НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра прикладной математики

Курсовая работа

“Компилятор модельного языка”

Руководитель:

Санников Николай Александрович

Студент:

Корноухов Илья Владимирович

Группа 23-ПМ-2

Нижний Новгород 2025

**Содержание**

[Содержание 1](#_Toc1533514480)

[Введение 1](#_Toc1533514480)

1 [Постановка задачи 1](#_Toc1533514480)

2 [Формальная модель задачи 1](#_Toc1533514480)

3 [Спецификация основных процедур и функций 1](#_Toc1533514480)

[3.1 Лексический анализатор 1](#_Toc1533514480)

3.2 [Синтаксический анализатор 1](#_Toc1533514480)

3.3 [Семантический анализатор 1](#_Toc1533514480)

3.4 [Генерация внутреннего представления программы 1](#_Toc1533514480)

3.5 [Интерпретатор программы 1](#_Toc1533514480)

4 [Структурная организация данных 1](#_Toc1533514480)

4.1 [Спецификация входных данных 1](#_Toc1533514480)

1.2 [Спецификация выходных данных 1](#_Toc1533514480)

[5 Разработка алгоритма решения задачи 1](#_Toc1533514480)

[5.1 Укрупненная схема алгоритма программного средства 1](#_Toc1533514480)

[5.2 Детальная разработка алгоритмов отдельных подзадач 1](#_Toc1533514480)

[6 Установка и эксплуатация программного средства 1](#_Toc1533514480)

[7 Работа с программным средством 1](#_Toc1533514480)

[Заключение 1](#_Toc1533514480)

[Список использованных источников 1](#_Toc1533514480)

[Приложение А – Текст программы 1](#_Toc1533514480)

[Приложение Б – Контрольный пример 1](#_Toc1533514480)

**Введение**

**Постановка задачи**

Разработать компилятор модельного языка, выполнив следующие действия.

В соответствии с номером варианта составить формальное описание модельного языка программирования с помощью РБНФ, диаграмм Вирта или формальных грамматик.

Написать пять содержательных примеров программ, раскрывающих особенности конструкций учебного языка программирования, отразив в этих примерах все его функциональные возможности.

Составить таблицы лексем и диаграмму состояний с действиями для распознавания и формирования лексем языка.

По диаграмме с действиями написать функцию сканирования текста входной программы на модельном языке.

Разработать программное средство, реализующее лексический анализ текста программы на входном языке.

Реализовать синтаксический анализатор текста программы на модельном языке методом рекурсивного спуска.

Построить цепочку вывода и дерево разбора простейшей программы на модельном языке из начального символа грамматики.

Дополнить синтаксический анализатор процедурами проверки семантической правильности программы на модельном языке в соответствии с контекстными условиями вашего варианта.

Распечатать пример таблиц идентификаторов и двуместных операций.

Показать динамику изменения содержимого стека при семантическом анализе программы на примере одного синтаксически правильного выражения.

Записать правила вывода грамматики с действиями по переводу в ПОЛИЗ программы на модельном языке.

Пополнить разработанное программное средство процедурами, реализующими генерацию внутреннего представления введенной программы в форме ПОЛИЗа.

Разработать интерпретатор ПОЛИЗа программы на модельном языке.

Составить набор контрольных примеров, демонстрирующих все возможные типы лексических, синтаксических и семантических ошибок в программах на модельном языке, перевод в ПОЛИЗ различных конструкций языка представить ход интерпретации синтаксически и семантически правильной программы с помощью таблицы.

**Формальная модель задачи**

Регулярной грамматикой G называется кортеж вида

<T, N, S, P>, где:

T - множество терминалов

N - множество нетерминалов

S - начальный символ грамматики из множества нетерминалов

P - правила грамматики, имеющие вид A -> b, A -> bB, A -> Bb, где A и B - нетерминал, b - терминал.

Конечным автоматом M называется кортеж вида

<Q, T, S, F, P>, где:

Q - множество состояний автомата

T - алфавит, состоящий из терминалов исходной грамматики

S - начальное состояние

F - множество конечных состояний

P - множество переходов вида <q1, a> -> q2, где q1 и q2 - состояния автомата, a - символ из алфавита

КС-грамматикой G1 называется кортеж вида <T, N, S, P>, где:

T - множество терминалов

N - множество нетерминалов

S - начальный символ грамматики из множества нетерминалов

P - правила грамматики, имеющие вид a -> b, где a - нетерминал, b - последовательность терминалов и нетерминалов

.

**Спецификация основных процедур и функций**

**Лексический анализатор**

Лексический анализатор (ЛА) – это первый этап процесса компиляции, на котором символы, составляющие исходную программу, группируются в лексемы.

На вход лексическому анализатору подаётся цепочка символов исходного кода компилируемой программы. ЛА проверяет цепочку на принадлежность к регулярной грамматике модельного языка и формирует таблицы лексем и идентификаторов.

Лексический анализатор содержит набор процедур и состояний, необходимых для составления таблиц лексем и идентификаторов.

При чтении символа из цепочки ЛА может переходить из одного состояния в другое и вызывать процедуры.

Лексический анализатор использует таблицы:

serviceTable - таблица служебных слов

delimiterTable - таблица ограничителей

identifierTable - таблица идентификаторов

numberTable - таблица числовых значений

lexemesSeqTable - таблица лексем исходной программы.

Лексический анализатор имеет следующие состояния:

READ - начальное состояние, чтение лексемы

IDENT - чтение идентификатора

NUM\_BIN, NUM\_BIN\_FIN - чтение двоичного числа

NUM\_OCT, NUM\_OCT\_FIN - чтение восьмеричного числа

NUM\_DEC, NUM\_DEC\_FIN - чтение десятичного числа

NUM\_HEX, NUM\_HEX\_FIN - чтение шестнадцатеричного числа

NUM\_REAL\_POINT\_1, NUM\_REAL\_POINT\_ORDER\_START\_1, NUM\_REAL\_ORDER, NUM\_REAL\_POINT\_ORDER\_START\_2, NUM\_REAL\_POINT\_2, NUM\_REAL\_ORDER\_OR\_HEX, NUM\_REAL\_POINT\_ORDER - чтение действительного числа

DELIMITER - чтение разделителя

COMMENT, COMMENT\_START, COMMENT\_END - чтение комметнария

MORE\_THEN\_EQUAL - чтение разделителя “>=”

LESS\_THEN\_EQUAL - чтение разделителя “<=”

END - конечное состояние

ERROR - состояние ошибки

Лексический анализатор содержит набор переменных:

lexBuffer - переменная для накопления символов текущей лексемы

currentState - текущее состояние лексического анализатора

currentChar - текущий прочитанный символ

curLexId - номер лексемы из таблицы

curTableId - номер таблицы лексемы

curRow - номер текущей строки программы

curCol - номер текущего символа в строке программы

Лексический анализатор может вызывать следующие процедуры:

isBinAllow - проверка текущего символа на принадлежность к символам двоичного числа

isOctAllow - проверка текущего символа на принадлежность к символам восьмеричного числа

isDecAllow - проверка текущего символа на принадлежность к символам десятичного числа

isHexAllow - проверка текущего символа на принадлежность к символам шестнадцатеричного числа

isNumber - проверка текущего символа на принадлежность к символам числовой строки

isLetter - проверка текущего символа на принадлежность к множеству букв

clean - очистка lexBuffer

add - добавление текущего символа к lexBuffer

read - чтение следующего символа в currentChar

check(t) - проверка содержимого lexBuffer на соответствие лексеме из таблицы t

put(t, nt, d) - добавление лексемы из lexBuffer в таблицу t с возможностью указания типа nt и системы счисления d при сохранении числа

write(t, l) - запись номера t таблицы и номера l лексемы в lexemesSeqTable.

Ниже представлен модельный язык в формате РБНФ:

<буква>::= (A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p q | r | s | t | u | v | w | x | y | z)

<цифра>::= (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9)

<операции\_группы\_отношения>:: = (< > | = | < | <= | > | >=)

<операции\_группы\_сложения>:: =(+ | - | or)

<операции\_группы\_умножения>::=(\* | / | and)

<логическая\_константа>::= (true | false)

<унарная\_операция>::= not

<тип>::= (int | float | bool)

<двоичное>::= {/ 0 | 1 /} (B | b)

<восьмеричное>::= {/ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 /} (O | o)

<идентификатор>::= <буква> {<буква> | <цифра>}

<шестнадцатеричное>::= <цифра> {<цифра> | A | B | C | D | E | F | a | b | c | d | e | f} (H | h)

<десятичное>::= {/ <цифра> /} [D | d]

<числовая\_строка>::= {/ <цифра> /}

<целое>::= (<двоичное> | <восьмеричное> | <десятичное> | <шестнадцатеричное>)

<порядок>::= ( E | e )[+ | -] <числовая\_строка>

<ввода>::= read (<идентификатор> {, <идентификатор> })

<описание>::= <тип> <идентификатор> { , <идентификатор> }

<действительное>::= (<числовая\_строка> <порядок> | [<числовая\_строка>] . <числовая\_строка> [порядок])

<число>::= (<целое> | <действительное>)

<множитель>::= (<идентификатор> | <число> | <логическая\_константа> | <унарная\_операция> <множитель> | <выражение>)

<слагаемое>::= <множитель> {<операции\_группы\_умножения> <множитель>}

<операнд>::= <слагаемое> {<операции\_группы\_сложения> <слагаемое>}

<выражение>::= <операнд>{<операции\_группы\_отношения> <операнд>}

<составной>::= <оператор> { ( : | перевод строки) <оператор> }

<присваивания>::= <идентификатор> ass <выражение>

<условный>::= if <выражение> then <оператор> [ else <оператор>]

<фиксированного\_цикла>::= for <присваивания> to <выражение> do <оператор>

<условного\_цикла>::= while <выражение> do <оператор>

<вывода>::= write (<выражение> {, <выражение> })

<оператор>::= (<составной> | <присваивания> | <условный> | <фиксированного\_цикла> | <условного\_цикла> | <ввода> | <вывода>)

<программа>::= “{“ {/ (<описание> | <оператор>) ; /} “}”

**Синтаксический анализатор**

Синтаксический анализ - этап процесса компиляции, в ходе которого последовательность лексем исходной программы проверяется на принадлежность к КС-грамматике языка.

Алгоритм синтаксического анализатора основан на методе рекурсивного спуска.

Синтаксический анализатор содержит функции для проверки начала оставшейся цепочки лексем на принадлежность к соответствующему нетерминалу:

OGO - операции группы отношения

OGS - операции группы сложения

OGU - операции группы умножения

LC - логическая константа

UO - унарная операция

T - тип

MNOZH - множитель

EXPR - выражение

SLAG - слагаемое

OPRND - операнд

ENTER - оператор ввода

OUT - оператор вывода

PRISV - оператор присваивания

USLOV - оператор условный

FIXLOOP - оператор фиксированного цикла

USLLOOP - оператор условного цикла

OPERATOR - оператор

DESC - описание

PROG - программа

В качестве входных данных СиА принимает таблицы лексем и идентификаторов, сформированные на этапе лексического анализа.

Результатом работы СиА является заключение о принадлежности последовательности лексем программы к КС-грамматике языка.

**Семантический анализатор**

Семантический анализ - этап компиляции, необходимый для проверки программы на соответствие семантическим соглашениям, таким как правильность типов данных в выражениях и операторах, а так же проверка описаний переменных.

Семантический анализ выполняется параллельно с синтаксическим.

Для проверки выражений в СиА:

1. Вводится таблица binOperationTable
2. Добавляется exprStack - стек обработки операндов в выражениях
3. Функции EXPR, MNOZH, EXPR, SLAG и OPRND заполняют стек операндами и операторами.
4. В функции EXPR выполняется проверка соответствия оператора операндам из стека согласно таблице бинарных операций и запись в стек результата операции.

Для проверки описаний в СиА:

1. Таблица идентификаторов дополняется столбцами isInit и type.
2. В функции DESC выполняется проверка отсутствия описания переменной, отметка переменной как описанной и присвоение ей типа.

Для проверки операторов в функции PRISV, USLOV, USLLOOP добавляется проверка типа в exprStack после обработки выражения.

Для выполнения семантического анализа используются функции:

getType(l) - функция для определения типа лексемы l из очереди лексем

getExprType(s) - функция обработки стека выражения s для определения типа выражения.

Результатом работы СиА является заключение о соблюдении семантических соглашений.

**Генерация внутреннего представления программы**

В качестве внутреннего представления программы используется польская инверсная запись ПОЛИЗ. Генерация ПОЛИЗ выполняется параллельно с синтаксическим анализом с помощью синтаксически управляемого перевода.

Для этого в СиА:

1. Вводятся таблицы: polizTable - таблица для записи ПОЛИЗ программы, polizPointerItems - таблица меток для записи меток перехода
2. В таблицу разделителей добавляются операции: ‘!’ - безусловный переход к метке, ‘!F’ - условный переход по лжи к метке, ‘R’ - чтение в переменную, ‘W’ - вывод значения в консоль
3. В функциях EXPR, SLAG, OPRND, MNOZH, ENTER, OUT, PRISV, USLOV, FIXLOOP и USLLOOP СиА выполняется формирование ПОЛИЗ программы и запись в polizTable.

**Интерпретатор программы**

Для выполнения ПОЛИЗ программы используется интерпретатор. Интерпретатор читает последовательность ПОЛИЗ по порядку один раз. При чтении операнда он заносится в стек, при чтении оператора его операнды выводятся из стека, результат операции записывается в стек.

Для хранения значений логических констант, переменных и чисел используется хэш таблица memory.

Для интерпретации программы используются функции:

getTypeById(t, l) - функция определения типа лексемы l из таблицы t

getValueById(t, l) - функция определения значения для лексемы l из таблицы t

getMemoryAddress(t, l) - функция получения ключа memory для лексемы l из таблицы t

getOperandsCount(s) - функция для получения колличества операндов для оператора s

getBinOpResultType(optr, opnd1, opnd2) - функция для получения типа результата бинарной операции для оператора optr и операндов opnd1, opnd2.

Результатом работы интерпретатора является выполнение операций из ПОЛИЗ.

**Структурная организация данных**

**Спецификация входных данных**

**Спецификация выходных данных**

**Разработка алгоритма решения задачи**

**Укрупненная схема алгоритма программного средства**

**Детальная разработка алгоритмов отдельных подзадач**

**Установка и эксплуатация программного средства**

Для эксплуатации программного средства требуется наличие установленной JVM версии 8 или выше. Программное средство представляет собой .jar файл, исполняемый виртуальной машиной Java и не требует предварительной установки.

Для корректной работы компилятора требуется от 200Мб оперативной памяти, в зависимости от размера компилируемой программы.

**Работа с программным средством**

Для запуска программы необходимо выполнить команду: java -jar <Название исполняемого файла >.

По умолчанию программа ищет файл main.prog в текущем каталоге, переводит в ПОЛИЗ и интерпретирует.

Программа может принимать в качестве первого аргумента:

‘-i’ - перевод в ПОЛИЗ и интерпретация

‘-l’ - вывод в консоль заполненных таблиц, используемых при компиляции без интерпретации

‘-p’ - вывод ПОЛИЗ программы без интерпретации.

В качестве второго аргумента программа может принимать путь к файлу с исходным кодом компилируемой программы.

**Заключение**

**Список использованных источников**

**Приложение А – Текст программы**

**Приложение Б – Контрольный пример**

{

/\* чтение чисел пока не прочитается отрицательное \*/

float num;

read(num);

while num >= 0 do

read(num);

}

{

/\* факториал числа \*/

int n, i, fact;

read(n);

fact ass 1;

for i ass 2 to n + 1 do

fact ass fact \* i;

write(fact);

}

{

/\* сортировка трех чисел по возрастанию \*/

int x, y, z, t;

read(x, y, z);

if x > y then

t ass x

x ass y

y ass t;

if x > z then

t ass x

x ass z

z ass t;

if y > z then

t ass y

y ass z

z ass t;

write(x, y, z);

}