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: Взаимно-рекурсивные функции и процедуры. Синтаксический анализатор понятия скобки

Студент гр. 9384 _____ Гурин С.Н.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

Разработать анализатор, проводящий синтаксический анализ строки с помощью взаимно-рекурсивных функций и процедур.

Задание.

ВАРИАНТ 6.

Построить синтаксический анализатор для понятия *простое выражение*.

простое_выражение::=простой_индефикатор|(простое_выражение знак_операции простое_выражение)

простой_идентификатор::= буква

знак_операции::= -|+|*

Анализ задачи.

Задача заключается в том, что нужно написать алгоритм, проверяющий правильность синтаксиса в веденном простом выражении. Его реализация происходит с помощью рекурсий.

Реализация.

Была реализована программа, на вход которой подаются данные из файла input.txt, введенные пользователем. Программа проводит синтаксический анализ каждой строки из данного файла и создает файл output.txt, в котором записаны результаты проведенной работы. Так же результат проведенной работы выводится в консоль/терминал пользователя.

Проверка на синтаксис происходит с помощью рекурсивной функции `bool Check(string sentence){…}`. На вход данной функции подается строка, которую нужно анализировать. Далее происходит проверка является ли эта строка простым идентификатор или нет, если является, то функция возвращает значение true, если нет, то функция проверяет синтаксис простого выражения, проверяя первый и последний символ на наличие открытой и закрытой скобки. Затем происходит проверка на наличие простых выражений в первом простом

выражении, но перед этим происходит выполнение процедуры проверки на наличие соблюденной орфографии во всей строке.

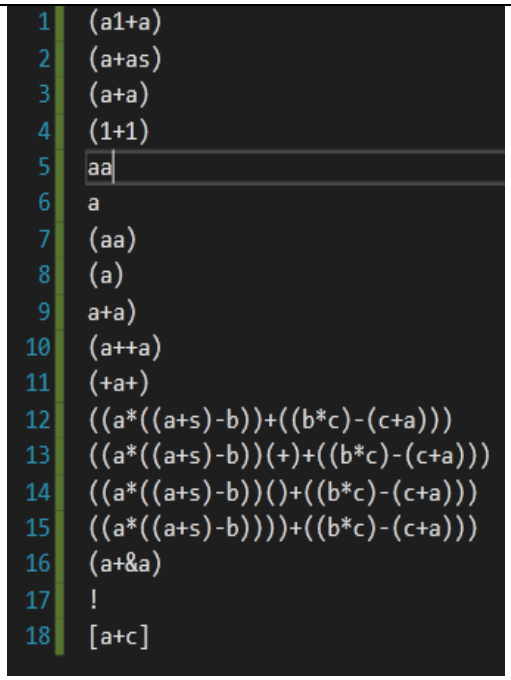
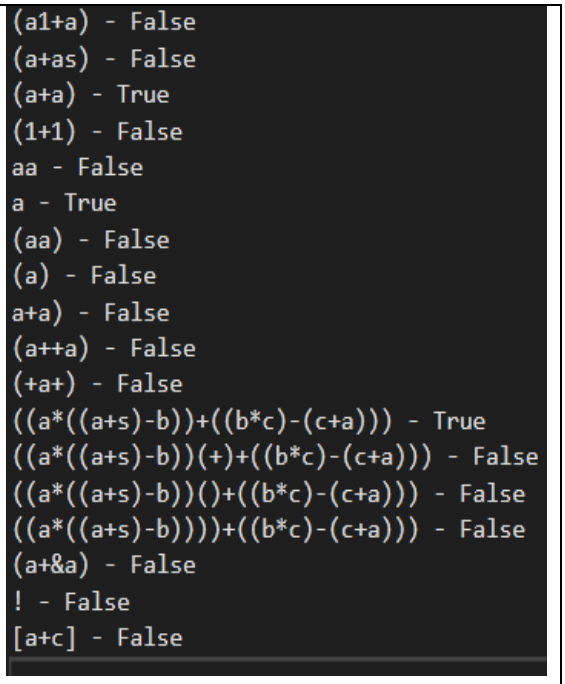
Проверка на орфографический синтаксис всей строки происходит с помощью функции `Check_sin()`. Она проверяет является ли первый и последующий символы буквами, если являются, то функция возвращает `false`. Так же эта функция проверяет является ли символ буквой, скобкой или знаком. Если нет, функция возвращает `false`. В другом случае функция возвращает `true`.

Если второй символ строки – “(“, то есть имеется наличие первого простого выражения в первом простом выражении, то происходит подсчет количества знаков, открытых и закрытых скобок. Так же происходит проверка на синтаксис скобок и знаков. После цикла подсчета количество закрытых и открытых скобок должно совпадать. Этот цикл проходит до конца первого главного простого выражение. Далее происходит считывание этого простого выражение в переменную `string a`. Затем происходит считывание второго простого выражение в переменную `string b`. После записи простых выражений в данные переменные происходит рекурсивный вызов функции `Check(a)`, `Check(b)`, благодаря которым происходит повторное выполнение алгоритма с выражениями в `a` и в `b`.

После выполнения алгоритма проверки синтаксиса строки, происходит запись результатов этой проверки в ранее созданный файл output.txt.

Тестирование.

№ теста	input.txt	output.txt

1		

Выводы.

При выполнении данной лабораторной работы была реализована программа, которая считывает строки из отдельного файла, проверяет их на синтаксис, а затем записывает результат в созданный файл. Так же был изучен принцип работы рекурсивных функций, принцип работы со строкой и принцип работы с файлами на языке C++.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

bool Check_sin(string sentence, int n) {
  //  char signs[20] =
{'!','@','#','$','%','^','&','{','}','[',']','=','?','>','<',',','.','"',
'`','~'};
    for (int i = 0; i < n - 2;i++) {
        if (isalpha(sentence[i]) && isalpha(sentence[i + 1])) {
            return false;
        }
        if (!isalpha(sentence[i]) && sentence[i] != '(' && sentence[i] !=
')' && sentence[i] !='+' && sentence[i] != '*' && sentence[i] != '-') {
            return false;
        }

    /*    if (isdigit(sentence[i])) {
            return false;
        }
        for (int j = 0; j < 20; j++) {
            if (sentence[i] == signs[j]) {
                return false;
            }*/
    }
    return true;
}

bool Check(string sentence) {
    int n = sentence.length();
    string a = "", b = "";
    if (isalpha(sentence[0]) && n == 1) {
        return true;
```

```
    }
    else {
        if (sentence[0] == '(' && sentence[n - 1] == ')') {
            int count = 1;
            int count_of_open = 0;
            int count_of_close = 0;
            int signs = 0;
            if (!Check_sin(sentence, n)) {
                return false;
            }
            if (sentence[1] == '(') {
                while (signs <= count_of_open && count != n-1) {
                    if (sentence[count] == '(') {
                        count_of_open++;
                    }
                    if (sentence[count] == ')') {
                        count_of_close++;
                    }
                    if (sentence[count] == '+' || sentence[count] == '-'
|| sentence[count] == '*') {
                        signs++;
                    }
                    if (sentence[count+1] == ')' && (sentence[count] ==
'+' || sentence[count] == '-' || sentence[count] == '*')) {
                        return false;
                    }
                    if (sentence[count - 1] == '(' && (sentence[count] ==
'+' || sentence[count] == '-' || sentence[count] == '*')) {
                        return false;
                    }
                    count++;
                }
                if (count_of_close != count_of_open) {
                    return false;
                }
                for (int i = 1; i < count - 1;i++) {
                    a += sentence[i];
                }
```

```
            }
            else {
                while (sentence[count] != '+' && sentence[count] != '-'
&& sentence[count] != '*') {
                    a += sentence[count];
                    if (sentence[count] == ')' && !isalpha(a[1])) {
                        return false;
                    }
                    count++;
                }
            }
            int ptr = count;
            while (count != sentence.length()-1) {
                if ((count == ptr) && (sentence[ptr] == '+' ||
sentence[ptr] == '-' || sentence[ptr] == '*')) {
                    count++;
                }
                b += sentence[count];
                count++;
            }
            if (Check(a), Check (b)) {
                return true;
            }
            else {
                return false;
            }
        }
        else {
            return false;
        }
    }
}

int main(){
    string sentence;
    ifstream file;
    ofstream file1("output.txt");
    fstream file2;
```

```
    cout << "Result:" << endl;
    file.open("input.txt");
    file2.open("output.txt", fstream::in |fstream::out);
    if (!file.is_open()) {
        cout << "Error opening file (input.txt)" << endl;
        return 0;
    }
    if (!file2.is_open()) {
        cout << "Error opening file (output.txt)" << endl;
        return 0;
    }
    while (!file.eof()) {
        getline(file, sentence);
        if (sentence == "") {

        }
        else{
            if (Check(sentence)) {
                cout << "\t" << sentence << " - True" << endl;
                file2 << sentence << " - True\n";
            }
            else {
                 cout << "\t" << sentence << " - False" << endl;
                 file2 << sentence << " - False\n";
            }
        }
    }
    file.close();
    file2.close();
}
```

Название файла: Makefile

```
all:
     g++ LR1_AiSD.cpp && clear && ./a.out
```