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. 9303 _____ Махаличев Н.А.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с понятием «рекурсия», её приёмах и методах. Реализовать программу-анализатор для понятия скобки с помощью метода рекурсии на языке программирования C++.

Задание.

Вариант 13.

Построить синтаксический анализатор для понятия скобки.

скобки::=A | *скобка скобки*

скобка::=(B *скобки*)

Основные теоретические положения.

Рекурсия — определение, описание, изображение какого-либо объекта или процесса внутри самого этого объекта или процесса, то есть ситуация, когда объект является частью самого себя.

В программировании рекурсия — вызов функции (процедуры) из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия). Количество вложенных вызовов функции или процедуры называется глубиной рекурсии. Рекурсивная программа позволяет описать повторяющееся или даже потенциально бесконечное вычисление, причём без явных повторений частей программы и использования циклов.

Рекурсивные функции используют так называемый «Стек вызовов». Когда программа вызывает функцию, функция отправляется на верх стека вызовов. Адрес возврата и локальные переменные функции записываются в стек, благодаря чему каждый следующий рекурсивный вызов этой функции пользуется своим набором локальных переменных и за счёт этого работает корректно. Оборотной стороной этого механизма является то, что при чрезмерно большой глубине рекурсии может наступить переполнение стека вызовов.

Выполнение работы.

Создан класс Workspace, в котором выбирается метод ввода данных (с помощью консоли, или из файла). В зависимости от выбора пользователя вызывается метод void Console() или void File(string inputFile, bool showDepth), в которых происходит построчное считывание данных из консоли и из файла соответственно.

После считывания строки вызывается метод bool Run(string line, int position, int depth, bool showDepth) класса Recursion, в который была передана строка, отформатированная методом string ClearSymbols(string line) класса Workspace, удаляющим из строки все пробелы и символы табуляции. Метод Run() рекурсивно проверяет, подходит ли введённая строка под описание скобок или нет, а также каждый раз выводит глубину рекурсии. Если строка является скобками, Run() возвращает true, иначе false. Полученный результат записывается в файл output.txt.

Для проверки работоспособности программы создан класс Test. Принцип его работы следующий:

1) В директории ./Tests/ToCheck находятся файлы, в которых записаны строки, которые должна обработать программа, а в директории ./Tests/Answers находятся файлы с написанными в них ожидаемыми результатами работы программы. Стоит заметить, что название файла, содержащего строки для проверки, должно совпадать с названием файла, в котором записаны ожидаемые результаты работы программы.
2) С помощью цикла while каждый файл директории ./Tests/ToCheck открывается в программе, из него считываются данные и выполняется программа.
3) Выходные данные программы для каждого файла построчно сравниваются с данными файла с таким же названием в директории ./Tests/Answers. Если строки являются одинаковыми, программа выводит в консоль надпись «Correct answer» и строку, которая была проверена, иначе выводит «Not correct answer» и сравниваемые строки.

Выводы.

В процессе выполнения лабораторной работы было изучено понятие рекурсия, её приёмы и особенности.

Была разработана программа, являющаяся синтаксическим анализатором для понятия скобки. Для тестирования программы был разработан класс Test, проверяющий корректность выходных данных программы с правильным ответом.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp

```
#include "../Headers/Workspace.h"
#include "../Headers/Test.h"

int main(int argc, char **argv){
    string arg;
    if (argc == 2){
        arg = argv[1];
    }
    if (arg == "test"){
        Test::Run();
    } else {
        Workspace a;
        a.ChooseInput();
    }
    return 0;
}
```

Файл Workspace.h

```
#ifndef WORKSPACE_H
#define WORKSPACE_H

#define RED "\033[1;31m"
#define ORANGE "\033[1;33m"
#define GREEN "\033[1;32m"
#define RESET "\033[0m"

#include <iostream>
#include <string>
#include <fstream>
#include "../Headers/Recursion.h"

using namespace std;

class Workspace{
public:
    Workspace();
    void ChooseInput();
    static string ClearSymbols(string line);
    static string Output(string line, bool isBrackets);
    void Console();
    void File(string inputFile, bool deepon = true);
    string line;
    string inputFile;
    Recursion recursion;
    ifstream input;
    ofstream output;
    bool isBrackets;
};

#endif
```

Файл Workspace.cpp

```
#include "../Headers/Workspace.h"

Workspace::Workspace(){}

void Workspace::ChooseInput(){
    Workspace method;
    cout << "Please select a data entry method" << endl;
    cout << "Press \"1\" if the console" << endl;
    cout << "Press \"2\" if the file" << endl;
    string choice;
    getline(cin, choice);
    switch (choice[0]) {
        case '1':
            method.Console();
            break;
        case '2':
            cout << "Please enter filename: ";
            getline(cin, inputFile);
            method.File(inputFile);
            break;
        default:
            cout << "The entered digit does not match the input options"
<< endl;
            break;
    }
}

void Workspace::Console(){
    cout << "Enter the line: ";
    getline(cin, line);
    cout << Workspace::Output(line,
recursion.Run(Workspace::ClearSymbols(line))) << endl;
}

void Workspace::File(string inputFile, bool showDepth){
    input.open(inputFile);
    output.open("output.txt");
    if (input.is_open() && output.is_open()){
        while (getline(input, line)){
            cout << "Checking line: " << line << endl;
            output << Workspace::Output(line, isBrackets =
recursion.Run(Workspace::ClearSymbols(line), 0, 0, showDepth)) << endl;
            cout << Workspace::Output(line, isBrackets) << endl;
        }
        cout << "The program has been completed. The result is written to
the file \"output.txt\"" << endl;
    } else {
        cout << "Can't open file" << endl;
    }
    input.close();
    output.close();
}

string Workspace::ClearSymbols(string line){
    for (int i = 0; i < line.length(); i++){
        if ((line[i] == ' ') || (line[i] == '\t')) {
```

```
            line.erase(i, 1);
            i--;
        }
    }
    return line;
}

string Workspace::Output(string line, bool isBrackets){
    if (isBrackets){
        return line + " - THIS IS A BRACKETS";
    } else {
        return line + " - THIS IS NOT A BRACKETS";
    }
}
```

Файл Recursion.h

```
#ifndef RECURSION_H
#define RECURSION_H

#include <string>
#include <iostream>

using namespace std;

class Recursion{
public:
    Recursion();
    bool Run(string line, int position = 0, int depth = 0, bool showDepth
= true);
private:
    bool isBrackets;
};

#endif
```

Файл Recursion.cpp

```
#include "../Headers/Recursion.h"

Recursion::Recursion(){}

bool Recursion::Run(string line, int position, int depth, bool
showDepth){
    isBrackets = false;
    if (showDepth) cout << string(depth,' ') << "Depth " << depth << " -
joining" << endl;
    switch(line[position]){
        case 'A':
            if (line.length() == 1) {
                isBrackets = true;
                break;
            }
            if (line[position + 1] == '\0') {
                isBrackets = true;
                break;
```

```
            }
            if (line[position + 1] != '(' && line[position + 1] != 'B' &&
line[position + 1] != 'A') {
                isBrackets = Recursion::Run(line, position + 1, depth +
1, showDepth);
            }
            break;
        case '(':
            if (line[position+1] == 'B'){
                isBrackets = Recursion::Run(line, position + 1, depth +
1, showDepth);
            }
            break;
        case 'B':
            if (line[position + 1] != 'B'){
                isBrackets = Recursion::Run(line, position + 1, depth +
1, showDepth);
            }
            break;
        case ')':
            if (line[position + 1] != 'B'){
                isBrackets = Recursion::Run(line, position + 1, depth +
1, showDepth);
            }
            break;
        default:
            isBrackets = false;
            break;
    }
    if (showDepth) cout << string(depth,' ') << "Depth " << depth << " -
exiting" << endl;
    return isBrackets;
}
```

Файл Test.h

```
#ifndef TESH_H
#define TESH_H

#include <iostream>
#include <dirent.h>
#include <string>
#include "Workspace.h"

using namespace std;

class Test{
public:
    Test();
    static void Run();
    void File(char *name);
    void Compare(string inputFile);
    string lineOutput;
    string lineAnswers;
    string inputFile;
    string directory;
    ifstream inputFromOutput;
```

```
    ifstream inputFromAnswers;
    Workspace workspace;
};

#endif
```

Файл Test.cpp

```
#include "../Headers/Test.h"

Test::Test(){}

void Test::Run(){
    Test test;
    DIR *myDir = opendir("./Tests/ToCheck");
    if(myDir == NULL) {
        perror("Unable to open directory ./Tests/ToCheck");
        return;
    }
    struct dirent *entry;
    while ((entry = readdir(myDir)) != nullptr){
        test.File(entry->d_name);
    }
    closedir(myDir);
}

void Test::File(char *name){
    inputFile = name;
    if ((inputFile != ".") && (inputFile != "..")){
        directory = "./Tests/ToCheck/";
        cout << ORANGE "Comparing file " RESET << inputFile << endl;
        workspace.File(directory + inputFile, false);
        Test::Compare(inputFile);
    }
}

void Test::Compare(string inputFile){
    inputFromOutput.open("output.txt");
    inputFromAnswers.open("./Tests/Answers/" + inputFile);
    if (inputFromOutput.is_open() && inputFromAnswers.is_open()){
        while (getline(inputFromOutput, lineOutput) &&
getline(inputFromAnswers, lineAnswers)){
            if (lineOutput == lineAnswers){
                cout << GREEN "Correct answer - " RESET << lineAnswers <<
endl;
            } else {
                cout << RED "Not correct answer - " RESET << lineAnswers
<< RED "\n - Program response - " RESET << lineOutput << endl;
            }
        }
    }
    inputFromOutput.close();
    inputFromAnswers.close();
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Таблица Б.1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1	A	A - THIS IS A BRACKETS	Получен ожидаемый вывод.
2	A A	A A - THIS IS NOT A BRACKETS	Получен ожидаемый вывод (несоответствие условию скобки).
3	(B)	(B) - THIS IS NOT A BRACKETS	Получен ожидаемый вывод (несоответствие условию скобки).
4	A	A - THIS IS A BRACKETS	Получен ожидаемый вывод (проверка на пропуск пробелов).
5	(BA)	(BA) - THIS IS NOT A BRACKETS	Получен ожидаемый вывод (несоответствие условию скобки).
6	(BA) A	(BA) A - THIS IS A BRACKETS	Получен ожидаемый вывод.
7	(BA) (BA A	(BA) (BA A - THIS IS NOT A BRACKETS	Получен ожидаемый вывод (отсутствие закрывающей скобки).
8	()	() - THIS IS NOT A BRACKETS	Получен ожидаемый вывод (несоответствие условию скобки).
9	(B(B (B (B (B A) A) (B (B (B A) A) A) A) A) A	(B(B (B (B (B A) A) (B (B (B A) A) A) A) A) A) A - THIS IS A BRACKETS	Получен ожидаемый вывод (проверка на глубокую рекурсию).
10	(B A (B)) A	(B A (B)) A - THIS IS NOT A BRACKETS	Получен ожидаемый вывод (несоответствие условию скобки).
11	(B (B A) Av) A	(B (B A) Av) A - THIS IS NOT A BRACKETS	Получен ожидаемый вывод (лишние символы).
12	Пустая строка	- THIS IS NOT A BRACKETS	Получен ожидаемый вывод (проверка пустой строки).