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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Алгоритмы и структуры данных»

Тема: «Рекурсия»

Студент гр. 6381 Минуллин М. А.

Преподаватель Фирсов М. А.

Санкт-Петербург

2018

Задание:

14. Построить синтаксический анализатор для понятия скобки:

скобки ::= A | (B <скобки> <скобки>)

Пояснение задания:

На вход программе подаётся строка. Задача состоит в том, чтобы определить, удовлетворяет ли она определённому понятию «скобки». Реализовать рекурсивную функцию проверки.

Описание алгоритма:

Из входных данных удаляются все пробельные символы. Затем вызывается рекурсивная функция проверки входной информации.

Описание функций:

```
void print_tabs(size_t tabs_count);
```
- функция используется для создания отступа в консоли символами табуляции.

```
void error_handler(int error_code, size_t tabs_count);
```
- функция-обработчик ошибок, возникших при анализе входных данных. Используется для вывода сообщений об ошибках во входных данных.

```
bool brackets(const std::string &str, size_t &index, size_t tabs_count);
```
- рекурсивная функция, проверяющая, является ли строка входных данных понятием скобки (возможно ложноположительное срабатывание). Входные данные передаются по ссылке, чтобы избежать многократного копирования. Обрабатываемый индекс так же передаётся по ссылке, чтобы его можно было увеличивать рекурсивно во внутренних вызовах функции.

```
bool lexical_analyzer(std::string str);
```
- функция, удаляющая лишние пробелы из входных данных, обрабатывающая возможное ложноположительное срабатывание предыдущей функции.

Тестирование:

На вход программе подаётся строка, на выход информация о проверки со всеми рекурсивными вызовами и краткий результат: success, если строка удовлетворяет условию и failure в противном случае. Для автоматизации тестирования, был написан bash-скрипт.

Входные данные	Результат
(B(B(B(BAA)A)A)A)A	lexical analyser: input: (B(B(B(BAA)A)A)A)A call [index: 0] call [index: 2] call [index: 4] call [index: 6] call [index: 8] llac call [index: 9] llac llac

	call [index: 11] llac llac call [index: 13] llac llac call [index: 15] llac llac error: excess symbols result: failure
(B(BAA)(BAA))	lexical analyser: input: (B(BAA)(BAA)) call [index: 0] call [index: 2] call [index: 4] llac call [index: 5] llac llac call [index: 7] call [index: 9] llac call [index: 10] llac llac llac result: success

Вывод:

В процессе выполнения лабораторной работы были поли получены навыки по написанию рекурсивных функции, bash-скриптов и автоматизации тестирования. Работа была написана на C++.

Приложение А. Код main.cpp

```
#include <iostream>
#include <string>
#include <algorithm>

void print_tabs(size_t tabs_count) {
    for (size_t i = 0; i < tabs_count; ++i)
        std::cout << '\t';
}

void error_handler(int error_code, size_t tabs_count) {
    if (error_code != 0) {
        print_tabs(tabs_count);
        std::cout << "error: ";
    }
```

```
    switch (error_code) {
    case 1:
        std::cout << "'A' or '(' expected" << std::endl;
        break;
    case 2:
        std::cout << "'B' expected" << std::endl;
        break;
    case 3:
        std::cout << "')' expected" << std::endl;
        break;
    case 4:
        std::cout << "excess symbols" << std::endl;
        break;
    }
}

bool brackets(const std::string &str, size_t &index, size_t tabs_count) {
    print_tabs(tabs_count);
    std::cout << "call [index: " << index << "]" << std::endl;

    bool result = true;

    if (str[index] == 'A') {
        ++index;
        result = true;
    }
    else if (str[index] == '(') {
        ++index;
        if (str[index] == 'B') {
            ++index;
            bool brackets1 = brackets(str, index, tabs_count + 1);
            bool brackets2 = brackets(str, index, tabs_count + 1);
            result = brackets1 && brackets2;
            if (str[index] == ')')
                ++index;
            else {
                result = false;
                error_handler(3, tabs_count);
            }
        }
        else {
            result = false;
            error_handler(2, tabs_count);
        }
    } else {
        result = false;
        error_handler(1, tabs_count);
    }

    print_tabs(tabs_count);
```

```
    std::cout << "llac" << std::endl;

    return result;
}

bool lexical_analyzer(std::string str) {
    str.erase(std::remove(str.begin(), str.end(), ' '), str.end());

    size_t index = 0;
    bool result = brackets(str, index, 0);

    if (result && str.length() != index) {
        result = false;
        error_handler(4, 0);
    }

    return result;
}

int main() {

    std::string str;
    std::cin >> str;

    std::cout << "lexical analyser: " << std::endl;
    std::cout << "input: " << str << std::endl;
    bool result = lexical_analyzer(str);

    std::cout << "result: ";
    if (result)
        std::cout << "success";
    else
        std::cout << "failure";
    std::cout << std::endl << std::endl;

    return 0;
}
```

Приложение Б. Файл Makefile:

```
CODE    = ./Source/
OBJ     = main.o
EXE     = lexical_analyzer
CXX     = g++
CFLAGS  = -Wall -Wextra -c

all: $(OBJ)
    $(CXX) $(OBJ) -o $(EXE)

main.o: $(CODE)main.cpp
    $(CXX) $(CFLAGS) $(CODE)main.cpp
```

```
clean:
	rm $(OBJ) $(EXE)
```

Приложение В. Файл checker.sh:

```
#!/bin/bash
make
if [ -f "result.txt" ]; then
    rm result.txt
fi
touch result.txt
for i in $(ls Tests/Correct); do
    echo "running test: \"Tests/Correct/$i\" ";
    sleep 0.2s;
    echo "correct brackets test: \"$i\"" >>result.txt;
    ./lexical_analyzer <Tests/Correct/$i >>result.txt;
done;
for i in $(ls Tests/Incorrect); do
    echo "running test: \"Tests/Incorrect/$i\" ";
    sleep 0.2s;
    echo "incorrect brackets test: \"$i\"" >>result.txt;
    ./lexical_analyzer <Tests/Incorrect/$i >>result.txt;
done;
make clean
```