Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчёт**

**по лабораторной работе №4**

**Дисциплина: Методы разработки трансляторов**

**Тема: «Построение синтаксического анализатора»**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К.А. Корнилов

Направление подготовки 02.03.02 Фундаментальная информатика и

информационные технологии

Направленность (профиль) Математическое и программное обеспечение

компьютерных технологий

Преподаватель

д-р техн. наук, проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю. М. Вишняков

**СОДЕРЖАНИЕ**

[1 Вариант задания 3](#_Toc101878801)

[2 Построение синтаксического анализатора 3](#_Toc101878802)

[3 Результаты экспериментов 7](#_Toc101878803)

[Приложение А Грамматика языка программирования Java 12](#_Toc101878804)

[Приложение Б Листинг программы и комментарии к нему 14](#_Toc101878805)

**1 Вариант задания**

Вариант задания представляет собой пару: входной язык и выходной язык (таблица 1).

Таблица 1 – Вариант задания

|  |  |  |
| --- | --- | --- |
| № варианта | Входной язык | Выходной язык |
| 47 | PHP | R |

Разработать программу синтаксического анализатора исходного текста.

Программа получает на входе файл – результат лексического анализа и выполняет синтаксический анализ исходной программы. Результатом работы должно быть сообщение о корректности программы или сообщение об первой обнаруженной ошибке с указанием строки и конструкции языка, при разборе которых обнаружена ошибка.

Отчет по работе должен содержать полную грамматику заданного подмножества входного языка, описание алгоритма синтаксического разбора, листинг программы и комментарии к нему, пример.

**2 Построение синтаксического анализатора**

Основная задача синтаксического анализа – проверка исходной программы на соответствие грамматике языка программирования. Следует еще раз напомнить, что синтаксический анализ производится над кодом программы, который получен на выходе лексического анализа. Результат синтаксического анализа, который часто называется разбором, представляется в виде дерева разбора. Данное дерево должно демонстрировать вывод исходной программы как цепочки символов из начального символа грамматики.

Для построения дерева разбора используются различные методы синтаксического анализа, каждый из которых имеет свои особенности. В нашем случае будет рассмотрен пример разбора «сверху-вниз», который называется рекурсивным спуском. Основная особенность рекурсивного спуска заключается в том, что для анализа каждого нетерминала используется отдельная семантическая процедура, а многократное обращение к ней в процессе анализа и дало название методу – «рекурсивный спуск».

Рассмотрим суть синтаксического анализа методом рекурсивного спуска на примере.

Вначале построим грамматику языка программирования Java. Его грамматика приведена в приложении А.

При программной реализации данного алгоритма были определены следующие переменные и процедуры, за которыми закреплены определенные функциональные назначения.

1. Переменная NXTSYMB – глобальная переменная, содержащая символ, следующий за разбираемым символом. NXTSYMB содержит символ (лексему) исходной программы, который будет обрабатываться следующим, а при поиске новой цели в переменной NXTSYMB всегда находится первый символ, который должен быть исследован.

При выходе из процедуры с сообщением об успехе символ, следующий за закрытой подцепочкой, помещается в NXTSYMB.

1. Процедура SCAN готовит очередной символ исходной программы и помещает его в NXTSYMB.
2. Процедура ERROR предназначена для обработки ошибочных ситуаций.

Ниже в таблице 2 собраны нетерминалы грамматики и имена соответствующих им рекурсивных семантических процедур.

Таблица 2 – Нетерминалы грамматики и имена соответствующих им рекурсивных семантических процедур

|  |  |
| --- | --- |
| Нетерминальные символы грамматики | Имена рекурсивных процедур |
| <программа> | PROGRAM |
| <операторы> | OPERATORS |
| <оператор> | OPERATOR |
| <имя> | NAME |
| <описание> | DESCRIPTION |
| <описание переменных> | DESCRIPTION\_OF\_VARIABLES |
| <список имен> | LIST\_OF\_NAMES |
| <описание массивов> | DESCRIPTION\_OF\_ARRAYS |
| <описание массива> | ARRAY\_DESCRIPTION |
| <функция> | FUNCTION |
| <составной оператор> | COMPOUND\_OPERATOR |
| <выражение> | EXPRESSION |
| <число> | NUMBER |
| <целое число> | INTEGER |
| <вещественное число> | REAL\_NUMBER |
| <строка> | LINE |
| <переменная> | VARIABLE |
| <арифметическая операция> | ARITHMETIC\_OPERATION |
| <опер. присв.> | ASSIGNMENT\_OPERATOR |
| <условный оператор> | CONDITIONAL\_OPERATOR |
| <условие> | CONDITION |
| <унарная лог. операция> | UNARY\_LOG\_OPERATION |
| <лог. выражение> | LOG\_EXPRESSION |
| <операция сравнения> | COMPARISON\_OPERATION |

Продолжение таблицы 2

|  |  |
| --- | --- |
| Нетерминальные символы грамматики | Имена рекурсивных процедур |
| <бинарная лог. операция> | BINARY\_LOG\_OPERATION |
| <цикл for> | FOR\_LOOP |
| <цикл while> | WHILE\_LOOP |
| <оператор goto> | GOTO\_OPERATOR |
| <оператор break> | BREAK\_OPERATOR |
| <оператор continue> | CONTINUE\_OPERATOR |
| <оператор return> | RETURN\_OPERATOR |

Процедура NAME возвращает значение ИСТИНА или ЛОЖЬ в зависимости от того, является ли лексема, содержащаяся в переменной NXTSYMB, идентификатором.

Процедуры NUMBER, INTEGER и REAL\_NUMBER возвращают значение ИСТИНА или ЛОЖЬ в зависимости от того, является ли лексема, содержащаяся в переменной NXTSYMB, числовой константой.

Процедура LINE возвращает значение ИСТИНА или ЛОЖЬ в зависимости от того, является ли лексема, содержащаяся в переменной NXTSYMB, символьной константой.

Процедура ARITHMETIC\_OPERATION возвращает значение ИСТИНА или ЛОЖЬ в зависимости от того, является ли лексема, содержащаяся в переменной NXTSYMB, арифметической операцией.

Процедура UNARY\_LOG\_OPERATION возвращает значение ИСТИНА или ЛОЖЬ в зависимости от того, является ли лексема, содержащаяся в переменной NXTSYMB, унарной логической операцией.

Процедура COMPARISON\_OPERATION возвращает значение ИСТИНА или ЛОЖЬ в зависимости от того, является ли лексема, содержащаяся в переменной NXTSYMB, операцией сравнения.

Процедура BINARY\_LOG\_OPERATION возвращает значение ИСТИНА или ЛОЖЬ в зависимости от того, является ли лексема, содержащаяся в переменной NXTSYMB, бинарной логической операцией.

Процедура BREAK\_OPERATOR возвращает значение ИСТИНА или ЛОЖЬ в зависимости от того, является ли лексема, содержащаяся в переменной NXTSYMB, оператором break.

Процедура CONTINUE\_OPERATOR возвращает значение ИСТИНА или ЛОЖЬ в зависимости от того, является ли лексема, содержащаяся в переменной NXTSYMB, оператором continue.

При инициализации синтаксического анализатора идет обращение к процедуре SCAN, которая помещает первый символ исходной программы в глобальную переменную NXTSYMB и вызывает процедуру PROGRAM.

Для рассмотренной грамматики программа синтаксического анализатора по методу рекурсивного спуска приведена в приложении Б.

**3 Результаты экспериментов**

Пример работы программы для программы для тестирования: на рисунке 1, 3 и 5 представлены скриншоты файла, содержащего текст на входном языке программирования, на рисунках 2, 4 и 6 – скриншоты результата работы программы.

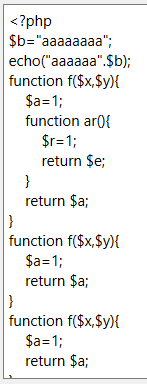


Рисунок 1 – Скриншот файла, содержащего текст на входном языке программирования



Рисунок 2 – Результат работы программы

Ничего не было выведено, значит, программа, записанная в файл, синтаксически верна.

Допустим ошибку нарочно: уберем символ начала программы (рисунок 3).

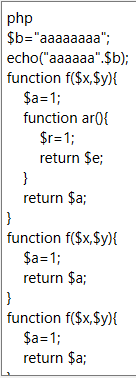


Рисунок 3 – Скриншот файла, содержащего текст на входном языке программирования

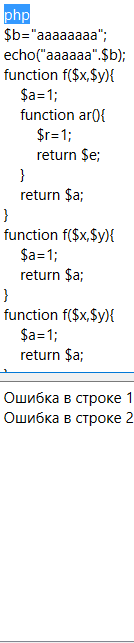


Рисунок 4 – Результат работы программы

Сделаем ошибку в строке 3, убрав точку с запятой в конце строки (рисунок 5).

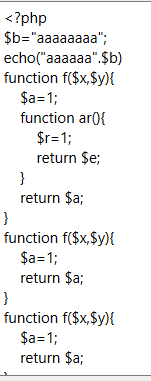


Рисунок 5 – Скриншот файла, содержащего текст на входном языке программирования

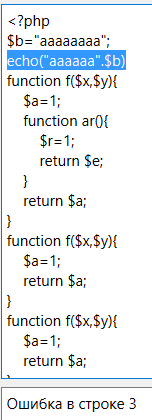


Рисунок 6 – Результат работы программы

**ПРИЛОЖЕНИЕ А   
Грамматика языка программирования PHP**

Начальным символом грамматики является нетерминал <программа>. Грамматика имеет следующий вид:

<программа> ::= <операторы>

<операторы> ::= <оператор>; {<оператор>;}

<оператор> ::= [<метка>] (<описание>

| <функция>

| <оператор вызова функции>

| <составной оператор>

| <опер. присв.>

| <условный оператор>

| <цикл while>

| <оператор break>

| <оператор continue>

| <оператор return>)

<метка> ::= <имя>:

<имя> ::= <буква> {<буква> | <цифра>}

<буква> ::= *A* | *B* | … | *Z*

<цифра> ::= 0 | 1 | … | 9

<описание> ::= <описание переменных> | <описание массива>

<описание переменных> ::= *<*тип*>* <описание переменной> {, <описание переменной>}

<описание переменной> ::= <имя> [= <выражение>]

<функция> ::= *FUNCTION*<имя>«(»[<список имён>]«)» <составной оператор>

<составной оператор> ::= «{»<операторы>«}»

<оператор вызов функции> ::= <имя>«(»<выражение> {, <выражение>}«)»

<выражение> ::= <оператор вызова функции>

| <число>

| <строка>

| <переменная>

| «(»<выражение>«)»

| <выражение> <арифметическая операция> <выражение>

<число> ::= <целое число> | <вещественное число>

<целое число> ::= <цифра> {, <цифра>}

<вещественное число> ::= <целое число>

| <целое число>.<целое число> [(e | E) [(- | +)] <целое число>]

<строка> ::= ''{символ}'' | '{символ}'

<переменная> ::= <имя>[«[»<выражение>«]»]

<арифметическая операция> ::= %| \* | \*\* | + | - | / | . | >> | >> | &| '|' | ^

<опер. присв.> ::= <переменная> = <выражение>

<условный оператор> ::= *IF* «(»<условие>«)» <оператор> [*ELSE* <оператор>]

<условие> ::= <унарная лог. операция> «(»<лог. выражение>«)»

| <лог. выражение> {<бинарная лог. операция> <лог. выражение>}

<унарная лог. операция> ::= !

<лог. выражение> ::= <выражение> <операция сравнения> <выражение>

<операция сравнения> ::= != | < | <= | == | > | >= | <>

<бинарная лог. операция> ::= and | or | xor

<цикл while> ::= *WHILE* «(»<условие>«)» <оператор>

<оператор break> ::= *BREAK*

<оператор continue> ::= *CONTINUE*

<оператор return> ::= *RETURN* <выражение>

**ПРИЛОЖЕНИЕ Б   
Листинг программы и комментарии к нему**

def error(self):

print(self.nxtsymb,self.row\_counter,22222)

out\_sq = 'Ошибка в строке '

f = open('./files/error.txt', 'a')

out\_sq += str(self.row\_counter)

f.write(out\_sq+'\n')

f.close()

return

def program(self):

self.scan()

if(self.nxtsymb!='<?'):

self.error()

self.scan()

if(self.nxtsymb!='php'):

self.error()

self.scan()

self.text()

if(self.nxtsymb!='?>'):

self.error()

def text(self):

self.no\_end\_op\_error = ['-','+','/','\*','\*\*',')',']','.','==','<=','>=','<','>'

',','<>','!=','%',';','^','&','|','<<','>>','and','or','xor']

while(self.nxtsymb!='?>'):

if(self.nxtsymb=='function'):

self.procedure()

elif self.identifier() or self.nxtsymb=='echo':

self.scan()

if self.nxtsymb == '(':

self.brackets.append('(')

self.scan()

if self.nxtsymb != ')':

self.expression()

while self.nxtsymb == ',':

self.scan()

self.expression()

if self.nxtsymb != ')':

self.error()

if len(self.brackets)==0 or (self.brackets and self.brackets.pop() != '('):

self.error()

self.scan()

elif self.nxtsymb == '=':

self.scan()

self.expression()

if self.nxtsymb != ';':

self.row\_counter-=1

self.error()

self.row\_counter+=1

elif self.nxtsymb=='$':

self.scan()

if not self.identifier():

self.error()

elif self.nxtsymb=='return':

self.scan()

self.expression()

if self.nxtsymb!=';':

self.row\_counter-=1

self.error()

self.row\_counter+=1

elif self.nxtsymb == 'break':

self.break\_operator()

self.scan()

if self.nxtsymb != ';': self.error()

elif self.nxtsymb == 'continue':

self.continue\_operator()

self.scan()

if self.nxtsymb != ';': self.error()

elif self.nxtsymb=='if':

self.conditional\_operator()

elif self.nxtsymb == 'while':

self.while\_loop()

elif self.nxtsymb =='do':

self.do\_loop()

elif self.nxtsymb=='{':

self.compound\_operator()

elif self.nxtsymb=='[':

self.scan()

self.brackets.append('[')

self.expression()

if self.nxtsymb!=']':

self.error()

if len(self.brackets)==0 or (self.brackets and self.brackets.pop()!='['):

self.error()

self.scan()

if self.nxtsymb == '=':

self.scan()

self.expression()

if self.nxtsymb != ';': self.error()

elif self.nxtsymb==';':

self.scan()

else:

break

# оператор break

def break\_operator(self):

return self.nxtsymb == 'break'

# оператор continue

def continue\_operator(self):

return self.nxtsymb == 'continue'

#Сканирование do-while

def do\_loop(self):

if self.nxtsymb !='do': self.error()

self.scan()

self.compound\_operator()

self.while\_loop()

#Сканирование while

def while\_loop(self):

if self.nxtsymb != 'while': self.error()

self.scan()

if self.nxtsymb != '(': self.error()

self.brackets.append('(')

self.condition()

if self.nxtsymb != ')': self.error()

if len(self.brackets)==0 or (self.brackets and self.brackets.pop()!='('):

self.error()

self.scan()

#Сканирование if

def conditional\_operator(self):

if self.nxtsymb != 'if': self.error()

self.scan()

if self.nxtsymb != '(': self.error()

self.condition()

self.brackets.append('(')

if self.nxtsymb != ')': self.error()

if len(self.brackets)==0 or (self.brackets and self.brackets.pop() != '('):

self.error()

self.scan()

self.text()

if self.nxtsymb == 'else':

self.scan()

self.text()

# условие

def condition(self):

if self.unary\_log\_operation():

self.scan()

if self.nxtsymb != '(': self.error()

self.brackets.append('(')

self.log\_expression()

if self.nxtsymb != ')': self.error()

if len(self.brackets)==0 or (self.brackets and self.brackets.pop() != '('):

self.error()

self.scan()

else:

self.log\_expression()

while self.binary\_log\_operation():

self.log\_expression()

# унарная логическая операция

def unary\_log\_operation(self):

return self.nxtsymb == '!'

# логическое выражение

def log\_expression(self):

self.scan()

self.expression()

self.comparison\_operation()

self.scan()

self.expression()

# операция сравнения

def comparison\_operation(self):

return self.nxtsymb in ['!=', '<', '<=', '==', '>', '>=','<>']

# бинарная логическая операция

def binary\_log\_operation(self):

return self.nxtsymb == 'and' or self.nxtsymb == 'or' or self.nxtsymb=='xor'

#Сканирование выражения

def expression(self):

if self.nxtsymb == '(':

self.brackets.append('(')

self.scan()

self.expression()

if self.nxtsymb != ')': self.error()

if len(self.brackets)==0 or (self.brackets and self.brackets.pop()!='('):

self.error()

self.scan()

elif self.nxtsymb =='$':

self.scan()

if not self.identifier():

self.error()

self.scan()

self.expression()

elif self.identifier():

self.scan()

if self.nxtsymb == '(':

self.brackets.append('(')

self.scan()

if self.nxtsymb != ')':

self.expression()

while self.nxtsymb == ',':

self.scan()

self.expression()

if self.nxtsymb != ')':

self.error()

if len(self.brackets) == 0 or (self.brackets and self.brackets.pop() != '('):

self.error()

self.scan()

elif self.nxtsymb == '[':

self.brackets.append('[')

self.scan()

self.expression()

if self.nxtsymb != ']': self.error()

if len(self.brackets) == 0 or (self.brackets and self.brackets.pop() != '['):

self.error()

self.scan()

elif self.number\_const() or self.constant():

self.scan()

else:

self.error()

if self.arithmetic\_operation():

self.scan()

self.expression()

#Проверяем арифметическая ли операция

def arithmetic\_operation(self):

return self.nxtsymb in ['%', '\*','\*\*', '+', '-', '/','.','>>','>>','&','|','^']

#Сканирование следующего символа

def scan(self):

self.i+=1

if(self.i>len(self.match)):

if self.nxtsymb not in ['\n',';','}']:

self.error()

else:

for token\_class in self.tokens.keys():

if self.match[self.i] in self.tokens[token\_class]:

self.nxtsymb = self.tokens[token\_class][self.match[self.i]]

self.printer()

if self.nxtsymb=='\n':

if self.match[self.i-1] in self.tokens['R'].keys() and self.tokens['R'][self.match[self.i-1]] not in ['{','}', ';','\n'] and self.i!=2:

self.error()

self.row\_counter += 1

self.scan()

#Сканирование функций-процедуры

def procedure(self):

#Сканируем название

self.scan()

if not self.identifier():

self.error()

self.scan()

#Сканируем аргументы

if self.nxtsymb!='(':

self.error()

self.brackets.append('(')

self.scan()

if self.nxtsymb!=')':

self.list\_of\_names(type\_read=1)

if self.nxtsymb!=')':

print(self.nxtsymb,1)

self.error()

if len(self.brackets) == 0 or (self.brackets and self.brackets.pop() != '('):

print(self.brackets)

self.error()

self.scan()

if self.nxtsymb!='{':

self.error()

else:

self.compound\_operator()

#Считываем список аргументов

def check\_is\_agrument(self,type\_read=0):

if(self.nxtsymb=='$'):

self.scan()

if not self.identifier():

self.error()

elif type\_read==1:

self.error()

else:

if not(self.constant() or self.number\_const()):

self.error()

def list\_of\_names(self,type\_read=0):

self.check\_is\_agrument(type\_read)

self.printer()

self.scan()

while(self.nxtsymb==','):

self.scan()

self.check\_is\_agrument(type\_read)

self.scan()

#Проверка идентификатора

def identifier(self):

return self.nxtsymb in self.tokens['I'].values()

#Проверка константы

def constant(self):

return self.nxtsymb in self.tokens['C'].values()

def number\_const(self):

return self.nxtsymb in self.tokens['N'].values()

#Сканирование блока {}

def compound\_operator(self):

if self.nxtsymb != '{':

self.error()

self.brackets.append('{')

self.scan()

self.text()

if self.nxtsymb != '}':

self.error()

print(self.brackets)

if len(self.brackets) == 0 or (self.brackets and self.brackets.pop() != '{'):

self.error()

self.scan()

def analyzer(self):

self.i = -1

self.nxtsymb = None # разбираемый символ

self.row\_counter = 1 # счётчик строк

# лексемы

self.tokens = {'W': {}, 'I': {}, 'O': {}, 'R': {}, 'N': {}, 'C': {}}

# файлы, содержащие все таблицы лексем

for token\_class in self.tokens.keys():

with open('./files/%s.json' % token\_class, 'r') as read\_file:

data = json.load(read\_file)

self.tokens[token\_class] = data

# файл, содержащий последовательность кодов лексем входной программы

f = open('./files/Output.txt', 'r')

input\_sequence = f.read()

f.close()

regexp = '[' + '|'.join(self.tokens.keys()) + ']' + '\d+'

self.match = re.findall(regexp, input\_sequence)

print(self.tokens)

print(self.match)

f=open('./files/error.txt','w')

f.close()

self.brackets = []

self.program()