Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Методы трансляции»

|  |  |
| --- | --- |
|  |  |

**ОТЧЕТ**

к лабораторной работе № 2

на тему «Лексический анализ»

Выполнил             К. А. Мамченко

Проверил                          Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc157960226)

[2 Теоретические сведения 4](#_Toc157960227)

[3 Результат выполнения 5](#_Toc157960235)

[Выводы 6](#_Toc157960236)

[Список использованных источников 7](#_Toc157960237)

[Приложение А (обязательное) Листинг кода 8](#_Toc157960238)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является освоение работы существующих лексических анализаторов. Разработка лексического анализатора подмножества языка программирования, определенного в лабораторной работе 1 и определение лексических правил.

## 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Лексический анализатор (также известный как сканер) – это компонент, используемый в процессе компиляции или интерпретации программного кода для разбора входного текста на отдельные лексемы (токены). Лексемы представляют собой минимальные синтаксические единицы языка программирования, такие как идентификаторы, ключевые слова, операторы, числа и строки.

Основная задача лексического анализа - разбить входной текст, состоящий из последовательности одиночных символов, на последовательность слов, или лексем, т.е. выделить эти слова из непрерывной последовательности символов.[1]

Лексема — это последовательность символов, включенная в исходную программу в соответствии с шаблоном соответствия токена. Это не что иное, как экземпляр токена.

Токены в конструкции компилятора представляют собой последовательность символов, которая представляет единицу информации в исходной программе.

Лексический анализатор выполняет следующие задачи:

1 Помогает идентифицировать токен в таблице символов.

2 Удаляет пробелы и комментарии из исходной программы.

3 Сопоставляет сообщения об ошибках с исходной программой.

4 Помогает вам расширить макросы, если они найдены в исходной программе.

5 Чтение входных символов из исходной программы[2]

Для реализации лексического анализатора обычно используются различные подходы, такие как регулярные выражения, конечные автоматы или генераторы лексических анализаторов

Лексический анализатор может работать или как самостоятельная фаза трансляции, или как подпрограмма, работающая по принципу "дай лексему". Оба случая представлены на рисунке 2.1.

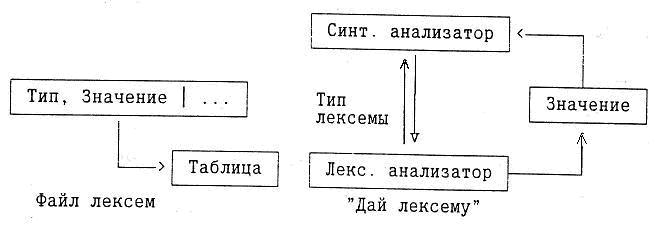


Рисунок 2.1 – Принципы работы лексического анализатора

# 3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ

В результате выполнения лабораторной работы было создан лексический анализатор, читающий данные из файла расширения .*cpp* и выводящий данные результата анализа в консоль. Для реализации используется Python 3.12 и среда разработки PyCharm 2023.3.3. Таблица переменных представлена на рисунке 3.1.



Рисунок 3.1 – Таблица переменных

Таблица констант, полученная в результате анализа, представлена на рисунке 3.2.

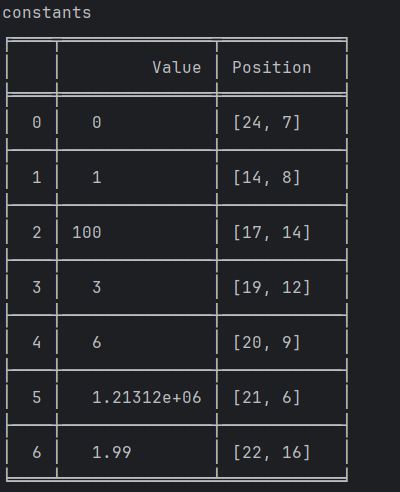


Рисунок 3.2 – Таблица констант

Таблица ключевых слов, полученная в результате анализа, представлена на рисунке 3.3.

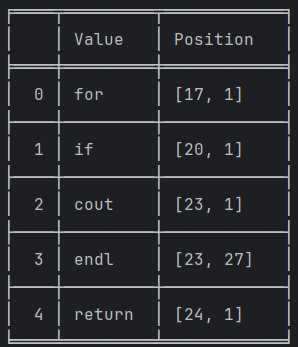
.

Рисунок 3.3 – Таблица ключевых слов

Таблица операторов, полученная в результате анализа, представлена на рисунке 3.4.

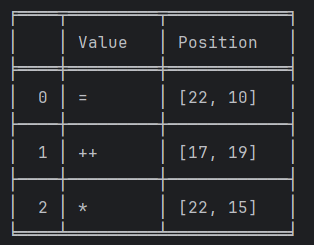


Рисунок 3.4 – Таблица операторов

Таблица функций, полученная в результате анализа, представлена на рисунке 3.4.

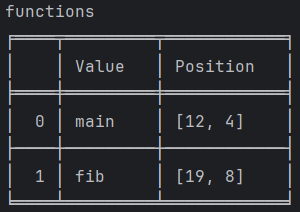


Рисунок 3.4 – Таблица функций

Таблица *stl*-контейнеров, полученная в результате анализа, представлена на рисунке 3.5.

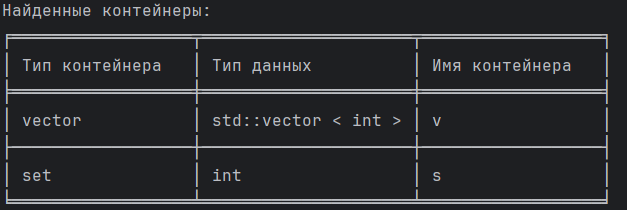


Рисунок 3.5 – Таблица контейнеров

Также программа предоставляет анализ лексических ошибок. Ниже приведены примеры нахождения таких ошибок.

На рисунке 3.6 представлен вывод в консоль при нахождении лексической ошибки, связанной с именем переменной, в которой используются не разрешенные символы.



Рисунок 3.6 – Ошибка с символами а названии переменной

На рисунке 3.7 представлен вывод в консоль при нахождении лексической ошибки, связанной с не закрытыми кавычками при объявлении строки.



Рисунок 3.7 – Ошибка с кавычками в строке

На рисунке 3.8 представлен вывод в консоль при нахождении лексической ошибки, связанной с использованием необъявленной переменной.



Рисунок 3.8 – Ошибка с необъявленной переменной

На рисунке 3.9 представлен вывод в консоль при нахождении лексической ошибки, связанной с неверным использованием операторов.



Рисунок 3.9 – Неправильное использование операторов

Это самые основные лексические ошибки которые можно встретить при анализе языка программирования C++.

## ВЫВОДЫ

В процессе выполнения данной лабораторной работы была проведена детальная аналитика и классификация ключевых аспектов модели языка программирования C++. Для этой цели был выбран Python в качестве языка для написания интерпретатора и проведения анализа. В ходе анализа были изучены переменные и константы языка программирования C++, типы данных, структуры данных, а также различные операторы и способы подключения библиотек. Полученные результаты позволили описать широкий спектр возможностей, предоставляемых языком программирования C++ для работы с различными типами данных, структурами и операторами.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Лексический анализ [Электронный ресурс]. – Режим доступа: http://edu.tsu.ru/eor/resourse/577/html/46.html. – Дата доступа: 18.02.2024.
2. Лексический анализ (Анализатор) в разработке компилятора с примером [Электронный ресурс]. – Режим доступа: https://www.guru99.com/ru/compiler-design-lexical-analysis.html – Дата доступа: 18.02.2024.

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

Листинг 1 – lexer.py  
import re

from tabulate import tabulate

from lexer.lexer\_constants import regex\_map, syntax\_types

import pandas as pd

class CustomToken:

def \_\_init\_\_(self, value, token\_type, line, column):

self.token\_type = token\_type

self.value = value

self.line = line

self.column = column

def \_\_str\_\_(self):

return f"token with value: {self.value}, type: {self.token\_type}, line {self.line}, column {self.column}"

class Lexer:

def \_\_init\_\_(self):

self.func\_list = []

self.tokens = []

self.errors = []

self.line\_num = 1

self.col\_num = 1

self.cur\_func = ""

def get\_tokens(self, code):

self.tokens = []

self.errors = []

self.line\_num = 1

self.col\_num = 1

self.cur\_func = ""

i = 0

while i < len(code):

match = None

for key, value in regex\_map.items():

tag, pattern = key, value

regex = re.compile(pattern)

match = regex.match(code[i:])

#print(key, value)

if match:

text = match.group(0)

if tag == "function":

text = text.split()

if len(self.tokens) != 0:

self.func\_list.append((self.cur\_func, self.tokens))

self.tokens = []

if not text[0] in syntax\_types:

print(f"\033[31m[ERROR] lex error: unknown '{text[0]}' at line {self.line\_num}, column {self.col\_num}\033[0m")

exit()

token = CustomToken(text[0], syntax\_types[text[0]], self.line\_num, self.col\_num)

self.tokens.append(token)

self.col\_num += len(text[0])

token = CustomToken(text[1], syntax\_types[tag], self.line\_num, self.col\_num)

self.tokens.append(token)

self.col\_num += len(text[1])

self.cur\_func = text[1]

elif tag == "whitespace":

if '\n' in text:

self.line\_num += 1

self.col\_num = 1

elif tag == "operator":

if len(text) > 2 or self.tokens[-1].token\_type == "ARITHMETIC\_OPERATION" or text not in syntax\_types:

print(

f"\033[31m[ERROR] unexpected operator '{text}' at line {self.line\_num}, column {self.col\_num}\033[0m")

exit()

self.col\_num += len(text)

else:

token = CustomToken(text, syntax\_types[text], self.line\_num, self.col\_num)

self.tokens.append(token)

self.col\_num += len(text)

elif tag == "string\_value":

token = CustomToken(text, syntax\_types[tag], self.line\_num, self.col\_num)

self.tokens.append(token)

self.col\_num += len(text)

elif tag == "identifier":

token = CustomToken(text, syntax\_types[tag], self.line\_num, self.col\_num)

for item in self.func\_list:

if item[0] == text:

token.token\_type = "FUNCTION"

self.tokens.append(token)

self.col\_num += len(text)

elif tag == "const":

if text.count('.') > 1:

print(

f"\033[31m[ERROR] unexpected value '{text}' at line {self.line\_num}, column {self.col\_num}\033[0m")

exit()

self.col\_num += len(text)

else:

token = CustomToken(text, syntax\_types[tag], self.line\_num, self.col\_num)

self.tokens.append(token)

self.col\_num += len(text)

else:

token = CustomToken(text, syntax\_types[text], self.line\_num, self.col\_num)

self.tokens.append(token)

self.col\_num += len(text)

break

if not match and not code[i] in [':', '#']:

print(f"\033[31m[ERROR] unexpected character '{code[i]}' at line {self.line\_num}, column {self.col\_num}\033[0m")

exit()

#self.col\_num += 1

#i += 1

else:

if code[i] in [':', '#']:

i+=1

else:

i += len(match.group(0))

self.func\_list.append((self.cur\_func, self.tokens))

const\_map = dict()

var\_map = dict()

build\_map = dict()

square\_map = dict()

arithmetic\_map = dict()

function\_map = dict()

cont\_list = ['vector','map','set','unordered\_map', 'list', 'deque', 'queue', 'unordered\_set']

prev = None

s\_data = []

cur\_deep = 0

for token in self.tokens:

if token.token\_type == 'VARIABLE':

if not token.value in var\_map:

if prev == None and not token.value in cont\_list:

print(f"\033[31m[ERROR] lex error: unknown {token.value} at line {token.line}, column {token.column}\033[0m")

exit()

elif prev.token\_type != 'DATA\_TYPE' and not token.value in cont\_list and token.value != 'std':

if prev.value != '>':

print(f"\033[31m[ERROR] lex error: unknown {token.value} at line {token.line}, column {token.column}\033[0m")

exit()

var\_map[token.value] = [f"id\_{len(var\_map)}", prev.value]

if token.token\_type == 'CONSTANT\_VALUE':

const\_map[token.value] = [token.line, token.column]

print(token.value, ' ---- ', token.token\_type)

if token.token\_type == 'BUILD\_IN':

build\_map[token.value] = [token.line, token.column]

if token.token\_type in ['RIGHT\_CURLY\_BRACKET', 'LEFT\_ROUND\_BRACKET', 'RIGHT\_ROUND\_BRACKET', 'LEFT\_CURLY\_BRACKET']:

if token.value in ['{', '(']:

cur\_deep += 1

else:

cur\_deep -= 1

if cur\_deep < 0:

print(f"\033[31[ERROR] unexpected end of file at line {token.line}, {token.column}\033[0m")

exit()

square\_map[token.value] = [token.line, token.column, token.token\_type]

s\_data.append({'val': token.value, 'row': token.line, 'column': token.column, 'type': token.token\_type})

if token.token\_type == 'ARITHMETIC\_OPERATION':

arithmetic\_map[token.value] = [token.line, token.column]

if token.token\_type == 'FUNCTION':

function\_map[token.value] = [token.line, token.column]

prev = token

if cur\_deep != 0:

print(f"\033[31[ERROR] unexpected end of file at line {self.line\_num}, {self.col\_num}\033[0m")

exit()

df = pd.DataFrame.from\_dict(var\_map, orient='index', columns=['id', 'data\_type'])

df.reset\_index(inplace=True)

df.columns = ['variable', 'id', 'data\_type']

print('vars')

print(tabulate(df, headers='keys', tablefmt='fancy\_grid'))

df = pd.DataFrame(const\_map.items(), columns=['Value', 'Position'])

print('constants')

print(tabulate(df, headers='keys', tablefmt='fancy\_grid'))

df = pd.DataFrame(build\_map.items(), columns=['Value', 'Position'])

print('builds')

print(tabulate(df, headers='keys', tablefmt='fancy\_grid'))

df = pd.DataFrame(s\_data)

print('square')

print(tabulate(df, headers='keys', tablefmt='fancy\_grid'))

df = pd.DataFrame(arithmetic\_map.items(), columns=['Value', 'Position'])

print('arithmetic operators')

print(tabulate(df, headers='keys', tablefmt='fancy\_grid'))

df = pd.DataFrame(function\_map.items(), columns=['Value', 'Position'])

print('functions')

print(tabulate(df, headers='keys', tablefmt='fancy\_grid'))

pattern = r'#include\s\*<([^\s>]\*)>'

libraries = re.findall(pattern, code)

print("Подключенные библиотеки:")

for library in libraries:

print(library)

pattern = r'std::\s\*(vector|map|set|unordered\_map|list|queue|deque)\s\*<\s\*(.\*?)\s\*>\s\*(\w+)'

table\_data = []

headers = ["Тип контейнера", "Тип данных", "Имя контейнера"]

matches = re.findall(pattern, code)

for match in matches:

container\_type = match[0]

container\_data\_type = match[1]

container\_name = match[2]

table\_data.append([container\_type, container\_data\_type, container\_name])

# print(f"Тип контейнера: {container\_type}, Тип данных: {container\_data\_type}, Имя контейнера: {container\_name}")

table = tabulate(table\_data, headers, tablefmt="fancy\_grid")

print("Найденные контейнеры:")

print(table)

return self.func\_list, self.errors

Листинг 2 – Lexer\_constants.py

syntax\_types = {

"int": "DATA\_TYPE",

"float": "DATA\_TYPE",

"double": "DATA\_TYPE",

"char": "DATA\_TYPE",

"void": "DATA\_TYPE",

"string": "DATA\_TYPE",

"identifier": "VARIABLE",

"const": "CONSTANT\_VALUE",

"string\_value": "STRING\_CONST",

'comment': 'COMMENT',

"if": "BUILD\_IN",

"rand": "BUILD\_IN",

"else": "BUILD\_IN",

"while": "BUILD\_IN",

"for": "BUILD\_IN",

"break": "BUILD\_IN",

"continue": "BUILD\_IN",

"return": "BUILD\_IN",

"sizeof": "BUILD\_IN",

"cout": "BUILD\_IN",

"endl": "BUILD\_IN",

"[": "LEFT\_SQUARE\_BRACKET",

"]": "RIGHT\_SQUARE\_BRACKET",

"{": "LEFT\_CURLY\_BRACKET",

"}": "RIGHT\_CURLY\_BRACKET",

"(": "LEFT\_ROUND\_BRACKET",

")": "RIGHT\_ROUND\_BRACKET",

",": "COMMA",

";": "SEMICOLON",

"+": "ARITHMETIC\_OPERATION",

"-": "ARITHMETIC\_OPERATION",

"++": "ARITHMETIC\_OPERATION",

"--": "ARITHMETIC\_OPERATION",

"\*": "ARITHMETIC\_OPERATION",

"/": "ARITHMETIC\_OPERATION",

"%": "ARITHMETIC\_OPERATION",

"=": "ARITHMETIC\_OPERATION",

"+=": "ARITHMETIC\_OPERATION",

"\*=": "ARITHMETIC\_OPERATION",

"/=": "ARITHMETIC\_OPERATION",

"-=": "ARITHMETIC\_OPERATION",

"<": "COMPARISON\_SIGN",

">": "COMPARISON\_SIGN",

"<=": "COMPARISON\_SIGN",

">=": "COMPARISON\_SIGN",

"==": "COMPARISON\_SIGN",

"!=": "COMPARISON\_SIGN",

"<<": "OVERRIDE\_OPERATION",

">>": "OVERRIDE\_OPERATION",

"function": "FUNCTION"

}

nodes = (

"Block\_node",

"Else\_node",

"If\_node",

"While\_node",

"For\_node",

"Expression\_node",

"Func\_declaration\_node",

"Variable\_node",

"Func\_node",

"Compare\_node",

"Assign\_node",

"Build\_in\_node"

)

tags = (

('DATA\_TYPE'),

('VARIABLE'),

('CONSTANT\_VALUE'),

('STRING\_CONST'),

('COMMENT'),

('BUILD\_IN'),

('LEFT\_SQUARE\_BRACKET', 'RIGHT\_SQUARE\_BRACKET', 'LEFT\_CURLY\_BRACKET', 'RIGHT\_CURLY\_BRACKET', 'LEFT\_ROUND\_BRACKET',

'RIGHT\_ROUND\_BRACKET'),

('COMMA', 'SEMICOLON'),

('ARITHMETIC\_OPERATION'),

('COMPARISON\_SIGN'),

('OVERRIDE\_OPERATION'),

('FUNCTION')

)

regex\_map = {

'function': r'(\w+)\s+(\w+)(?=\()([^)]\*?)\s\*',

'int': r'int\b',

'float': r'float\b',

'double': r'double\b',

'char': r'char\b',

'bool': r'bool\b',

'string': r'string\b',

'if': r'if\b',

'else': r'else\b',

'for': r'for\b',

'while': r'while\b',

'do': r'do\b',

'break': r'break\b',

'continue': r'continue\b',

'return': r'return\b',

'void': r'void\b',

'sizeof': r'sizeof\b',

'cin': r'cin\b',

'rand': r'rand\b',

'cout': r'cout\b',

'endl': r'endl\b',

'identifier': r'[a-zA-Z\_]\w\*',

'const': r'\d+(\.\d+)\*',

'string\_value': r'\".+?\"',

'operator': r'[+\-\*/%<>&|^=!]+|<<|>>',

'semicolon': r';',

'comma': r',',

'comment': r'//.\*?$|/\\*.\*?\\*/',

'whitespace': r'[\t|\n|\s|\r]+',

'left\_round\_bracket': r'\(',

'right\_round\_bracket': r'\)',

'left\_square\_bracket': r'\[',

'right\_square\_bracket': r'\]',

'left\_curly\_bracket': r'\{',

'right\_curly\_bracket': r'\}'

}

c\_plus\_plus\_libraries = [

"#include<iostream>",

"#include<algorithm>",

"#include<vector>",

"#include<map>",

"#include<set>",

"#include<queue>",

"#include<stack>",

"#include<cmath>",

"#include<cstring>",

"#include<fstream>",

"#include<sstream>",

"#include<cstdlib>",

"#include<ctime>",

"#include<bitset>",

"#include<unordered\_map>"

]

Листинг 3 – test.cpp

#include<iostream>

#include<set>

#include<map>

int fib(int n){

int result = 1;

for (int i = 2; i <= n; i++){

result \*= i;

}

kk = 1;

return result;

}

int main(){

int val = 0;

int val{1};

std::vector < std::vector < int > > v;

std::set < int > s;

for (int k = 0; k < 100; k++){

}

int res = fib(3)

if (res == 6){

int n = 1213124;

int answer = (n \* n \* 1.99);

cout << "n \* n = " << answer << endl;

return 0;

}

}

Листинг 4 – errors\_test.cpp

int fib(int n){

int result = 1;

for (int i = 2; i <= n; i++){

result \*= i;

}

return result;

}

int main(){

int @val = 1;

var val = 1;

val+++;

string str = "123;

int res = fib(3);

if (res == 6){

string n = 1213124;

int answer = (n \* n \* 1.99);

cout << "n \* n = " << answer << endl;

return 0;

}

}