МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Пермский государственный

национальный исследовательский университет

ОТЧЕТ

по дисциплине «Формальные грамматики и методы трансляции»

Работу выполнила

студент группы ПМИ - 2

Густокашина Виктория

Проверил

\_\_\_\_\_\_\_\_\_\_ кафедры МОВС

(доц., ст. преп., асс.)

Пономарёв Ф.А.\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_» \_\_\_\_\_\_\_\_ 2021

Пермь 2021

**Компилятор и его структура**

Компилятор – это компьютерная программа, которая переводит компьютерный код, написанный на одном языке программирования (исходный язык), на другой язык (целевой язык).

Структура компилятора:

* модуль ввода – вывода
* лексический анализатор
* синтаксический анализатор
* семантический анализатор

Анализатор – определение правильности исходной программы и формирование сообщений об ошибках, если они были найдены.

Также в компилятор входит генератор кода. Генератор кода – отображение машинно-независимого промежуточного представления исходной программы на реальную ЭВМ. Генератор является машинно-зависимой частью компилятора, так как преобразует исходную программу в объектный код для конкретной машины с её архитектурой и операционной системой.

**Модуль ввода – вывода**

Модуль ввода-вывода считывает последовательность литер исходной программы, собирает их в лексему и передает ее лексическому анализатору. Также он передает позицию (номер строки и символа) литеры в коде.

Если какой – либо анализатор нашел ошибку, он передает информацию о ней модулю ввода – вывода, а именно сообщение о ошибке и позицию в коде. После завершения работы анализаторов этот модуль выдает список всех найденных ошибок.

Лексемы передаются в лексический анализатор последовательно – с помощью функции GetLexeme(), которая использует функцию GetNextChar() для получения очередного символа. Если в программе были на комментарии, на этом этапе они будут пропущены.

**Реализация**

Модуль ввода – вывода помещен в отдельный класс IOModule.

class IOModule {

public List<Error> errors = new List<Error>(); // список ошибок

public Position currentTextPos = new Position(); // позиция

private char currentChar; // текущий символ

public bool endOfFile = true;

//переменная для проверки конца файла

public IOModule(StreamReader reader) { … } // конструктор

public void printError() { … } //вывод ошибок

private bool CheckDoubleSymbol() { … }

// проверка на «склеенные» символы

private bool isDoubleSymbol() { … }

// разделение «склеенных» символов

private char GetNextChar { … } // получение следующего символа

public string GetLexeme() { … } //создание лексемы

}

**Лексический анализатор**

Функции:

* Полученные от модуля ввода-вывода лексемы переводит в токены. Токен — это логическая, неделимая единица синтаксиса языка.

Виды токенов:

- Ключевые слова ("+", "-", "\*", "/", ":", ";", "=", ":=", "==", "<>", ">", "<", ">=", "<=", ",", "(", ")", ".", "program", "var", "begin", "end", "integer", "real", "char", "boolean", "const")

- Константы (хранится значение константы и ее тип)

- Идентификаторы (имена программы, переменных, функций и т.д.)

* Если вид токена не определен, то в модуль ввода – вывода передается позиция этой лексемы и соответствующее сообщение об ошибке.

Полученные токены передаются в синтаксический анализатор функцией NextToken();

**Реализация**

class Lexer

{

public IOModule ioMod;

private string lexem;

private Token token;

static public Position pos;

public Lexer(StreamReader reader) { … } // конструктор

private bool isKeyWord(string lexem) { … }

// функция проверяет, является ли токен ключевым словом

private bool isConst(string lexem, ref type\_const type) { … }

// функция проверяет, является ли токен константой

private bool isIdent(string lexem) { … }

// функция проверяет, является ли токен идентификатором

public Token GetToken() { // функция получения токена из лексемы

type\_const type = type\_const.real;

lexem = ioMod.GetLexeme();

token = null;

if (isKeyWord(lexem))

{

token = new KeyWordToken(KeyWords.FindKW(lexem), pos);

}

else if (isConst(lexem, ref type))

{

token = new ConstToken(type, pos, lexem);

}

else if (isIdent(lexem))

{

token = new IdentToken(lexem, pos);

}

else

{

token = new UndefinedToken(lexem, pos);

Error err = new Error(pos, "Лексическая ошибка", lexem);

ioMod.errors.Add(err);

}

return token;}

**Тестирование**

Проверим работу программы на корректных входных данных.

Входные данные:

program Name1;

const pi = 3.14;

otr = -1;

var x,y,d,v: real;

s: integer;

ch: char;

b: boolean;

begin

d := 12.23; // kakoy-to komment

x := -100;

v := x+d - 2;

x := x -d;

y := v / (d \* 3);

ch:='c';

b := True;

end.

Пример работы лексического анализатора:

lex= program -> KEYWORD

lex= Name1 -> IDENT

lex= ; -> KEYWORD

lex= const -> KEYWORD

lex= pi -> IDENT

lex= = -> KEYWORD

lex= 3.14 -> CONST-real

lex= ; -> KEYWORD

lex= otr -> IDENT

lex= = -> KEYWORD

lex= - -> KEYWORD

lex= 1 -> CONST-integer

lex= ; -> KEYWORD

lex= var -> KEYWORD

lex= x -> IDENT

lex= , -> KEYWORD

lex= y -> IDENT

…

lex= ch -> IDENT

lex= := -> KEYWORD

lex= 'c' -> CONST-char

lex= ; -> KEYWORD

lex= b -> IDENT

lex= := -> KEYWORD

lex= True -> CONST-boolean

lex= ; -> KEYWORD

lex= end -> KEYWORD

lex= . -> KEYWORD

Добавим ошибки в исходную программу:

program Name1;

const 1pi = 3.14; {неправильное имя}

otr = -1;

var x,y,d,v: real;

s: integer;

ch: char;

b: boolean;

begin

d := 12.23; // kakoy-to komment

x := -100;

s := 123456789; {большое число для integer}

v := x+d - 2;

x := x -d;

y := v / (d \* 3);

ch:=''; {нет значения}

b := True?; {другой символ}

end.

Выведенные ошибки:

Неизвестные символы (2, 6) 1pi

Неизвестные символы (11, 7) 123456789

Неизвестные символы (15, 6) ''

Неизвестные символы (16, 7) True?

**Синтаксический анализатор**

Синтаксический анализатор проверяет программу на соответствие формальным правилам языка.

БНФ (Форма Бэкуса-Наура) - формальная система описания синтаксиса, в которой одни синтаксические категории последовательно определяются через другие категории БНФ.

Вид БНФ: <S> ::= L, где S – обозначение описываемой синтаксической конструкции, знак ‘::=’ читается как «это есть», L – описание конструкции S (состоит из других синтаксических конструкций, символов языка и метасимволов ‘|’ (‘или’) и ‘{ }’ (повторение внутренней конструкции 0 и более раз). В коде БНФ описаны в виде функций, то есть S{L}.

Используемые БНФ:

<программа>::= program <имя> <блок>.

<имя>::=<буква>{<буква>|<цифра>}

<блок>::=<раздел констант><раздел переменных><раздел операторов>

<раздел констант>::=<пусто>|const <определение константы>;

{<определение константы>;}

<определение константы>::=<имя>=<константа>

<раздел переменных>::= var <описание однотипных переменных>;

{<описание однотипных переменных>;}|<пусто>

<описание однотипных переменных>::=<имя>{,<имя>}:<тип>

<раздел операторов>::=<составной оператор>

<составной оператор>::= begin <оператор>{;<оператор>} end

<оператор>::=<простой оператор>| <сложный оператор>

<сложный оператор>::=<составной оператор>

<простой оператор>::=<оператор присваивания>|<пустой оператор>

<оператор присваивания>::=<переменная>:=<выражение>

<выражение>::=<простое выражение>

<простое выражение>::=<знак><слагаемое>

{<аддитивная операция><слагаемое>}

<аддитивная операция>::=+ | -

<слагаемое>::=<множитель>{<мультипликативная операция>

<множитель>}

<мультипликативная операция>::=\* | /

<множитель>::= <переменная>|<константа без знака>|(<выражение>)

Пример кода для БНФ <слагаемое>:

private void Term()

{

starters = new List<Keyword> { Keyword.Multiply, Keyword.Divide };

Multiplier();

{

while (Waiting(Keyword.Multiply) || Waiting(Keyword.Divide))

{

NextToken();

Multiplier();

}

}

}

**Реализация**

Функция Accept() проверяет совпадает ли ожидаемый токен с текущим. Если нет, то формируется сообщение об ошибке.

Функция Waiting() проверяет, является ли текущий символ одним из ожидаемых (которые не обязательно должны быть).

private bool Waiting(Token\_type type)

{

return (curToken.Type() == type && curToken != null);

}

Нейтрализация – действия, которые позволяют продолжить анализ после обнаружения ошибки, пропустив несколько символов. Используются списки starters (множество начальных символов) и followers(множество символов, которые могут встретиться после текущей инструкции), функции IsBelong(starters) - проверка, что токен соответствует началу конструкции и Skip(starters, followers) – если символ начала не найден пропускаются все токены, пока не будут найдены значения из starters или followers.

**Тестирование**

Добавим во входные данные синтаксические ошибки:

program Name1;

const pi = 3.14;

otr = -1;

var x,y,d,v: real;

s: integer;

ch: char;

b: boolean {пропуск точки с запятой}

begin

d := 12.23; // kakoy-to komment

x := -100;

v := x+d - 2;

x := x -d;

y := v / (d 3); { пропуск знака}

ch:='с';

b := ; {пропуск значения}

end.

Результат:

Отсутствует символ (8, 8) Semicolon

Синтаксическая ошибка (13, 13)

Синтаксическая ошибка (15, 15)

**Семантический анализатор**

Семантический анализатор проверяет программу на соответствие неформальным правилам языка. Неформальные правила задаются с помощью естественного языка и называются контекстными условиями. Многие контекстные условия связаны с идентификаторами и их использованием.

Этот анализатор проверяет уникальность идентификаторов и соответствие их типов, то есть нельзя записать в строковую переменную число и т.д.

Таблица идентификаторов – структура, в которой хранится имя идентификатора и его тип.

Когда синтаксический анализатор обрабатывает раздел переменных и констант семантический записывает имена этих констант и их тип в таблицу идентификаторов, если имя уже существует выдается сообщение об ошибке.

При анализе основной программы семантический анализатор проверяет, чтобы все переменные уже существовали и каждой переменной присваивалось значение ее типа (кроме real, так как можно присвоить real и integer).

**Тестирование**

program Name1;

const pi = 3.14;

otr = -1;

var x,y,d,v: real;

s: integer;

ch: char;

b: boolean;

begin

d := 12.23; // kakoy-to komment

x := -100;

v := x+d - 2;

t := x -d; {неизвестный символ}

y := v / (d + 3);

ch:= 12; {присваивание числового значения символьной переменной}

b := True;

end.

Результат:  
Несуществующий идентификатор (12, 12) t

Несоответствие типов (14, 14)

Проверим работу компилятора с разными типами ошибок.

program Name1;

const pi = 3.14;

otr = -1 {нет ; }

var x,y,d,v: real;

s: integer;

ch char; {нет : }

b: boolean;

begin

d := 12.23; // kakoy-to komment

x := -100 + 'x'; {символьная константа}

v := x+d - 2;

t := x -d; {несуществующий идентификатор}

y := v / (d 3); {нет знака}

ch:= 12; {числовое значение}

b := ; {нет значения}

end.

Результат:

Отсутствует символ (4, 4) Semicolon

Отсутствует символ (6, 6) Colon

Несоответствие типов (10, 10)

Несуществующий идентификатор (12, 12) t

Синтаксическая ошибка (13, 13)

Несоответствие типов (14, 14)

Синтаксическая ошибка (15, 15)