| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |
| --- | --- |

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.03 Прикладная информатика.**

**ОТЧЕТ**

| **по домашнему заданию №** | 2 |
| --- | --- |

Название: Лексические и синтаксические анализаторы

**Дисциплина: Машинно-зависимые языки и основы компиляции**

| Студент |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Я.С. Петрова |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2023

**Цель работы:**  закрепление знаний теоретических основ и основных методов приемов разработки лексических и синтаксических анализаторов регулярных и контекстно свободных формальных языков.

**Задание**

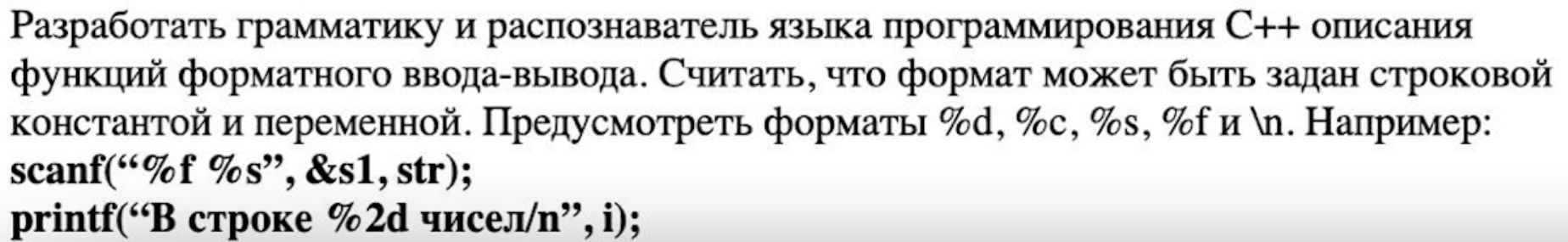


Рисунок 1 – Текст задания согласно варианту

# 

# Ход выполнения

1) Опишем грамматику в форме Бэкуса–Наура:

<основа> ::= <функция>(<форматная строка>, <параметры>);

<функция> ::= scanf|printf

<форматная строка> ::= "<форматы><форматная строка>"|"<форматная строка><форматы>"|"<последовательность символов>"

<параметры> ::= <имя>, <параметры>|<имя>

<форматы> ::= <формат><форматры>|<формат>

<последовательность символов> ::= <литерал>|<последовательность символов><литерал>

<имя> ::= <буква><последовательность символов>|<буква>

<формат> ::= %<спецификатор>|<перенос строки>

<литерал> ::= <буква>|<цифра>

<буква> ::= a|b|c|...|Z|\_

<спецификатор> ::= d|c|s|f

<цифра> ::= 0|1|2|3|4|5|6|7|8|9

<перенос строки> ::= \n

Грамматика является регулярной (т.е. 3го типа по Хомскому), т.к. использует правила вида A->α, A->αB или A->Bα, где A, ВVN; αVT.

2) Поскольку грамматика 3го типа, то возможно применить синтаксический анализ посредством построения конечного автомата.

Разработаем синтаксическую диаграмму для проверки правильности составление прототипа функции.

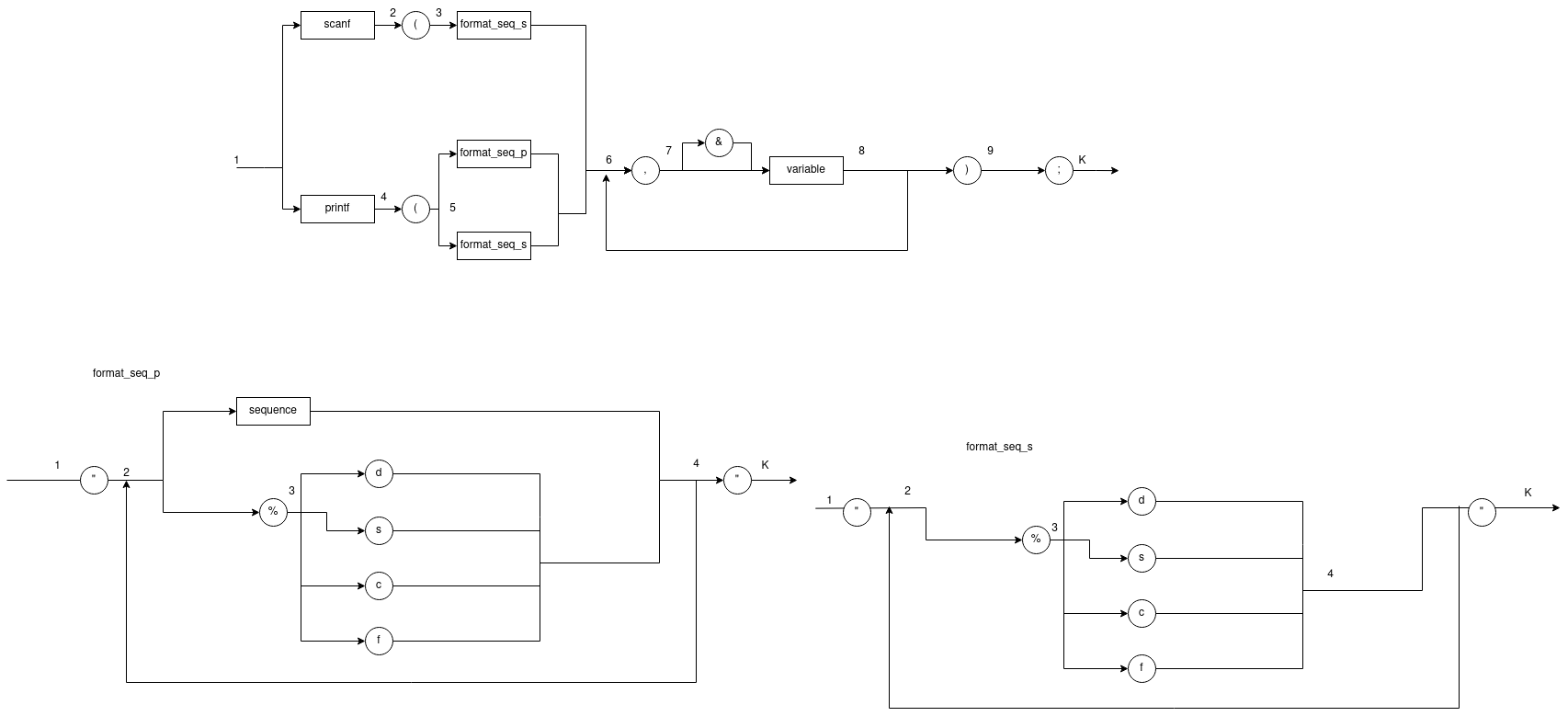


Рисунок 2 – Синтаксические диаграммы проверки правильности записи

Таблица 1. Переходы состояний для автомата, описывающего саму запись io.

| Состояние | scanf | printf | ( | format\_seq\_s | format\_seq\_p | , | & | variable | ) | ; |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 4 | E | E | E | E | E | E | E | E |
| 2 | E | E | 3 | E | E | E | E | E | E | E |
| 3 | E | E | E | 6 | E | E | E | E | E | E |
| 4 | E | E | 5 | E | E | E | E | E | E | E |
| 5 | E | E | E | 6 | 6 | E | E | E | E | E |
| 6 | E | E | E | E | E | 7 | E | E | E | E |
| 7 | E | E | E | E | E | E | 7 | 8 | E | E |
| 8 | E | E | E | E | E | 7 | E | E | 9 | E |
| 9 | E | E | E | E | E | E | E | E | E | K |

Таблица 2. Переходы состояний для автомата, описывающего текст в кавычках.

| Состояние | sequence | % | d | s | c | f | " | digit |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | E | E | E | E | E | E | 2 | E |
| 2 | 4 | 3 | E | E | E | E | E | E |
| 3 | E | E | 4 | 4 | 4 | 4 | E | 3 |
| 4 | 4 | 3 | E | E | E | E | K | E |

3) Напишем программу на языке Python 3.6, выполняющую лексический и синтаксический анализ заданного прототипа функции ввода-вывода. Код программы представлен в листинге ниже.

Листинг 1 – Код программы на Python 3.6

import io

import tokenize as t

from operator import itemgetter

import string

letters = string.ascii\_lowercase + string.ascii\_uppercase + "\_"

serv\_words = {

"(": "(\_open\_par",

")": ")\_close\_par",

",": ",\_comma",

";": ";\_semicolon",

"scanf": "s\_func",

"printf": "p\_func",

"&": "&\_ampersand"

}

transition\_io = {

"s\_func": [1, "E", "E", "E", "E", "E", "E", "E", "E", "E"],

"p\_func": [3, "E", "E", "E", "E", "E", "E", "E", "E", "E"],

"(\_open\_par": ["E", 2, "E", 4, "E", "E", "E", "E", "E"],

"format\_seq\_s": ["E", "E", 5, "E", 5, "E", "E", "E", "E"],

"format\_seq\_p": ["E", "E", "E", "E", 5, "E", "E", "E", "E"],

",\_comma": ["E", "E", "E", "E", "E", 6, "E", 6, "E"],

"&\_ampersand": ["E", "E", "E", "E", "E", "E", 6, "E", "E"],

"variable": ["E", "E", "E", "E", "E", "E", 7, "E", "E"],

")\_close\_par": ["E", "E", "E", "E", "E", "E", "E", 8, "E"],

";\_semicolon": ["E", "E", "E", "E", "E", "E", "E", "E", "K"]

}

serv\_words\_quoted\_text = {

"%": "%\_percentage",

"d": "d\_format",

"s": "s\_format",

"c": "c\_format",

"f": "f\_format",

'"': '"\_quote'

}

format\_printf\_transition = {

"sequence": ["E", 3, "E", 3],

"%\_percentage" : ["E", 2, "E", 2],

"d\_format" : ["E", "E", 3, "E"],

"s\_format": ["E", "E", 3, "E"],

"c\_format": ["E", "E", 3, "E"],

"f\_format": ["E", "E", 3, "E"],

'"\_quote': [1, "E", "E", "K"],

"digit": ["E", "E", 2, "E"]

}

format\_scanf\_transition = {

"%\_percentage" : ["E", 2, "E", 2],

"d\_format" : ["E", "E", 3, "E"],

"s\_format": ["E", "E", 3, "E"],

"c\_format": ["E", "E", 3, "E"],

"f\_format": ["E", "E", 3, "E"],

'"\_quote': [1, "E", "E", "K"],

"digit": ["E", "E", 2, "E"]

}

def split(data):

tokens = t.tokenize(io.BytesIO(data.strip().encode()).readline)

next(tokens) # первый токен содержит информацию о кодировке файла/потока

return list(map(itemgetter(1), tokens))[:-2]

def tokenize(data):

res = [serv\_words.get(word) for word in data] # заменили все слова

for i in range(len(res)):

if res[i] is None:

if check\_sequence(data[i]):

res[i] = "variable"

else:

if data[i][0] == '"' and data[i][-1] == '"':

# подготовка к токенизации кавычек

splited\_quoted\_string = split(data[i][1:-1])

splited\_quoted\_string.insert(0, '"') # добавляем в лист токенов в начало кавычки

splited\_quoted\_string.append('"') # добавляем в лист токенов в конец кавычки

token\_quoted\_string = [serv\_words\_quoted\_text.get(word) for word in splited\_quoted\_string] # заменили все слова на токены

# если есть None, то эти элементы заменяем на sequence

for j in range(len(token\_quoted\_string)):

if token\_quoted\_string[j] is None:

if splited\_quoted\_string[j].isdigit():

token\_quoted\_string[j] = "digit"

else:

token\_quoted\_string[j] = "sequence"

res[i] = what\_format\_sequence(token\_quoted\_string)

if res[i] is None:

print("Ошибка в форматной строке")

exit(0)

# должен быть вызов функции проверки формата (текста в кавычках)

else:

print("Синтаксическая ошибка в слове", data[i] + data[i + 1])

exit(0)

return res

def check\_sequence(seq):

if seq[0] in letters:

return True

else:

return False

def what\_format\_sequence(seq):

state = 0

if "sequence" in seq:

# начинаем проверку автомата текста в кавычках для printf

for elem in seq:

state = format\_printf\_transition.get(elem)[state]

if state == "E":

return None

return "format\_seq\_p"

else:

# начинаем проверку автомата текста в кавычках для scanf

for elem in seq:

state = format\_scanf\_transition.get(elem)[state]

if state == "E":

return None

return "format\_seq\_s"

try:

line = split(input("Введите строку: "))

print("Введенная строка:", line)

line\_tok = tokenize(line)

print("Токен:", line\_tok)

# начало проверки

state = 0

for tok in line\_tok:

state = transition\_io.get(tok)[state]

if state == "K":

print("Ура! Строка разобрана. Ошибок нет.")

exit(0)

if state == "E":

print("Встречена ошибка: " + tok)

exit(0)

except Exception:

print("Упс! Что-то пошло не так. Некорректный ввод :(")

3) Составим и проведём тесты программы.

Таблица 3. Сравнение ожидаемых и полученных данных

| Исходные данные | Ожидаемый результат | Полученный результат |
| --- | --- | --- |
| scanf("%f %s", &s1, str); | Success |  |
| printf("dfgd %22d", i); | Success |  |
| fdgdfgd | Ошибка |  |

Как видно из таблицы 1, все результаты тестов совпали с ожидаемыми.

# Контрольные вопросы

**1. Дайте определение формального языка и формальной грамматики**

Формальный язык L(A) – это совокупность допустимых предложений, составленных из A, где А - некоторый алфавит. Формальная грамматика – математическая система, определяющая язык посредством порождающих правил (правила продукции).

**2. Как определяется тип грамматики по Хомскому?**

Грамматики по Хомскому различаются по набору, которые они используют.

Тип 0 использует правила вида α ->β где αV+ ;βV\*; V = VT ⋃ VN

Тип 1 использует правила вида α**X**β α->xβ, где XVN; xVТ; α, β V\*; V= VT ⋃VN.

Тип 2 использует правила вида A->β, где AVN; βV\*

Тип 3 использует правила вида A->α, A->αB или A->Bα, где A, ВVN; αVT

**3. Поясните физический смысл и обозначения формы Бэкуса–Наура.**

БНФ связывает терминальные и нетерминальные символы. Обозначения: «::=» – «можно заменить на» и « | » – «или».

**4. Что такое лексический анализ? Какие методы выполнения лексического анализа вы знаете?**

Лексический анализ - процесс преобразования исходной строки в последовательность однородных символов (токенов), которые используются на этапе синтаксического анализа для распознавания конструкций.

Выполнять лексический анализ можно с помощью последовательного сравнения строк, или при помощи конечных автоматов.

**5. Что такое синтаксический анализ? Какие методы синтаксического анализа вы знаете? К каким грамматикам применяются перечисленные вами методы?**

Синтаксический анализ - процесс распознавания конструкций языка в строке токенов.

Для регулярных грамматик возможен синтаксический анализ при помощи конечных автоматов.

Для КС-грамматик необходимо разработать автомат с магазинной памятью LL(k), LR(k).

**6. Что является результатом лексического анализа?**

Результатом лексического анализа является строка, состоящая из токенов, или ошибка именования.

**7. Что является результатом синтаксического анализа?**

Результатом синтаксического анализа является распознавание заданной конструкции или информация о наличии ошибки.

**8. В чем заключается метод рекурсивного спуска?**

Метод рекурсивного спуска заключается в распознавании аксиомы, вызова соответствующей процедуры, которая вызывает процедуру нетерминала, которые вызывают процедуру аксиомы и т.д. Это обеспечивается рекурсивным вложением в правилах продукции грамматики.

**9. Что такое таблица предшествования и для чего она строится?**

Таблица предшествования строится для сведения в удобную форму отношений предшествования терминалов с учетом приоритетов различных операций. Отношения предшествования характерны для грамматик, в которых существует однозначное отношение предшествования между соседними символами (оно позволяет определить очередную основу, т.е. момент выполнения каждой свёртки). Впоследствии на таблицах предшествования строится метод синтаксического анализа LR(k).

**10. Как с использованием таблицы предшествования осуществляют синтаксический анализ**

В начале выполняется лексический анализ, получается строка токенов. После чего строка токенов разбирается в соответствие с таблицей предшествования. Далее с применением данной таблицы по стековому методу выявляются команды или операции или конструкции.

**Вывод:** Я закрепила знания об основных методах приемов разработки лексических и синтаксических анализаторов регулярных и контекстно свободных формальных языков.