Министерство образования Республики Беларусь
Учреждения образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

# Отчет по лабораторной работе №2

## По теме «Лексический анализ программ.»

Выполнил:
студент гр. 053501
Шебеко Ю.А.

Проверил:
Гриценко Н. Ю.

Минск 2023

СОДЕРЖАНИЕ

## 1. Цель работы

Разработка лексического анализатора подмножества языка программирования, определенного в лабораторной работе 1. Программа определяет лексические правила и выполняет перевод потока символов программы лабораторной работы 1 в поток лексем(токенов).

## 2.Теория

Лексический анализатор (или сканер) – часть компилятора, которая читает литеры программы на исходном языке и строит из них слова (лексемы) исходного языка. На вход лексического анализатора поступает текст исходной программы, а выходная информация передается для дальнейшей обработки компилятором на этапе синтаксического анализа и разбора.

Лексема (лексическая единица языка) – структурная единица языка, которая состоит из элементарных символов языка и не содержит в своем составе других структурных единиц языка. Лексемами языков программирования являются идентификаторы, константы, ключевые слова языка, знаки операций и т.п. Состав возможных лексем каждого конкретного языка программирования определяется синтаксисом этого языка.

С теоретической точки зрения лексический анализатор не является обязательной, необходимой часть компилятора. Его функции могут выполняться на этапе синтаксического анализа. Однако существует несколько причин, исходя из которых в состав практически всех компиляторов включают лексический анализ:

1) Упрощается работа с текстом исходной программы на этапе синтаксического разбора и сокращается объем обрабатываемой информации, так как лексический анализатор структурирует поступающий на вход исходный текст программы и удаляет всю незначащую информацию;

2) Лексический анализатор отделяет сложный по конструкции синтаксический анализатор от работы непосредственно с текстом исходной программы, структура которого может варьироваться в зависимости от версии входного языка – при такой конструкции компилятора при переходе от одной версии языка к другой достаточно только перестроить относительно простой лексический анализатор.

## 3.Демонстрация работы

```
keyWords = {"while", "for", "if", "else", "int",
            "float", "break", "continue", "double",
            "array", "false", "true"}
operators = {"+", "-", "*", "/", "%",
             "=", ">", "<", ">=", "<=", "==", "!=",
             "++", "--", "+=", "-=", "*=", "/="}
types = {"int ", "float ", "double ", "boolean ", "char ", "string "}
par = {"(", ")", "{", "}"}
```

Рисунок 3.1 - Перечисление основных констант

На рисунке 3.1 представлен основной набор констант выбранного языка программирования (Java).

В качестве входных данных рассмотрим задачу о нахождении первых 11 членов последовательности Фибоначчи.

```
+ ---------------------------------------- +
|            VARIABLES TABLE                |
+ ---------------------------------------- +
|     n0   |variable of "int " type         |
|     n1   |variable of "int " type         |
|     n2   |variable of "int " type         |
+ ---------------------------------------- +
```

Рисунок 3.2 - Таблица переменных

```
+ ---------------------------------------- +
|            KEY WORDS TABLE                |
+ ---------------------------------------- +
|   for    | key word "for"  -- 1           |
|   int    | key word "int"  -- 4           |
+ ---------------------------------------- +
```

Рисунок 3.3 - Таблица ключевых слов

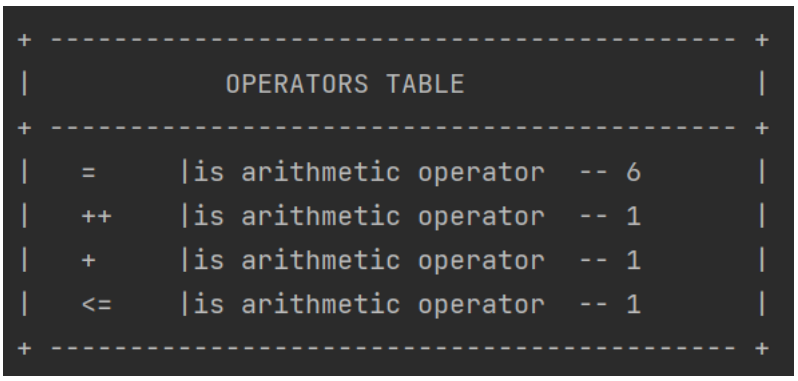


Рисунок 3.3 - Таблица операторов

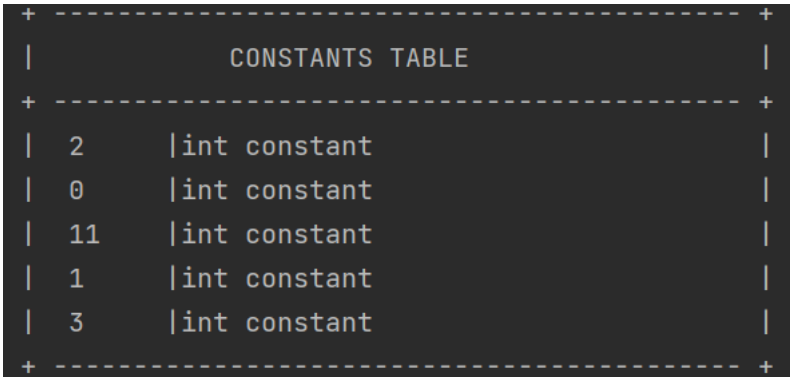


Рисунок 3.4 - Таблица констант

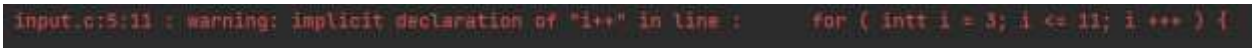


Рисунок 3.5 - Лексическая ошибка “;”

На рисунке 3.5 продемонстрирована ошибка отсутствия “;”.


Рисунок 3.6 - Лексическая ошибка “int”

На рисунке 3.6 продемонстрирована лексическая ошибка, не верно указан тип переменной.


Рисунок 3.7 - Лексическая ошибка “i++”

На рисунке 3.7 продемонстрирована лексическая ошибка, не верно указан инкремент.


Рисунок 3.8 - Лексическая ошибка неверное количество скобок

## Приложение. Текст программ

```
import re
import sys


def out_red(text):
    print("\033[31m {}" .format(text))


keyWords = {"while", "for", "if", "else", "int",
            "float", "break", "continue", "double",
            "array", "false", "true"}
loop = {"while", "for"}
operators = {"+", "-", "*", "/", "%",
             "=", ">", "<", ">=", "<=", "==", "!=",
             "++", "--", "+=", "-=", "*=", "/="}
types = {"int ", "float ", "double ", "boolean ", "char ", "string "}
par = {"(", ")", "{", "}"}
prints = {"System.out.println"}
f = open('D://java.txt', 'r')
text = f.read()
one_word = text.split()
print(text)
f.close()
print("\n+", "-"*43, "+")
print("|", " "*10, "VARIABLES TABLE", " "*16, "|")
print("+", "-"*43, "+")
for words in types:
    count = 0
    s = []
    for word in text.split("\n"):
        if words in word and word.split()[0] not in loop:
            count = count + 1
            i = word.index(words)
            s.append(word.split()[i+1])
            continue
    if count > 0:
        for i in range(len(s)):
            print("|     %s   |variable of \"%s\" type  "%(s[i], words),
" "*7, "|")
print("+", "-"*43, "+")

print("\n+", "-"*43, "+")
print("|", " "*10, "KEY WORDS TABLE", " "*16, "|")
print("+", "-"*43, "+")
for words in keyWords:
    count = 0
    for word in one_word:
        if words == word:
            count = count + 1
    if count > 0:
        print("|   %s    | key word \"%s\"  -- %d             |"%(words,
words, count))
print("+", "-"*43, "+")

print("\n+", "-"*43, "+")
print("|", " "*10, "OPERATORS TABLE", " "*16, "|")
print("+", "-"*43, "+")
for words in operators:
    count = 0
    for word in one_word:
        if words == word:
            count = count + 1
```

```
            if words == "==":
                n = 2
                test = "is comparison operator"
            elif words == "<=" or words == "++" or words == "--":
                n = 2
                test = "is arithmetic operator"
            else:
                n = 3
                test = "is arithmetic operator"
    if count > 0:
        if words == "==":
            print("|   %s"%(words), " "*n, "|%s  -- %d"%(test, count), "
"*5, "|")
        else:
            print("|   %s" % (words), " " * n, "|%s  -- %d" % (test,
count), " " * 5, "|")
print("+", "-"*43, "+")

print("\n+", "-"*43, "+")
print("|", " "*10, "CONSTANTS TABLE", " "*16, "|")
print("+", "-"*43, "+")
for words in types:
    count = 0
    s = []
    for word in text.split("\n"):
        if words in word:
            count = count + 1
            i = word.index(words)
            s.append(word.split()[i+1])
            continue
    if count > 0:
        for i in range(len(s)-1):
            print("|  %s    |const of %s type "%(s[i], words), " "*15,
"|")
t = list(set(re.findall(r'\d+', text)))
for i in range(len(t)):
    if len(t[i]) > 1:
        print("|  %s    |int constant  "%(t[i]), " "*20, "|")
    else:
        print("|  %s     |int constant  " % (t[i]), " " * 20, "|")
print("+", "-"*43, "+")

print("\nERROR:")
line = 0
for word in text.split("\n"):
    line = line+1
    s = word.split()
    if s[-1] != ";" and s[-1] != "{" and s[-1] != "}":
        out_red("input.c:%d:%d : error: expected expression \";\" in
line :  %s" % (line, len(s)+1, word.replace(" ", "")))
        sys.exit()

line = 0
pos = 0
for word in text.split("\n"):
    line = line + 1
    pos = 0
    s = word.split()
    for letter in s:
        pos = pos + 1
        if len(letter) >= 3 and letter[0] == "i":
            if letter[1] != "n" or letter[2] != "t" or len(letter) != 3:
                out_red("input.c:%d:%d : warning: implicit declaration
```

```
of type \"int\" in line :  %s  " % (line, pos, word))
                sys.exit()

line = 0
pos = 0
for word in text.split("\n"):
    line = line + 1
    pos = 0
    s = word.split()
    for letter in s:
        pos = pos + 1
        if len(letter) >= 2 and letter[0] == "+":
            if letter[1] != "+" or len(letter) != 2:
                out_red("input.c:%d:%d : warning: implicit declaration
of \"i++\" in line :  %s  " % (line, pos, word))
                sys.exit()

line = 0
pos = 0
rpar = "("
lpar = ")"
count = 0
for word in text.split("\n"):
    line = line + 1
    pos = 0
    count = 0
    s = word.split()
    for letter in s:
        pos = pos + 1
        if letter == rpar:
            count = count + 1
        if letter == lpar:
            count = count - 1
    if count != 0:
        out_red("input.c:%d:%d : warning: invalid lexical per statement
in line :  %s  " % (line, pos, word))
        sys.exit()
```