

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. 8304

Ястребов Д.А.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2019

Цель работы.

Изучить основы рекурсии и составления эффективных алгоритмов.

Постановка задачи.

- 1) Разработать программу, использующую рекурсию;
- 2) Сопоставить рекурсивное и итеративное решение задачи;
- 3) Сделать вывод о целесообразности и эффективности рекурсивного подхода для решения данной задачи.

Вариант 25

25 Построить синтаксический анализатор для параметризованного

понятия скобки(T), где T – заданное конечное множество, а круглые скобки «(» и «)» не являются терминальными символами, а отражают зависимость определяемого понятия от параметра T .
скобки(T)::=элемент(T) | список(скобки(T))
список(E)::=N | [ряд(E)]
ряд(E)::=элемент(E) | элемент(E) ряд(E)

Описание алгоритма.

Для решения поставленной задачи был реализован набор взаимно-рекурсивных булевых функций, структурно совпадающих с определением анализируемого объекта из условия задачи. На каждом этапе программа смотрит на имеющийся объект, в зависимости от его параметров передает его на анализ другой функции, таким образом происходит полный анализ входных данных, на основании чего в самый внешний вызов возвращается результат — true или false.

Спецификация программы.

Программа предназначена для синтаксического анализа выражения методом рекурсии.

Программа написана на языке C++. Входными данными являются две строки — строка параметров (ака алфавит) и анализируемая строка. На выход программа дает ответ — подходит ли строка под определение параметрической скобочной последовательности с заданным алфавитом.

Описание кода.

```
class BracketAnalyzer
{
public:
    BracketAnalyzer();
    ~BracketAnalyzer();

    std::vector<std::vector<char>> parameters; // множество строк, которые алфавиты
    std::vector<std::vector<char>> text; // множество строк, которые надо
    проанализировать

    void Analyze(const char* textfile, const char* parametersfile); // собсна функция-
    анализатор

private:
    std::vector<std::vector<char>> GetTextData(const char* filename); // смотрит в файл,
    достает все что можно построчно

    bool brackets(std::vector<char>::iterator, std::vector<char>::iterator);
    bool element(std::vector<char>::iterator);
    bool list(std::vector<char>::iterator, std::vector<char>::iterator);
    bool row(std::vector<char>::iterator, std::vector<char>::iterator); // 4 рекурсивных
    брата-акробата
};
```

Данный класс осуществляет функционал, требуемый в задаче. Приватные поля — это его способность доставать текст из файла и внутренние рекурсивные определения. Публичные — метод analyze, который и начинает анализ входных данных.

```
std::vector<std::vector<char>> BracketAnalyzer::GetTextData(const char * filename)
{
    std::vector<std::vector<char>> textdata;

    std::vector<char> tmp_vect;

    textdata.push_back(tmp_vect);

    std::ifstream input;

    input.open(filename); //открываем файллик

    if (!input) {
        std::cout << "Couldn't open source file";
        exit(1);
    } //не открываем файллик

    char tmp;

    while (input.get(tmp))
    {
        if (tmp!='\n')
            textdata.back().push_back(tmp);
        else {
            textdata.push_back(tmp_vect); // строка кончилась - заводим новую
        }
    }
}
```

```

        input.close();

        return textdata; // возвращаем вектор char-векторов
    }

```

Эта функция отвечает за забор данных из source-файла и возвращение их в виде вектора из символьных векторов (массив строк). Допущение : строки разделяются символом „\n”.

```

void BracketAnalyzer::Analyze(const char* textfile, const char* parametersfile)
{
    text = BracketAnalyzer::GetTextData(textfile);
    parameters = BracketAnalyzer::GetTextData(parametersfile); //Начинаем движение.
    Сначала достали анализируемые строки, потом достали параметры

    while (text.size()) { // пока есть что анализировать

        std::vector<char>::iterator left = text.back().begin();
        std::vector<char>::iterator right = text.back().end(); right--; //всем
        дальнейшим функциям передаем строку как два итератора

        // - один указывает на начало, второй на конец

        if (brackets(left, right)) {
            std::cout << "This is a correct Bracket(T) sequence" << std::endl;
            //самый внешний вызов
        }
        else {
            std::cout << "This is NOT a correct Bracket(T) sequence" << std::endl;
            // -//-
        }

        text.pop_back(); //проанализировали - убрали, идем дальше
        parameters.pop_back();
    }
}

```

Это основной метод класса — он итеративно анализирует все входные данные, пока они не кончатся.

```

bool BracketAnalyzer::brackets(std::vector<char>::iterator left, std::vector<char>::iterator
right)
{
    return (element(left) && (left == right)) || list(left, right);
}

bool BracketAnalyzer::element(std::vector<char>::iterator left)
{
    return (std::find(parameters.back().begin(), parameters.back().end(), *left) !=
parameters.back().end());
}

bool BracketAnalyzer::list(std::vector<char>::iterator left, std::vector<char>::iterator
right)
{
    return ((*left == 'N') && (left == right)) || ((*left == '[') && (*right == ']') &&
row(left + 1, right - 1));
}
//три функции выше совсем дословно повторяет условие задачи - взаимнорекурсивные определения

```

```

bool BracketAnalyzer::row(std::vector<char>::iterator left, std::vector<char>::iterator
right)
{
    bool result = true;

    while (left != right + 1) {
        if (*left != '[') {
            result = result && brackets(left, left);
            left++;
        }

        else {
            auto tmp_iter = left + 1;
            int cnt = 1;

            while (tmp_iter != right + 1) {
                if (*tmp_iter == '[')
                    cnt++;
                if (*tmp_iter == ']')
                    cnt--;

                if (!cnt) {
                    result = result && row(left + 1, tmp_iter - 1);
                    left = tmp_iter + 1;
                    break;
                }

                tmp_iter++;
            }

            if (tmp_iter == right + 1)
                return false;
        }
    }
    // если коротко - я знаю, что для меня true_объект это набор разрешенных символов или
true_объект в квадратных скобках
    // поэтому я иду итеративно вправо по строке, смотря на каждый символ, пока не
уткнусь в '[' . В этот момент ищу ей пару и отправляю
    // на анализ содержимое, если пара нашлась
    return result;
}

```

Эти взаимно-рекурсивно определяемые функции соответствуют описанию объекта Скобки(Т) из условия задачи. Последнее определение имеет вид «одно или несколько старших определений», то есть происходит логическое заикливание. Для его разрешения использовано то, что известно, какое допустимое множество видов может принимать строка, на основании чего она делится на части, каждая из которых продолжает анализироваться дальше вглубь. Таким образом, происходит спуск и полный анализ входных данных.

Вывод.

Был получен опыт работы с рекурсией и с построением синтаксического анализатора. На мой взгляд, итеративное решение поставленной задачи более эффективно, так как требует меньше

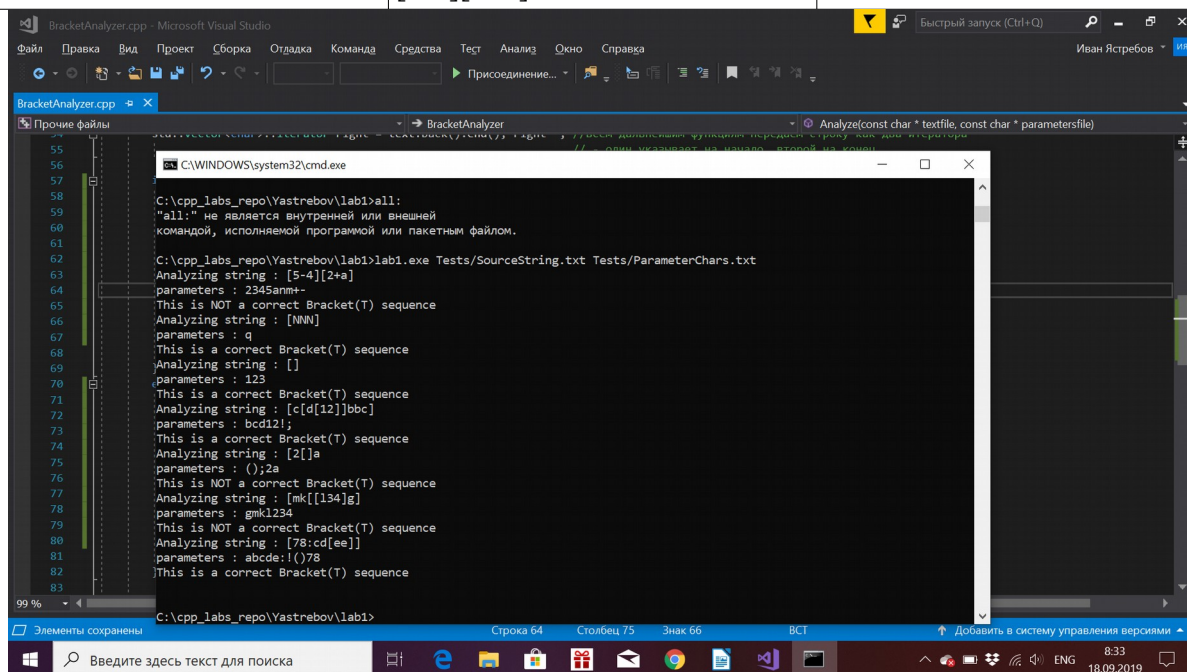
памяти, а неизбежная итеративность в рекурсивном алгоритме требует почти столько же времени, сколько брутфорсный итеративный обход слева направо.

ПРИЛОЖЕНИЕ

1) ТЕСТИРОВАНИЕ:

Работа программы, входные даны в таблице

abcde:>()78	[78:cd[ee]]	+
gmk1234	[mk[[134]g]	-
()2a	[2[]a	-
bcd12!;	[c[d[12]]bbc]	+
123	[]	+
q	[NNN]	+
2345anm+-	[5-4][2+a]	-



```
BracketAnalyzer.cpp - Microsoft Visual Studio
File Edit View Project Build Tools Team Explorer Output Console
C:\WINDOWS\system32\cmd.exe
C:\cpp_labs_repo\Yastrebov\lab1>all:
"all:" не является внутренней или внешней
командой, исполняемой программой или пакетным файлом.
C:\cpp_labs_repo\Yastrebov\lab1>lab1.exe Tests/SourceString.txt Tests/ParameterChars.txt
Analyzing string : [5-4][2+a]
parameters : 2345anm+-
This is NOT a correct Bracket(T) sequence
Analyzing string : [NNN]
parameters : q
This is a correct Bracket(T) sequence
Analyzing string : []
parameters : 123
This is a correct Bracket(T) sequence
Analyzing string : [c[d[12]]bbc]
parameters : bcd12!;
This is a correct Bracket(T) sequence
Analyzing string : [2[]a
parameters : ();2a
This is NOT a correct Bracket(T) sequence
Analyzing string : [mk[[134]g]
parameters : gmk1234
This is NOT a correct Bracket(T) sequence
Analyzing string : [78:cd[ee]]
parameters : abcde:>()78
This is a correct Bracket(T) sequence
```

2) ИСХОДНЫЙ КОД:

Analyze.cpp :

```
#include "BracketAnalyzer.h"
```

```
int main(int argc, char* argv[]) // пользуемся аргументами cmd для тестирования
{
    BracketAnalyzer analyzer; // создаем анализатор

    if (argc == 3)
        analyzer.Analyze(argv[1], argv[2]);
    else
```

```

        analyzer.Analyze("SourceString.txt", "ParameterChars.txt"); // если не
передали конкретные тестовые файлы, ищет дефолтные

        std::getchar();

        return 0;
}

```

BracketAnalyzer.h :

```

#pragma once

#include <vector>
#include <fstream>
#include <iostream>
#include <algorithm>

class BracketAnalyzer
{
public:
    BracketAnalyzer();
    ~BracketAnalyzer();

    std::vector<std::vector<char>> parameters; // множество строк, которые алфавиты
    std::vector<std::vector<char>> text; // множество строк, которые надо
проанализировать

    void Analyze(const char* textfile, const char* parametersfile); // собсна функция-
анализатор

private:
    std::vector<std::vector<char>> GetTextData(const char* filename); // смотрит в файл,
достаёт все что можно построчно

    bool brackets(std::vector<char>::iterator, std::vector<char>::iterator);
    bool element(std::vector<char>::iterator);
    bool list(std::vector<char>::iterator, std::vector<char>::iterator);
    bool row(std::vector<char>::iterator, std::vector<char>::iterator); // 4 рекурсивных
брата-акробата
};

```

BracketAnalyzer.cpp :

```

#include "BracketAnalyzer.h"

BracketAnalyzer::BracketAnalyzer()
{
}

BracketAnalyzer::~BracketAnalyzer()
{
}

std::vector<std::vector<char>> BracketAnalyzer::GetTextData(const char * filename)
{
    std::vector<std::vector<char>> textdata;

    std::vector<char> tmp_vect;

    textdata.push_back(tmp_vect);
}

```



```

std::ifstream input;

input.open(filename); //открываем файл

if (!input) {
    std::cout << "Couldn't open source file";
    exit(1);
} //не открываем файл

char tmp;

while (input.get(tmp))
{
    if (tmp!='\n')
        textdata.back().push_back(tmp);
    else {
        textdata.push_back(tmp_vect); // строка кончилась - заводим новую
    }
}
input.close();

return textdata; // возвращаем вектор char-векторов
}

void BracketAnalyzer::Analyze(const char* textfile, const char* parametersfile)
{
    text = BracketAnalyzer::GetTextData(textfile);
    parameters = BracketAnalyzer::GetTextData(parametersfile); //Начинаем движение.
    Сначала достали анализируемые строки, потом достали параметры

    while (text.size()) { // пока есть что анализировать

        std::vector<char>::iterator left = text.back().begin();
        std::vector<char>::iterator right = text.back().end(); right--; //всем
        дальнейшим функциям передаем строку как два итератора

        // - один указывает на начало, второй на конец

        if (brackets(left, right)) {
            std::cout << "This is a correct Bracket(T) sequence" << std::endl;
            //самый внешний вызов
        }
        else {
            std::cout << "This is NOT a correct Bracket(T) sequence" << std::endl;
            // -//-
        }

        text.pop_back(); //проанализировали - убрали, идем дальше
        parameters.pop_back();
    }
}

bool BracketAnalyzer::brackets(std::vector<char>::iterator left, std::vector<char>::iterator
right)
{
    return (element(left) && (left == right)) || list(left, right);
}

bool BracketAnalyzer::element(std::vector<char>::iterator left)
{
    return (std::find(parameters.back().begin(), parameters.back().end(), *left) !=
parameters.back().end());
}

```

```

}

bool BracketAnalyzer::list(std::vector<char>::iterator left, std::vector<char>::iterator
right)
{
    return ((*left == 'N') && (left == right)) || ((*left == '[') && (*right == ']') &&
row(left + 1, right - 1));
}
//три функции выше совсем дословно повторяет условие задачи - взаиморекурсивные определения
bool BracketAnalyzer::row(std::vector<char>::iterator left, std::vector<char>::iterator
right)
{
    bool result = true;

    while (left != right + 1) {
        if (*left != '[') {
            result = result && brackets(left, left);
            left++;
        }

        else {
            auto tmp_iter = left + 1;
            int cnt = 1;

            while (tmp_iter != right + 1) {
                if (*tmp_iter == '[')
                    cnt++;
                if (*tmp_iter == ']')
                    cnt--;

                if (!cnt) {
                    result = result && row(left + 1, tmp_iter - 1);
                    left = tmp_iter + 1;
                    break;
                }

                tmp_iter++;
            }

            if (tmp_iter == right + 1)
                return false;
        }
    }

    // если коротко - я знаю, что для меня true_объект это набор разрешенных символов или
true_объект в квадратных скобках
    // поэтому я иду итеративно вправо по строке, смотря на каждый символ, пока не
уткнусь в '['. В этот момент ищу ей пару и отправляю
    // на анализ содержимое, если пара нашлась
    return result;
}

```