**Practical - 1**

**Aim: Write a program to implement lexical analyzer to recognize few patterns of C/C++/Java/Python language. The lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments.**

**Code:**

***RUST***

***lib.rs***

pub mod lexical\_analyser;

***lexical\_analyser.rs***

use std::collections::{HashMap, HashSet};

const KEYWORDS: &[&str] = &[

"auto", "break", "case", "char", "const", "continue", "default", "do", "double", "else",

"enum", "extern", "float", "for", "goto", "if", "int", "long", "register", "return", "short",

"signed", "sizeof", "static", "struct", "switch", "typedef", "union", "unsigned", "void",

"volatile", "while", "printf", "scanf", "%d", "include", "stdio.h", "main",

];

const OPERATORS: &[&str] = &["=", "!", "~", "+", "-", "\*", "/", "%", "^", ","];

const DELIMITERS: &[&str] = &[

"{", "}", "(", ")", "[", "]", ".", "&", "|", ",", "#", ";", ":", "",

];

pub fn parse(lines: &mut Vec<String>) -> HashMap<String, HashSet<String>> {

let mut keywords = HashSet::new();

let mut operators = HashSet::new();

let mut delimiters = HashSet::new();

let mut identifiers = HashSet::new();

let mut constants = HashSet::new();

for word in lines.clone() {

if KEYWORDS.contains(&word.as\_str()) {

keywords.insert(word.to\_string());

}

}

for word in lines.clone() {

if OPERATORS.contains(&word.as\_str()) {

operators.insert(word.to\_string());

}

}

for word in &lines.clone() {

if DELIMITERS.contains(&word.as\_str()) {

delimiters.insert(word.to\_string());

}

}

for word in lines.clone() {

let mut number = String::new();

for c in word.chars() {

if c.is\_digit(10) {

number.push(c);

} else {

if !number.is\_empty() {

constants.insert(number);

number = String::new();

}

}

}

}

for word in lines.clone() {

if word.chars().all(|c| c.is\_alphanumeric()) {

identifiers.insert(word.to\_string());

}

}

let mut result = HashMap::new();

result.insert("keywords".to\_string(), keywords);

result.insert("operators".to\_string(), operators);

result.insert("delimiters".to\_string(), delimiters);

result.insert("identifiers".to\_string(), identifiers);

result.insert("constants".to\_string(), constants);

result

}

***main.rs***

use lexical\_analyser\_rs::lexical\_analyser;

use std::collections::{HashMap, HashSet};

use std::fs::File;

use std::io::{Read, Write};

fn read\_file(filename: &str) -> Result<String, std::io::Error> {

let mut file = File::open(filename)?;

let mut lines = String::new();

file.read\_to\_string(&mut lines)?;

Ok(lines)

}

fn main() {

let lines = read\_file("test.c").unwrap();

let mut words = vec];

for word in lines.split\_whitespace() {

words.push(word.to\_string());

}

let result = lexical\_analyser::parse(&mut words);

println!("{:#?}", result);

// write to file

write\_to\_file(&result);

}

fn write\_to\_file(result: &HashMap<String, HashSet<String>>) {

for (key, value) in result {

let filename: String = key.to\_string() + ".txt";

let mut file = File::create(filename).unwrap();

for word in value {

file.write\_all(word.as\_bytes()).unwrap();

file.write\_all(", ".as\_bytes()).unwrap();

}

file.write\_all("\n".as\_bytes()).unwrap();

}

}

***test.c***

#include <stdio.h>

void main()

{

int a, e = 10;

float b = 2.5;

float c;

c = a \* b;

return 0;

}

**Output on stdout:**

**![Text, table

Description automatically generated with medium confidence**

***constants.txt***

***Graphical user interface, text, application, chat or text message

Description automatically generated***

***delimiters.txt***

***Graphical user interface, text, application, chat or text message

Description automatically generated***

***identifiers.txt***

***Graphical user interface, text, application

Description automatically generated***

***keywords.txt***

***Graphical user interface, text, application

Description automatically generated***