если нет второго операнда, возвращаем левый
    return left

```

```
def visitAdditiveExpr(self, ctx: RusLangParser.AdditiveExprContext):
    # Вычисляем первый операнд (всегда существует)
    left = self.visit(ctx.multiplicativeExpr(0))

    # Если есть второй операнд (например, в случае "a + b"), обработаем его
    if ctx.multiplicativeExpr(1):
        right = self.visit(ctx.multiplicativeExpr(1))
        operator = ctx.getChild(1).getText() # Оператор "+" или "-"
        if operator == '+':
            # Если оба операнда строки или один из них – строка
            if isinstance(left, str) or isinstance(right, str):
                return str(left) + str(right) # Конкатенация строк
            return left + right
        elif operator == '-':
            return left - right

    # Если второго операнда нет, возвращаем только первый операнд
    return left

def visitMultiplicativeExpr(self, ctx: RusLangParser.MultiplicativeExprContext):
    # Вычисляем первый операнд
    left = self.visit(ctx.atom(0))

    # Проверяем наличие второго операнда
    if ctx.atom(1):
        right = self.visit(ctx.atom(1))
        operator = ctx.getChild(1).getText() # Оператор: "*", "/", "%"

        if operator == '*':
            return left * right
        elif operator == '/':
            if right == 0:
                raise ZeroDivisionError("Деление на ноль")
            return left / right
        elif operator == '%':
            return left % right

    # Если второго операнда нет, возвращаем только первый операнд
    return left
```

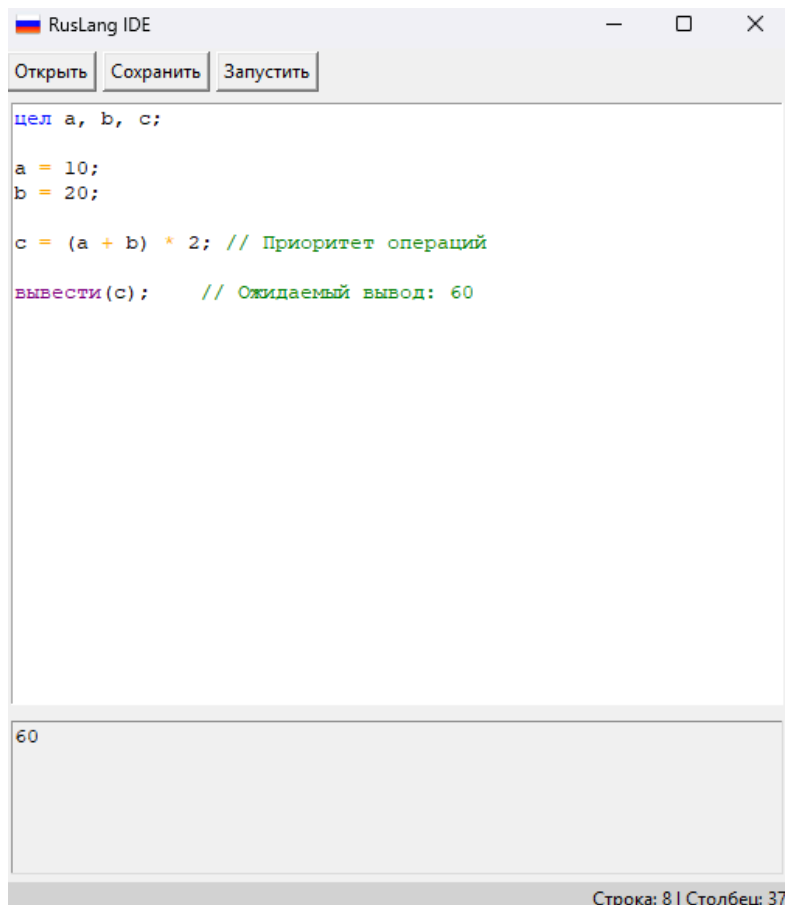

Среда разработки (IDE).

Для работы с кодом RusLang была разработана простая среда разработки на Tkinter. Она включает:

1. **Редактор текста** с подсветкой синтаксиса (ключевые слова, операторы, комментарии).
2. **Поле вывода**, где отображаются результаты выполнения.
3. **Статусная строка**, показывающая текущую строку и столбец.
4. **Кнопки управления**: открытие, сохранение, запуск программы.
5. **Контекстное меню**: копирование, вставка, выделение.

Примеры выполнения кода:

1. Приоритет операций.



```
цел a, b, c;

a = 10;
b = 20;

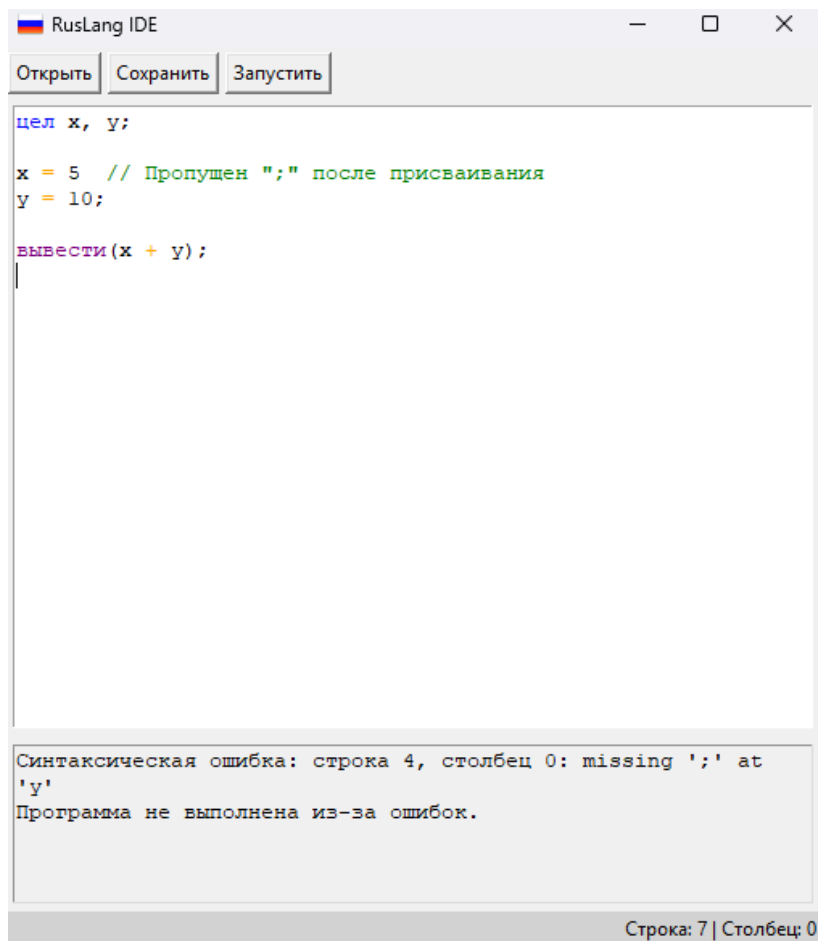
c = (a + b) * 2; // Приоритет операций

вывести(c);     // Ожидаемый вывод: 60
```

60

Строка: 8 | Столбец: 37

2. Пропуск “;” после операции присваивания.

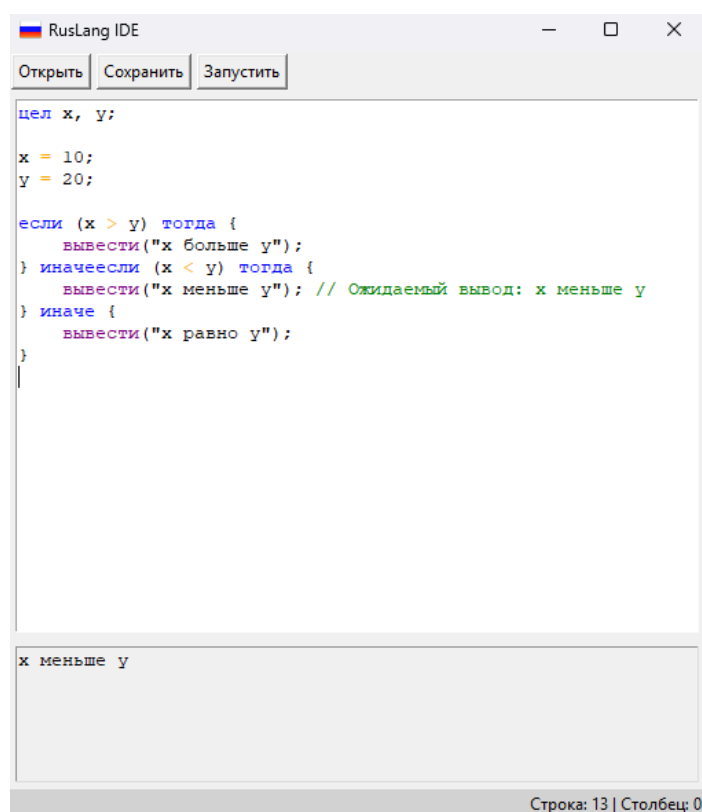


The screenshot shows the RusLang IDE interface. The editor contains the following code:

```
цел x, y;  
  
x = 5 // Пропущен ";" после присваивания  
y = 10;  
  
вывести(x + y);  
|
```

The error message at the bottom reads: "Синтаксическая ошибка: строка 4, столбец 0: missing ';' at 'y'". Below the error message, it says "Программа не выполнена из-за ошибок." (The program was not executed due to errors.) The status bar at the bottom right indicates "Строка: 7 | Столбец: 0".

3. Использование оператора if-elif-else.

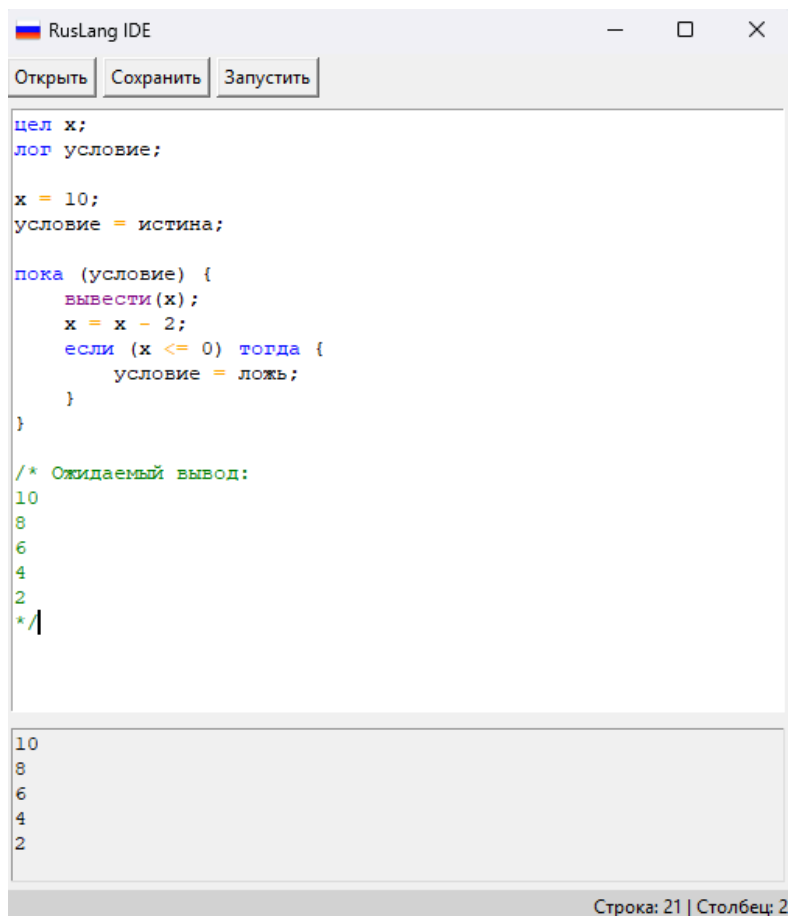


The screenshot shows the RusLang IDE interface. The editor contains the following code:

```
цел x, y;  
  
x = 10;  
y = 20;  
  
если (x > y) тогда {  
    вывести("x больше y");  
} иначеесли (x < y) тогда {  
    вывести("x меньше y"); // Ожидаемый вывод: x меньше y  
} иначе {  
    вывести("x равно y");  
}  
|
```

The output window at the bottom shows the result: "x меньше y". The status bar at the bottom right indicates "Строка: 13 | Столбец: 0".

4. Использование цикла while + многострочный комментарий.



```
цел x;  
лог условие;  
  
x = 10;  
условие = истина;  
  
пока (условие) {  
    вывести(x);  
    x = x - 2;  
    если (x <= 0) тогда {  
        условие = ложь;  
    }  
}  
  
/* Ожидаемый вывод:  
10  
8  
6  
4  
2  
*/
```

10
8
6
4
2

Строка: 21 | Столбец: 2

Заключение

В ходе работы был реализован компилятор для языка RusLang, включающий:

1. **Грамматика на ANTLR4:** описаны лексические и синтаксические правила.
2. **Интерпретатор:** обработка программ через обход дерева разбора.
3. **IDE:** редактор с подсветкой синтаксиса и функцией запуска программ.
4. **Демонстрация:** успешное выполнение корректных программ и обработка ошибок.

Язык RusLang и его интерпретатор могут быть расширены за счёт добавления новых возможностей (например, работы с массивами или функциями).