**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

Факультет инфокоммуникационных сетей и систем (ИКСС)

Лабораторная работа №5

по курсу

«Теория автоматов и формальных языков»

Группа: ИКПИ-14

Выполнил студент: Хохлов Т. В.

Принял преподаватель: Леонова М.Д.

Место для подписи \_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2023 г.

**Задание:**

Для вашего любимого языка программирования составить таблицу

токенов аналогично таблице выше. Таблица может не быть

исчерпывающей.

Для каждого типа токенов придумать и описать паттерн, которому

соответствуют значения такого токена. Для этих целей можно

использовать регулярные выражения или другой способ, кажущийся

вам более подходящим.

Написать программу, которая будет получать на вход строку на вашем

любимом языке программирования, а на выход выдавать

последовательность токенов (с указанием как имени токена, так и его

значения).

**Токены:**

|  |  |  |
| --- | --- | --- |
| Имя токена | Пример значений | Шаблон (regexpr) |
| Preprocessor | #include, #pragma\_once | r’#\b(?:include|define|if|endif|ifdef|ifndef|pragma)\b’ |
| header | <iostream>, <ctime> | r’<[A-Za-z\_][A-Za-z0-9\_]\*>‘ |
| keyword | If, else, int | r’\b(?:if|else|while|for|return|int|float|double|char|bool|void|class)\b’ |
| identifier | Var, I, \_precool | r’[A-Za-z\_][A-Za-z0-9\_]\*’ |
| Operator | +, -, !=, == | r’[-+\*/=<>!]?=|=?|&&|\|\|’ |
| separator | {, (, ) | r’[{}();,]’ |
| Literal\_number | 0, 25, 145 | r’\b\d+(\.\d\*)?|\b0x[0-9A-Fa-f]+\b’ |
| Literal\_string | Hehe, Hoho, HAha\n | r’"(?:\\.|[^"\\])\*"|\’(?:\\.|[^\’\\])\’’ |
| Comment | //Hehe /\*Haha\*/ | r’\/\\*[\s\S]\*?\\*\/|\/\/.\*’ |
| whitespace | ( ) ( ) | r’\s+’ |

**RegExpr(s):**

**Препроцессорные директивы**

**r’#\b(?:include|define|if|endif|ifdef|ifndef|pragma)\b’**

#: Начинается с символа #.

\b: Граница слова, чтобы удостовериться, что # не является частью другого слова.

(?: ... ): Группировка без создания подмаски. Здесь перечислены директивы:

include

define

if

endif

ifdef

ifndef

pragma

\b: Снова граница слова, чтобы убедиться, что после директивы нет лишних символов.

**Заголовочные файлы**

**r’<[A-Za-z\_][A-Za-z0-9\_]\*>‘**

<: Начало угловой скобки.

[A-Za-z\_]: Первый символ должен быть буквой или подчеркиванием.

[A-Za-z0-9\_]\*: Остальные символы могут быть буквами, цифрами или подчеркиваниями (может быть 0 или более).

>: Конец угловой скобки.

**Ключевые слова**

**r’\b(?:if|else|while|for|return|int|float|double|char|bool|void|class)\b’**

\b: Граница слова перед ключевым словом.

(?: ... ): Группировка, в которой перечислены ключевые слова:

if, else, while, for, return, int, float, double, char, bool, void, class

\b: Граница слова после ключевого слова.

**Идентификаторы**

**r’[A-Za-z\_][A-Za-z0-9\_]\*’**

[A-Za-z\_]: Первый символ должен быть буквой или подчеркиванием.

[A-Za-z0-9\_]\*: Остальные символы могут быть буквами, цифрами или подчеркиваниями (может быть 0 или более).

**Операторы**

**r’[-+\*/=<>!]?=|=?|&&|\|\|’**

[-+\*/=<>!]: Односимвольные операторы. Включает:

-, +, \*, /, =, <, >, !

=?: Следом может идти символ =, чтобы получить операторы <=, >=, ==, !=.

||: Двойной символ для логического "или".

&&: Двойной символ для логического "и".

**Разделители**

**r’[{}();,]’**

[{}();,]: Совпадает с одним из перечисленных символов: {}, (, ), ;, ,.

**Литералы чисел**

**r’\b\d+(\.\d\*)?|\b0x[0-9A-Fa-f]+\b’**

\b: Граница слова.

\d+: Одно или более цифровых символов (целое число).

(\.\d\*)?: Опционально, десятичная точка, за которой может следовать 0 или более цифр.

|: Логический "или".

0x: Начало шестнадцатеричного числа.

[0-9A-Fa-f]+: Одно или более символов (цифры от 0 до 9 и буквы A-F, a-f).

\b: Граница слова в конце.

**Строковые литералы**

**r’"(?:\\.|[^"\\])\*"|\’(?:\\.|[^\’\\])\’’**

": Начало строки в двойных кавычках.

(?: ... ): Группировка:

\\.: Экранированные символы (например, \", \\).

[^"\\]: Любые символы, кроме двойной кавычки и обратного слэша.

\*: 0 или более таких символов.

": Конец строки в двойных кавычках.

|: Логическое "или".

\’: Начало строки в одинарных кавычках.

(?:\\.|[^\’\\]): То же самое для одинарных кавычек.

\’: Конец строки в одинарных кавычках.

**Комментарии**

**r’\/\\*[\s\S]\*?\\*\/|\/\/.\*’**

\/\\*: Начало многострочного комментария /\*.

[\s\S]\*?: Любые символы (включая пробелы), 0 или более раз, с ленивым совпадением (пока не встретится \*/).

\\*\/: Конец многострочного комментария \*/.

|: Логическое "или".

\/\/.\*: Однострочные комментарии, начинающиеся с // и продолжающиеся до конца строки.

**Пробелы**

**r’\s+’**

\s: Совпадает с любым пробельным символом (пробел, табуляция, новая строка и т.д.).

+: Один или более пробельных символов.

**Программа:**

import re

token\_specification = [

(‘preprocessor’, r’#\b(?:include|define|if|endif|ifdef|ifndef|pragma)\b’), # Препроцессорные директивы

(‘header’, r’<[A-Za-z\_][A-Za-z0-9\_]\*>‘), # Заголовочные файлы внутри <>

(‘keyword’, r’\b(?:if|else|while|for|return|int|float|double|char|bool|void|class)\b’), # Ключевые слова

(‘identifier’, r’[A-Za-z\_][A-Za-z0-9\_]\*’), # Идентификаторы (переменные, функции и т.д.)

(‘operator’, r’[-+\*/=<>!]=?|&&|\|\|’), # Операторы

(‘separator’, r’[{}();,]’), # Разделители

(‘literal\_number’, r’\b\d+(\.\d\*)?|\b0x[0-9A-Fa-f]+\b’), # Литералы чисел

(‘literal\_string’, r’"(?:\\.|[^"\\])\*"|\’(?:\\.|[^\’\\])\’’), # Строковые литералы

(‘comment’, r’\/\\*[\s\S]\*?\\*\/|\/\/.\*’), # Комментарии

(‘whitespace’, r’\s+’), # Пробелы игнорируются

]

# Компиляция регулярных выражений

token\_regex = ‘|’.join(f’(?P<{name}>{pattern})’ for name, pattern in token\_specification)

get\_token = re.compile(token\_regex)

def lex\_analysis(text):

tokens = []

for match in re.finditer(get\_token, text):

token\_type = match.lastgroup

token\_value = match.group(token\_type)

if token\_type != ‘whitespace’: # Пропуск пробелов

tokens.append((token\_type, token\_value))

return tokens

# Тест

filename = input("Введите название файла для лексического разбора: ")

with open(filename, ‘r’, encoding=‘utf-8’) as file:

input\_code = file.read()

tokens = lex\_analysis(input\_code)

for token in tokens:

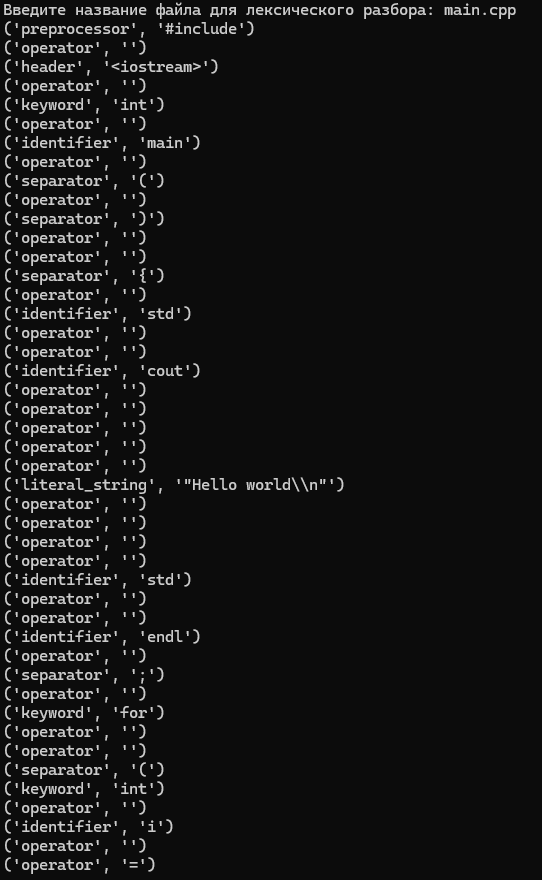
print(token)

**Пример работы программы:**

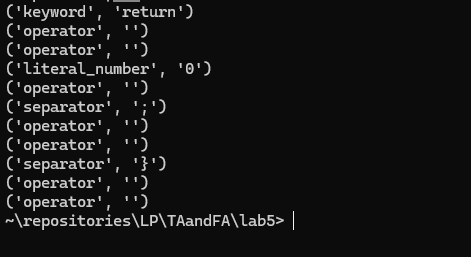
Файл с кодом c++:



Результат работы программы:







**Вывод:**

Я научился разрабатывать лексический анализатор, который разбивает код на токены, такие как идентификаторы, ключевые слова и операторы. Освоил использование регулярных выражений для обработки различных языковых элементов. Теперь у меня есть практические навыки написания лексера на Python.