

**МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ**

**ОТЧЕТ
По лабораторной работе №1
по дисциплине «Алгоритмы и структуры данных»
Тема: Рекурсия**

Студент гр. 9381

Судаков Е.В.

Преподаватель

Фирсов М.А.

Санкт-Петербург
2020

1. Цель работы.

Ознакомиться с рекурсивными алгоритмами и реализовать рекурсивную функцию соответствующую условию.

2. Задание.

16. Построить синтаксический анализатор для понятия *скобки*.

скобки ::= $A \mid B \mid (\text{скобки} \text{ скобки})$

3. Основные теоретические положения.

Рекурсия — вызов функции (процедуры) из неё же самой непосредственно или из других функций (процедур), которые вызываются в исходной.

Синтаксический анализатор – программа, определяющая, является ли заданная последовательность символов скобками или нет, т. е. соответствует ли она определенной рекурсивной формуле.

4. Пример работы программы:

Основной тест №1:

Входные данные: ((AB)B)

Выходные данные (с промежуточной информацией):

Parser for ((AB)B) is called

| *Parser for (AB)B is called*

| | *Parser for A is called*

```

|      |      A is skobki

|      |      Parser for B is called

|      |      B is skobki

|      (AB) is skobki

|      Parser for B is called

|      B is skobki

( (AB)B ) is skobki

( (AB)B ) is skobki

```

Дополнительное тестирование :

Номер теста	Входные данные	Результат
2	A	A is skobki
3	(AB)	(AB) is skobki
4	((AB)(AB))	((AB)(AB)) is skobki
5	((A((A(AB))B))B)	((A((A(AB))B))B) is skobki
6	((((AB)(AB))((AB)(AB))))	((((AB)(AB))((AB)(AB)))) is skobki
7	((AB)(A(((AB)(AB))((AB)(AB)))))	((AB)(A(((AB)(AB))((AB)(AB))))) is skobki
8	AAA	AAA is not skobki
9	(AAA)	(AAA) is not skobki
10	(A(B))	(A(B)) is not skobki

4. Выполнение программы:

1. Для того, чтобы запустить тесты необходимо ввести 1 после вопроса программы “Хотите запустить тесты (0/1)?”.
2. Для ввода информации из файла необходимо ввести “1” на вопрос программы “Строка из консоли или из файла (0/1)?”.
3. Для ввода информации через консоль необходимо ввести “0” на вопрос программы “Строка из консоли или из файла (0/1)?”.

Рекомендуется работать только с консолью, так как она позволяет использовать цвета, что в данной работе активно используется при выводе информации.

Программе подается на вход единственная строка для разбора.

После ввода строки вызывается функция `parser(string, int&, int)`, которая управляет всем разбором : находит скобки, ищет ошибки, запускает вывод “лестницы” разбора. После завершения работы рекурсивной функции разбора, функция `main` выводит результат работы алгоритма

5. Описание функций:

Все функции описаны в исходном коде в стиле Javadoc.

`void printDepth(string s, int k, int depth)` - Функция вывода промежуточной информации.

@param s - текущая подстрока парсинга

@param depth - глубина рекурсии

@param parsed - является ли подстрока скобкой

`string parser(string s, int &k, int depth)` -Рекурсивная функция проверки

строки по правилу *скобки* ::= $A \mid B \mid (\text{скобки} \text{ скобки})$.

@param - *s* строка для проверки

@param - *k* текущее положение указателя

@param - depth глубина рекурсии

@return строка с результатом разбора PARSED/FAILURE

Задача заключается в том, чтобы в каждой паре скобок () находить по две другие скобки, если это не базовый случай, когда скобка = A | B.

void error(string s, int &k, int c) - Функция вывода причины остановки парсинга

@param s - строка для проверки

@param k - место, где возникла ошибка

@param c - код ошибки

void runTests(string testFile, string testAssertFile) - Функция управления тестированием. Поочередно запускаются тесты из файла и результат сравнивается со значением из файла ответов. Производится подсчет успешных/провальных случаев.

@param testFile - файл с тестами

@param testAssertFile - файл с корректными ответами для каждого теста

6. Описание структур данных

В данной работе для удобства вывода промежуточной информации результат парсинга оборачивается в обертку

```
struct parserResult {  
  
    string s;  
  
    string status;  
  
};
```

где string s - скобка, которую самой последней получил парсер,

string status - результат парсинга скобки

7. Идея алгоритма:

В синтаксическом разборе есть 2 возможных состояния(исключая заведомо ошибочные):

1. Базовый случай. скобки::=A | B |. По достижении этого состояния либо заканчивается разбор(если строка и есть A|B), либо парсеру передается одна из скобок в ().
2. Строка начинается с символа (. Это означает, что парсеру необходимо найти две соседних скобки. Для этого 2 раза функция **parser(string s, int &k, int depth)** вызывает сама себя. Аналогично, если и в самой parser(...) встретится символ (, тогда опять таки необходимо найти две скобки.

8. Вывод:

В ходе выполнения лабораторной работы был изучен такой вид алгоритмов, как синтаксические анализаторы. Была реализована программа, которая анализирует строку рекурсивным методом, определяя соответствие определению.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

```
#include <bits/stdc++.h>

using namespace std;

#define TEST_FILE "../Tests/testInput.txt"
#define TEST_ASSERT "../Tests/testAssertOutput.txt"

#define PARSED "OK"
#define FAILURE "ERROR"

#define RESET_COLOR    "\033[0m"
#define FAILURE_COLOR   "\033[1m\033[31m"
#define SUCCESS_COLOR   "\033[1m\033[32m"
#define INFO_COLOR      "\033[1m\033[36m"
#define CALL_COLOR      "\033[1m\033[33m"

void error(string, int, int);

/**
 * Функция вывода промежуточной информации.
 * @param s - s текущая подстрока парсинга
 * @param depth - depth глубина рекурсии
 * @param parsed - parsed является ли подстрока скобкой
 */
void printDepth(string s, int depth, bool parsed) {
    for (int i = 0; i < depth; i++) cout << "|\t"; // табуируемся по глубине рекурсии
    if (!parsed) {
```



```

        cout << CALL_COLOR << "Parser for " << INFO_COLOR << s << CALL_COLOR " is
called\n";

    } else {

        cout << INFO_COLOR << s << SUCCESS_COLOR << " is skobki\n";

    }

    cout << RESET_COLOR;

}

```

```

/**

```

```

 * Структура-обертка результата парсинга
 * Помещается строка s - самое последнее, что обнаружил парсер и
 * статус - успешен ли парсинг.
 */

```

```

struct parserResult {

    string s;

    string status;

};

```

```

/**

```

```

 * Рекурсивная функция проверки строки по правилу
 * скобки ::= A | B | ( скобки скобки ).
 * @param - s строка для проверки
 * @param - k текущее положение указателя
 * @param - depth глубина рекурсии
 * @return строка с результатом разбора PARSED/FAILURE
 */

```

```

parserResult parser(string s, int &k, int depth) {

    // Крайний случай, когда вся строка заведомо не может быть скобкой.

    if ((s[0] == 'A' || s[0] == 'B') && s.size() > 1) {

        return {"", FAILURE};
    }
}

```

```

}

// Внутри () ждем 2 скобки

string res;

if (s[k] == '(') {
    res += '(';

    string snapshot = s.substr(k, s.size() - k - depth); // следующая подстрока для
разбора
    printDepth(snapshot, depth, false);
    parserResult result = parser(s, ++k, depth + 1);
    if (result.status == PARSED) {
        res += result.s;
        result = parser(s, ++k, depth + 1);
        if (result.status == PARSED) {
            res += result.s;
            if (s[++k] == ')') && !(depth == 0 && k != s.size() - 1) {
                res += ')';
                printDepth(res, depth, true);
                return {res, PARSED};
            } else {
                error(s, k, 4);
                return {FAILURE, FAILURE};
            }
        } else {
            error(s, k, 3);
            return {FAILURE, FAILURE};
        }
    } else {
        error(s, k, 2);
        return {FAILURE, FAILURE};
    }
} else {

```

```

// Дошли до базового случая скобки единичной длины
if ((s[k] == 'A' || s[k] == 'B')) {
    res += s[k];

    printDepth(s.substr(k, 1), depth, false);
    printDepth(s.substr(k, 1), depth, true);

    return {res, PARSED};
}

}

return {FAILURE, FAILURE};
}

/**
 * Функция вывода причины останова парсинга
 * @param s - строка для проверки
 * @param k - место, где возникла ошибка
 * @param c - код ошибки
 */
void error(string s, int k, int c) {
    cout << FAILURE_COLOR;

    switch (c) {
        case 1:
            cout << "Недопустимый символ в скобке: " << s[k] << " на позиции " << k <<
"\n";

            break;

        case 2:
            cout << "НЕ скобка после ( на позиции " << k << "\n";

            break;

        case 3:
            cout << "Нет второй скобки в () на позиции " << k << "\n";

            break;

        case 4:

```

```

        cout << "Нет закрывающей ) в скобке на позиции " << k << "\n";

        break;
    }

    cout << RESET_COLOR;
}

/**
 * Функция управления тестированием.
 * Поочередно запускаются тесты из файла и результат
 * сравнивается со значением из файла ответов.
 * Производится подсчет успешных/провальных случаев.
 * @param testFile - файл с тестами
 * @param testAssertFile - файл с корректными ответами для каждого теста
 */
void runTests(string testFile, string testAssertFile) {
    cout << "-----\n";
    cout << "T E S T S\n";
    cout << "-----\n\n";
    cout << "Running tests from " << TEST_FILE << "\n";
    cout << "Assertion file from " << TEST_ASSERT << "\n\n";
    ifstream testStream, testsResStream;
    testStream.open(testFile);
    testsResStream.open(testAssertFile);
    string test, res, num;
    assert(testStream);
    assert(testsResStream);
    int passed = 0, all = 0;
    int k = 0, depth = 0;
    while (true) {
        if (testStream.eof()) break;

```

```

k = 0, depth = 0;

testStream >> num;

testStream >> test;

testsResStream >> num >> res;

parserResult parserRes = parser(test, k, depth);

cout << INFO_COLOR << num << " " << test << " " << RESET_COLOR;

if (parserRes.status == FAILURE) cout << "not ";

cout << "parsed is ";

if (res == parserRes.status) {

    cout << SUCCESS_COLOR << '[' << PARSED << ']' << "\n";

    passed++;

} else {

    cout << FAILURE_COLOR << '[' << FAILURE << ']' << "\n";

}

cout << RESET_COLOR;

all++;

}

cout << INFO_COLOR << "\nResults :\n\n";

cout << "Tests run: " << SUCCESS_COLOR << all << ", " << INFO_COLOR <<

    "Failures: " << FAILURE_COLOR << all - passed << '\n';

cout << RESET_COLOR;

}

```

```

int main() {

    int needRunTests = 0;

    cout << "Хотите запустить тесты (0/1)?\n";

    cin >> needRunTests;

    if (needRunTests == 1)

        runTests(TEST_FILE, TEST_ASSERT);

```

```

int f;

cout << "Строка из консоли или из файла (0/1)?\n";

cin >> f;

if (f == 1) {

    freopen("input.txt", "r", stdin);

    freopen("output.txt", "w", stdout);

}

cout << "Введите строку:\n";

string s;

cin >> s;

int k = 0, depth = 0;

parserResult res = parser(s, k, depth);

cout << s << (res.s != FAILURE ? " is skobki" : " is not skobki") << "\n\n";

return 0;

}

```