Министерство НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Курский государственный университет»

Факультет физики, математики, информатики

Кафедра алгебры, геометрии и теории обучения математике

Курсовая работа

по дисциплине

Теория формальных языков и трансляций

на тему: РАЗРАБОТКА КОМПИЛЯТОРА МОДЕЛЬНОГО ЯЗЫКА ПРОГРАММИРОВАНИЯ

Вариант 29

Обучающегося 3 курса

очной формы обучения

направления подготовки

02.03.03 Математическое обеспечение и администрирование информационных систем

Направленность (профиль) Проектирование информационных систем и баз данных

Кузьмин Дмитрий Максимович

(фамилия, имя, отчество)

Руководитель: к.п.н., доцент Селиванова Ирина Васильевна

(ученая степень, должность, фамилия, имя, отчество)

Допустить к защите:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_\_г.

Курск, 2023

**Содержание**

Введение 3

1 Постановка задачи 4

2 Основы теории разработки компиляторов …………………………5

**Введение**

**Компилятор** — транслятор, который осуществляет перевод всей исходной программы в эквивалентную ей результирующую программу на языке машинных команд или на языке ассемблера. Большинство компиляторов переводят программу с некоторого высокоуровневого языка программирования в машинный код, который может быть непосредственно выполнен компьютером, то есть в набор инструкций для центрального процессора. Компьютер, для которого производится компиляция, называется целевой машиной.

Без компиляторов процесс разработки был бы более сложным и менее эффективным. Сложно представить процесс решения большинства современных задач без высокоуровневых языков программирования. С разработкой новых языков программирования тесно связано понятие компиляции. Поэтому, получение навыка разработки компиляторов языков программирования является актуальным направлением.

Объектом исследования являются языки программирования.

Предметом исследования – алгоритмы и методы, лежащие в основе разработки компиляторов.

**Цель курсовой работы­ -** получение практических навыков разработки компиляторов языков программирования, основанных на теоретических знаниях, полученных в области теории формальных языков и трансляций.

Из цели вытекают **задачи** исследования:

- разработать грамматику модельного языка программирования на основе варианта задания №29;

- описать алгоритмы лексического, синтаксического и семантического анализаторов, выделив предварительно семантические правила;

- реализовать программно алгоритмы анализаторов и представить результат визуализации каждого этапа компиляции;

- разработать алгоритмы и методы перевода кода программы на язык ассемблера.

Курсовая работа выполнена на основе варианта модельного языка №29, структурные особенности синтаксиса которого следующие:

1. **Структура программы:** <программа>::= program var <описание> begin <оператор> {; <оператор>} end.
2. **Синтаксис команд описания данных:** <описание>::= {<идентификатор> {, <идентификатор> } : <тип> ;}
3. **Синтаксис идентификаторов:** <буква> <последовательность цифр или ничего> <буква>
4. **Описание типов (в порядке следования: целый, действительный, логический):** <тип>::= integer | real | boolean
5. **Синтаксис составного оператора:** <составной>::= «{» <оператор> { ; <оператор> } «}»
6. Оператор оператора присваивания: <присваивание> ::= [ let ] <идентификатор> = <выражение>
7. Оператор условного перехода: <условный>::=switch <выражение>{ case Const1: <оператор>case Const2: <оператор> ... case ConstN: <оператор>}
8. Синтаксис оператора:

<фиксированного\_цикла>::=for each<идентификатор> in ( Const1,Const2,...) <оператор>

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

<ввода>::= readln идентификатор {, <идентификатор> }

<вывода>::= writeln <выражение> {, <выражение> }

1. Признак начала и конца комментария: /\* \*/

**1 Разработка грамматики модельного языка программирования**

Существуют три основных метода описания синтаксиса языков программирования: формальные грамматики, формы Бэкуса-Наура и диаграммы Вирта.

В данной работе рассмотрим две формы описания синтаксиса языка программирования:

- форма Бэкуса-Наура;

- с помощью формальной грамматики.

**1.1 Форма Бэкуса-Наура**

Метаязык, предложенный Бэкусом и Науром, использует следующие обозначения:

- символ «::=» отделяет левую часть правила от правой (читается: «определяется как»);

- нетерминалы обозначаются произвольной символьной строкой, заключенной в угловые скобки «<» и «>»;

- терминалы - это символы, используемые в описываемом языке; правило может определять порождение нескольких альтернативных цепочек, отделяемых друг от друга символом вертикальной черты «|» (читается: «или»).

Форма Бэкуса-Наура для варианта №29 имеет следующий вид:

<идентификатор> ::= <буква> < последовательность цифр или ничего > <буква>

<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*

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

**1.2 Формальная грамматика**

**Определение.** Формальной грамматикой называется четверка вида

****, где

- *VN* - конечное множество нетерминальных символов грамматики (обычно прописные латинские буквы);

*- VT* - множество терминальных символов грамматики (обычно строчные латинские буквы, цифры, и т.п.), *VT* ∩*VN =∅*;

*- Р* - множество правил вывода грамматики, являющееся конечным подмножеством множества (*VT∪ VN*)*+ ×* (*VT∪ VN*)*\**;

*- S -* начальный символ грамматики, *S* ∈*VN*.

Для варианта №29 множество терминальных символов имеет вид:

Формальная грамматика для варианта №29 имеет следующий вид:

Условныеобозначения:

P=<программа>, Info=<описание>, Ident=<идентификатор>, Idents=<идентификаторы>, Type=<тип>, FixCicle=<фиксированный цикл>, Expr=<выражение>, ConditCicle=<условный цикл>, K=<комментарий>, Oper=<оператор>, InputOper=<Ввод>, OutputOper=<Вывод>.

P→ program var info : begin oper end

info → type idents; | type idents; info

type → integer | real | boolean

idents → ident | ident, idents

oper → compositOper | initOper | conditOper | fixCicle | compositCicle | inputOper | outputOper

compositOper → begin opers end.

Opers → oper | oper opers

initOper → ident = expr

conditOper → switch expr {case c1:<oper> case c2:<oper> … case cN:<oper>}

fixCicle → for each ident in range {oper}

range → num to num

conditCicle → while expr do {oper}

inputOper → read(idents)

outputOper → write(expr) Символ U

**Расширенные формы Бэкуса-Наура (РБНФ)**

Для повышения удобства и компактности описаний, в РБНФ вводятся следующие дополнительные конструкции (метасимволы):

* квадратные скобки «[» и «]» означают, что заключенная в них синтаксическая конструкция может отсутствовать;
* фигурные скобки «{» и «}» означают повторение заключенной в них синтаксической конструкции ноль или более раз;
* сочетание фигурных скобок и косой черты «{/» и «/}» используется для обозначения повторения один и более раз;
* круглые скобки «(» и «)» используются для ограничения альтернативных конструкций.

**- Пример 2.3.** В соответствии с данными правилами синтаксис модельного языка для варианта №29 будет выглядеть следующим образом:

<prog>::= program var <info> begin <oper> {; <oper>} end.

<info>::= {<ident> {, <ident> } : <type> ;}

<ident> ::= <char> <noEmptyListNumbers > <char>

<type>::= integer | real | boolean

<compositOper>::= «{» <oper> { ; <oper> } «}»

<initOper> ::= [ let ] <ident> = <expr>

<conditOper>::=switch <expr>{ case c1: <oper>case c2: <oper> ... case cN: <oper>}

<fixCicle>::=for each<ident> in ( c1,c2,...) <oper>

<conditCicle>::= while <expr> do <oper>

<inputOper>::= readln <ident> {, <ident> }

<outputOper>::= writeln <expr> {, <expr> }

<K>::= /\* \*/

<numChar>:: = 1|2|3|4|5|6|7|8|9|0

<sing>:: = -|+

<num>:: = <numChar>|<numChar><num>

<intNum>:: = [<sing>] <num>

<char>:: = a|b|c|d|f|g|h|j…..

<SimpleStr>::= <char>|<char><simpleStr>

<str>:: = “<simpleStr>”

<compSing> :: = <|<=|>|>=|==

**2 Лексический, синтаксический и семантический анализаторы**

**2.1 Лексический анализатор**

Первым шагом в разработке компилятора является выполнение лексического анализа. Этот этап компиляции носит рекомендательный характер. Целью лексического анализа является выделение лексем и их приведение к форме, удобной для дальнейшей обработки.

Основным алгоритмом лексического анализатора является разбиение потока данных на отдельные объекты, называемые токенами, и сравнение этих объектов с разделителями, ключевыми словами и требуемым форматом констант и идентификаторов. В компиляторе токены представляются строками, а константы и переменные добавляются в один список.

В программе есть три вида токенов: ключевые слова (Key Words), разделители (Separators) и переменные (Variable).

В ключевые слова входят все слова используемые для функционала программы, например: program, var, for each, switch и т.д.

В разделители входят все символы-разделители, такие как: -, +, /, = и т.д.

В переменные входят все имена переменных, их значения, а также константы, например: a1, 123, PI, 3.14, example2 и т.д.

Сначала алгоритм удаляет комментарии, а затем передает весь текст, представленный в виде строки, в функцию лексического анализа. Внутри функции лексического анализа создается временная копия этой строки и, затем, уже сама временная копия проходит три проверки с помощью регулярных выражений, после которых образуются три группы токенов. Когда алгоритм находит совпадение, он удаляет его из временной копии строки и исходя из этого, невозможно допустить ошибки в правильности нахождения идентификатора.

Как было сказано выше, алгоритм лексического анализа разбивает текст на токены, структура которых следующая:

[Token] → {Value, TokenType}

Так же, алгоритм лексического анализа использует следующие функции:

1. Функция удаления комментариев.
2. Функция лексического анализа.
3. Функция токенизации.
4. Функция удаления дубликатов.

Вспомогательные функции:

1. Функция определения идентификатора.
2. Функция определения ключевого слова.
3. Функция определения разделителя.
4. Функция определения типа целых чисел.
5. Функция определения типа чисел двойной точности.
6. Функция определения логического типа.
7. Функция определения символьного типа.
8. Функция определения строчного типа.

Тест лексического анализа:

Входной текст:

program var

id1, id2, id3 : integer;

id4, id5 : real;

id6, id7, id8 : boolean;

begin

let id1 = 10;

let id2 = id1 + 5;

let id3 = id2 \* 2;

for each id in (1, 2, 3)

begin

writeln id;

end

/\* abcabfa = 228 \*/

switch id3

case 1: writeln "One";

case 2: writeln "Two";

case 3: writeln "Three";

end

Данные, полученные на выходе:

KEY WORD: program

KEY WORD: var

IDENTIFIER: id1

IDENTIFIER: id2

IDENTIFIER: id3

KEY WORD: integer

IDENTIFIER: id4

IDENTIFIER: id5

KEY WORD: real

IDENTIFIER: id6

IDENTIFIER: id7

IDENTIFIER: id8

KEY WORD: boolean

KEY WORD: begin

IDENTIFIER: let

IDENTIFIER: id1

INTEGER: 10

IDENTIFIER: let

IDENTIFIER: id2

IDENTIFIER: id1

INTEGER: 5

IDENTIFIER: let

IDENTIFIER: id3

IDENTIFIER: id2

INTEGER: 2

KEY WORD: for

KEY WORD: each

IDENTIFIER: id

KEY WORD: in

INTEGER: 1

INTEGER: 2

INTEGER: 3

KEY WORD: begin

KEY WORD: writeln

IDENTIFIER: id

KEY WORD: end

KEY WORD: switch

IDENTIFIER: id3

KEY WORD: case

INTEGER: 1

KEY WORD: writeln

STRING: "One"

KEY WORD: case

INTEGER: 2

KEY WORD: writeln

STRING: "Two"

KEY WORD: case

INTEGER: 3

KEY WORD: writeln

STRING: "Three"

KEY WORD: end