**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)**

# О т ч е т

## по домашнему заданию № 2

**Название:** Лексические и синтаксические анализаторы

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

Студент гр. ИУ6-41Б 23.05.2023 Т. Е. Старжевский
(Подпись, дата) (И.О. Фамилия)

Преподаватель 24.05.2023 С. С. Данилюк
(Подпись, дата) (И.О. Фамилия)

Москва, 2023

## Вариант 18

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

**Задание:** Разработать грамматику и распознаватель языка программирования C++ описаний функций. Предусмотреть передачу параметров только по значению, операторы присваивания, в которых используются переменные и константы целого типа.

int DD(char y,int u) { int i,j; i=y; j=u;}

## Ход работы

Опишем грамматику в форме Бэкуса-Наура:

<function> ::= <type_specifier> <identifier> '(' <parameter_list> ')' '{' <compound_statement> '}'

<type_specifier> ::= 'void' | 'int' | 'char' | 'float' | 'double' | 'bool';

<identifier> ::= ID;

<parameter_list> ::= <parameter> | <parameter> ',' <parameter_list> | e;

<parameter> ::= <type_specifier> <identifier>;

<compound_statement> ::= (<declaration> | <statement>)* | e;

<declaration> ::= <type_specifier> <identifier_list> ';';

<identifier_list> ::= <identifier> | <identifier> ',' <identifier_list> ;

<statement> ::= <assignment_statement> ';';

<assignment_statement> ::= <identifier> '=' <expression>;

<expression>: <identifier> | INT_CONST;

// Lexer rules

ID: [a-zA-Z_][a-zA-Z_0-9]*;

INT_CONST: [0-9]+;

WS: [ \t\r\n]+ -> skip; // Задаем пропуск символов

Данная грамматика является грамматикой 3 типа по Хомскому, так как не содержит вложенной рекурсии.

Синтаксическая диаграмма представлена на рисунке 1.

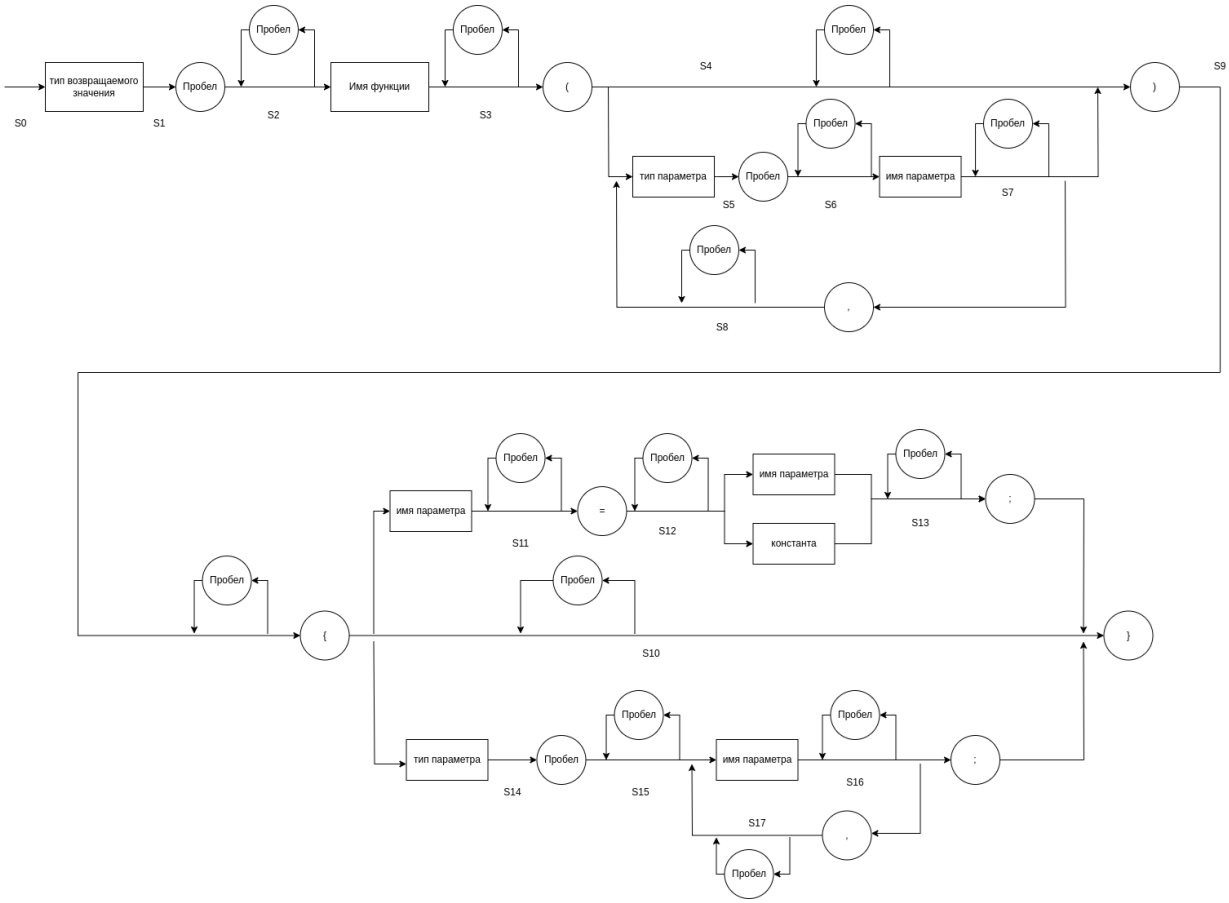


Рисунок 1 – Синтаксическая диаграмма состояний автомата.

Для разбора данной грамматики воспользуемся методом конечного автомата.

Всего символов:

1. Тип возвращаемого значения «Т. ф.»
2. Имя функции «И. ф.»
3. (
4. )
5. Тип параметра «Т. П-а»
6. Имя параметра «И. П-а»
7. ,
8. {
9. }
10. =
11. Константа «Конст»

12. ;

13. Пробел

14. Другое

Изобразим состояния автомата в виде таблицы:

Ошибку означает пустая колонка.

| Сост\Смвл | Т. ф. | И. ф. | ( | ) | Т. П-а | И. П-а | , | { | } | = | Конст. | ; | Пробел | Другое |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S0 | S1 | | | | | | | | | | | | | |
| S1 | | | | | | | | | | | | | S2 | |
| S2 | | S3 | | | | | | | | | | | S2 | |
| S3 | | | S4 | | | | | | | | | | S3 | |
| S4 | | | | S9 | S5 | | | | | | | | S4 | |
| S5 | | | | | | | | | | | | | S6 | |
| S6 | | | | | | S7 | | | | | | | S6 | |
| S7 | | | | S9 | | | S8 | | | | | | S7 | |
| S8 | | | | | S5 | | | | | | | | S8 | |
| S9 | | | | | | | | S10 | | | | | S9 | |
| S10 | | | | | S14 | S11 | | | END | | | | S10 | |
| S11 | | | | | | | | | | S12 | | | S11 | |
| S12 | | | | | | S13 | | | | | S13 | | S12 | |
| S13 | | | | | | | | | | | | S10 | S13 | |
| S14 | | | | | | | | | | | | | S15 | |
| S15 | | | | | | S16 | | | | | | | S15 | |
| S16 | | | | | | | S17 | | | | | S10 | S16 | |
| S17 | | | | | | S16 | | | | | | | S17 | |

Таблица 1 — Конечный автомат.

Текст программы

**main.py**

```
from enum import Enum
import re
import argparse
from termcolor import colored


class State(Enum):
    S0 = 0
    S1 = 1
    S2 = 2
    S3 = 3
```

```
    S4 = 4
    S5 = 5
    S6 = 6
    S7 = 7
    S8 = 8
    S9 = 9
    S10 = 10
    S11 = 11
    S12 = 12
    S13 = 13
    S14 = 14
    S15 = 15
    S16 = 16
    S17 = 17
    END = 18


class Symbol(Enum):
    VOID = 'void'
    INT = 'int'
    CHAR = 'char'
    FLOAT = 'float'
    DOUBLE = 'double'
    BOOL = 'bool'
    LPAREN = '('
    RPAREN = ')'
    COMMA = ','
    LBRACE = '{'
    RBRACE = '}'
    EQUALS = '='
    CONST = 'CONST'
    SEMICOLON = ';'
    SPACEBAR = ' '
    IDENTIFIER = 'IDENTIFIER'
    OTHER = 'OTHER'



transitions = {
    State.S0: {Symbol.VOID: State.S1, Symbol.INT: State.S1, Symbol.CHAR: State.S1, Symbol.FLOAT:
State.S1, Symbol.DOUBLE: State.S1, Symbol.BOOL: State.S1},
```

```
    State.S1: {Symbol.SPACEBAR: State.S2},
    State.S2: {Symbol.IDENTIFIER: State.S3, Symbol.SPACEBAR: State.S2},
    State.S3: {Symbol.SPACEBAR: State.S3, Symbol.LPAREN: State.S4},
    State.S4: {Symbol.RPAREN: State.S9, Symbol.SPACEBAR: State.S4, Symbol.VOID: State.S5, Symbol.INT: State.S5, Symbol.CHAR: State.S5, Symbol.FLOAT: State.S5, Symbol.DOUBLE: State.S5, Symbol.BOOL: State.S15},
    State.S5: {Symbol.SPACEBAR: State.S6},
    State.S6: {Symbol.IDENTIFIER: State.S7, Symbol.SPACEBAR: State.S6},
    State.S7: {Symbol.RPAREN: State.S9, Symbol.COMMA: State.S8, Symbol.SPACEBAR: State.S7},
    State.S8: {Symbol.SPACEBAR: State.S8, Symbol.VOID: State.S5, Symbol.INT: State.S5, Symbol.CHAR: State.S5, Symbol.FLOAT: State.S5, Symbol.DOUBLE: State.S5, Symbol.BOOL: State.S15},
    State.S9: {Symbol.LBRACE: State.S10, Symbol.SPACEBAR: State.S9},
    State.S10: {Symbol.RBRACE: State.END, Symbol.IDENTIFIER: State.S11, Symbol.SPACEBAR: State.S10, Symbol.VOID: State.S14, Symbol.INT: State.S14, Symbol.CHAR: State.S14, Symbol.FLOAT: State.S14, Symbol.DOUBLE: State.S14, Symbol.BOOL: State.S14},
    State.S11: {Symbol.EQUALS: State.S12, Symbol.SPACEBAR: State.S11},
    State.S12: {Symbol.IDENTIFIER: State.S13, Symbol.CONST: State.S13, Symbol.SPACEBAR: State.S12},
    State.S13: {Symbol.SEMICOLON: State.S10, Symbol.SPACEBAR: State.S13},
    State.S14: {Symbol.SPACEBAR: State.S15},
    State.S15: {Symbol.IDENTIFIER: State.S16, Symbol.SPACEBAR: State.S15},
    State.S16: {Symbol.COMMA: State.S17, Symbol.SEMICOLON: State.S10, Symbol.SPACEBAR: State.S16},
    State.S17: {Symbol.IDENTIFIER: State.S16, Symbol.SPACEBAR: State.S17},
}
initial_state = State.S0
final_states = {State.END}


def check_function_definition(string):
    current_state = initial_state
    current_word = ''
    for symbol in string:
        if symbol.isdigit() or symbol.isalpha() or symbol == '_':
            current_word += symbol
        else:
            if current_word:
                if re.match(r'void|int|char|float|double|bool', current_word):
                    symbol_enum = Symbol[current_word.upper()]
                elif re.match(r'[a-zA-Z_][a-zA-Z_0-9]*$', current_word):
                    symbol_enum = Symbol.IDENTIFIER
```

```
            elif re.match(r'[0-9]+$', current_word):
                symbol_enum = Symbol.CONST
            else:
                symbol_enum = Symbol.OTHER
            current_word = ''
            if current_state in transitions and symbol_enum in transitions[current_state]:
                current_state = transitions[current_state][symbol_enum]
            else:
                print(colored(f'Error: Expected symbol "{symbol_enum.value}"', 'red'))
                return False

        try:
            symbol_enum = Symbol(symbol)
        except ValueError:
            print(colored(f'Error: Invalid symbol "{symbol}"', 'red'))
            return False

        if current_state in transitions and symbol_enum in transitions[current_state]:
            current_state = transitions[current_state][symbol_enum]
        else:
            print(colored(f'Error: Expected symbol "{symbol_enum.value}"', 'red'))
            return False

    if current_state in final_states:
        print(colored('Parsed Successfully', 'green'))
        return True
    else:
        print(colored('Error while parsing', 'red'))
        return False



def main():
    parser = argparse.ArgumentParser(description='Check function definitions')
    parser.add_argument('-f', '--file', help='File name to read function definitions from')
    args = parser.parse_args()

    if args.file:
        file_path = args.file
```

```
        with open(file_path, 'r') as file:
            for line in file:
                line = line.strip()  # Удаляем лишние пробелы и символы новой строки
                check_function_definition(line)
    else:
        input_string = input('Введите определение функции: ')
        check_function_definition(input_string)

if __name__ == '__main__':
    main()
```

Таблица 1 – Тестирование программы

| Исходные данные | Ожидаемый результат | Полученный результат |
|---|---|---|
| int DD(char y,int u) { int i,j; i=y; j=u;} | parsed successfully | parsed successfully |
| int DD(char y,int u) { int i, j; i=y; j=u;} | parsed successfully | parsed successfully |
| int DD(char y,int u) { y=15; } | parsed successfully | parsed successfully |
| double sum(double a, double y) { double r,k; } | parsed successfully | parsed successfully |
| double sum( ) { } | parsed successfully | parsed successfully |
| 2double DD(float y,int u) { int l,m; int r; l=y; m=u; } | Error parsing function | Error parsing function |
| double DD(float y,int u) { int l,m int r; l=y; m=u; } | Error parsing function | Error parsing function |
| bool isNumber(char sb) { char sb2; sb2 = sb } | Error parsing function | Error parsing function |

Результаты тестирования представлены на рисунке 2.

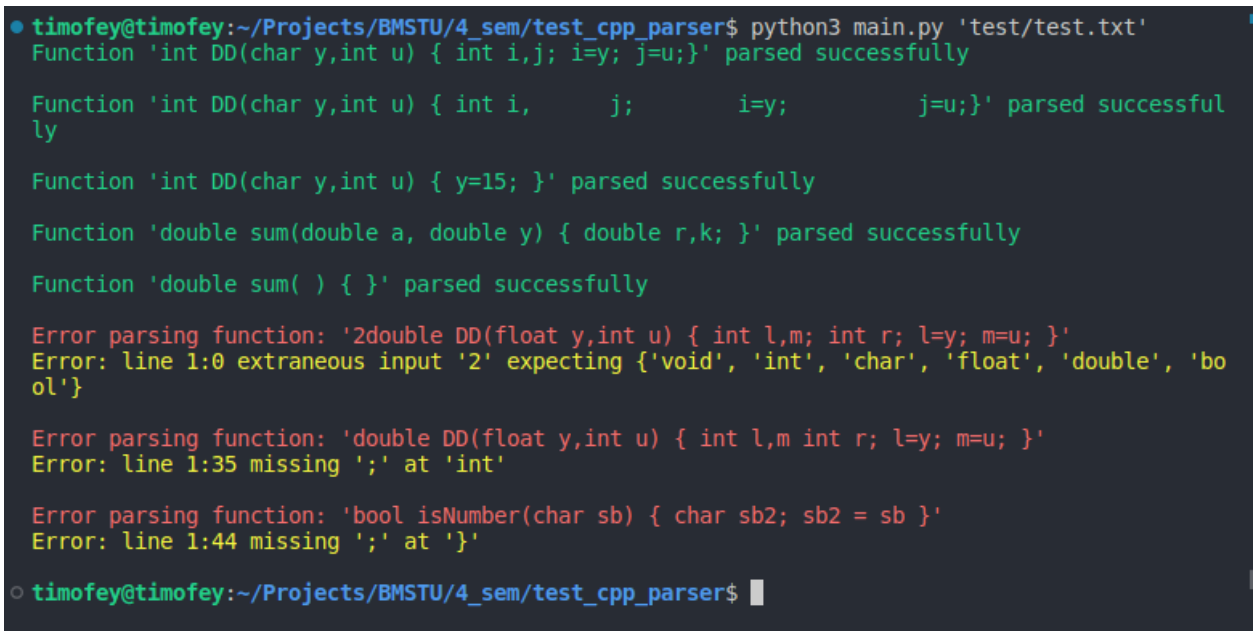


Рисунок 2 — Результаты тестирования.

Программа на данных тестах работает корректно.

## Контрольные вопросы

1. Дайте определение формального языка и формальной грамматики.

Формальная грамматика – это математическая система, определяющая язык посредством порождающих правил – правил продукции. Она определяется как четверка: G = (VT, VN, P, S), где VT – алфавит языка или множество терминальных (незаменяемых) символов; VN – множество нетерминальных (заменяемых) символов – вспомогательный алфавит, символы которого обозначают допустимые понятия языка, $VT \cap VN = \emptyset$ ; $V = VT \cup VN$ – словарь грамматики; P – множество порождающих правил – каждое правило состоит из пары строк (α, β), где $\alpha \in V+$ – левая часть правила, β V* – правая часть правила: $\alpha \in \beta$, где строка α должна содержать хотя бы один нетерминал; $S \in VN$ – начальный символ – аксиома грамматики.

Формальная грамматика, определяет правила допустимых конструкций языка.

2. Как определяется тип грамматики по Хомскому?

Тип 0 – грамматики фразовой структуры или грамматики «без ограничений»: α ->β, где $\alpha \in V+$, $\beta \in V*$– в таких грамматиках допустимо наличие любых правил вывода, что свойственно грамматикам естественных языков.

Тип 1 – контекстно-зависимые (неукорачивающие) грамматики: α -> β, где $\alpha \in V+$, $\beta \in V*$, | α | <= | β | – в этих грамматиках для правил вида α X β -> α x β возможность подстановки строки x вместо символа X определяется присутствием подстрок α и β, т. е. контекста, что также свойственно грамматикам естественных языков.

Тип 2 – контекстно-свободные грамматики: A -> β, где $A \in VN$, $\beta \in V*$ – поскольку в левой части правила стоит нетерминал, подстановки не зависят от контекста.

Тип 3 – регулярные грамматики: A -> α, A -> αB, где $A, B \in VN$, $\alpha \in VT$.

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

Форма Бэкуса-Наура (БНФ) связывает терминальные и нетерминальные символы,
используя две операции: «::=» – «можно заменить на»; «|» – «или». Главный плюс
этого - группировка правил, определяющих каждый нетерминал.

4. Что такое лексический анализ? Какие методы выполнения лексического анализа вы знаете?
Лексический анализ – это первый этап процесса компиляции текста, написанного на формальном языке программирования. Сканированием преобразуется строка в набор символов. Каждый из которых представляет из себя токен.

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. Метод рекурсивного спуска для грамматик LR(k). Стековый метод для грамматик LL(k).

6. Что является результатом лексического анализа?
Результатом лексического анализа является строка токенов, чаще всего записанных в виде таблицы.

7. Что является результатом синтаксического анализа?
Результатом, помимо распознавания заданной конструкции, является информация об ошибках в выражениях, операторах и описаниях программы.

8. В чем заключается метод рекурсивного спуска?
Метод рекурсивного спуска заключается в построении синтаксических диаграмм всех разбираемых конструкций, потом по этим диаграммам

разработать функции проверки конструкций, а затем составить основную программу, начинающую вызов функций с функции, реализующей аксиому языка.

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

Она строится во время использования грамматики с предшествованием LL(k). Когда строка разбивается на набор символов, являющиеся токенами, строится таблицы предшествования. Она нужна для того, чтобы было видно какой результат следует ожидать от той, или иной операции (когда начинается и заканчивается “основа”).

10. Как с использованием таблицы предшествования осуществляют синтаксический анализ?

Таблица использует специальные обозначении:

– ? – ошибка;

– < – начало основы;

– > – конец основы;

– () – скобки – принадлежат одной основе;

– ► – начало выражения;

– ◄ – конец выражения.

И когда идет проход по строке токенов, происходит сопоставление токенов со значениями в таблице, из чего определяется начало или конец основы или сигнал об ошибке.

**Вывод**: закрепил теоретические знания и изучил основные методы разработки лексических и синтаксических анализаторов регулярных и контекстно свободных формальных языков.