**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. 9304 |  | Тиняков С.А. |
| Преподаватель |  | Филатов Ар.Ю. |

Санкт-Петербург

2023

## Цель работы.

Научиться применять рекурсию в программировании. Для прохождения проверки отчёта надо добавить много воды, потому что 20 слов нормальных не придумаешь

## Задание.

Вариант 20.

Построить синтаксический анализатор понятия список\_параметров.

список\_параметров::= параметр | параметр , список\_параметров

параметр::= имя=цифра цифра | имя=(список\_параметров)

имя::= буква буква буква

## Основные теоретические положения.

Тут должна быть необязательная теория теория теория теория теория теория теория теория теория теория теория теория теория теория теория теория теория теория теория теория теория теория теория теория теория теория теория.

## Выполнение работы.

Программа считывает данные из входного файла. Алгоритм в своей сути прост: он рекурсивно идёт по понятиям. Всё начинается с понятия список\_параметров. Он, точно, начинается с понятия параметр. Алгоритм переходит к проверки понятия параметр. Он, точно, начинается с понятия имя, и алгоритм переходит к проверке этого понятия. Если после символо «=» в понятии параметр идет символ «(«, то алгоритм идёт проверять понятие список\_параметров. Аналогично, если в понятии список\_параметров после понятия параметр идёт запятая, то вызывается проверка для понятия список\_параметров.

На вход и выход программе подаются файлы через аргументы командной строки. Пробелы, символы табуляции и переноса строки и т. п. игнорируются. Во входном файле должно быть только то, что необходимо проанализировать. Никаких других предложений, символов и прочего быть не должно. В выходном файле выводится проанализированное понятие и то, что оно корректно, если понятие корректно. Иначе выводятся символы до ошибки(включая символ, на котором возникла она) и описание ошибки.

Класс ReaderWriter отвечает за ввод и вывод данных. В себе имеет два потока: один для входных данных, второй для выходных. Метод GetNextChar возвращает следующий символ из входного потока. Если достигнут конец, то возвращается нулевой символ. Метод GetAndWriteNextChar делает тоже самое, только дополнительно записывает символ в выходной поток(нулевой символ не записывается). Метов IsEOF() сообщает, достигнут ли конец входного потока. Методы WriteChar и WriteString записывают в выходной файл соответсвенно символ и строку.

Класс Analyzer проверяет корректность понятия список\_параметров. Метод CheckListParam проверяет корректность понятия список\_параметров. Метод ChecParam проверяет корректность понятия параметр. Метод ChecName проверяет корректность понятия имя. Метод StartAnalyz запускает проверку входных данных.

Разработанный программный код см. в приложении А.

Для проверки правильности работы были созданы тесты. Тесты проверяют, что анализ происходит правильно. Сначала тестируются базовые случаи, например: проверка корректности имени, отсутвие лишних символов, присутсвие запятой между параметрами в понятии список\_параметров и тд. Тестирование происходит при помощи python-скрипта. Запустить проверку тестов можно вручную командой python3 test.py или же при помощи утилиты make: make run\_tests.

python3 test.py

make run\_tests

Результаты тестирования см. в приложении Б.

На рис. 1 изображена схема, на которую надо сослаться

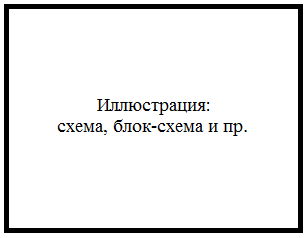


Рисунок 1 – Пример иллюстрации

## Выводы.

Научились применять рекурсию в программировании.

Была разработана программа для проверки понятия список\_параметров. Реализация проверки сделана через рекурсию. Класс ReaderWriter, который отвечает за входные и выходные данные, сделан через потоки, что позволяет при помощи класса Analyzer проанализировать любой объект, с которым можно взаимодействовать через интрефейс потоков(например строки). Метод WriteString принимает аргумент типа string\_view, что позволяет передовать методу как класс string, так и массив символов. Для проверки правильности работы программы были сделаны тесты.

# Приложение А Исходный код программы

Название файла: Source/lab1.cpp

#include<iostream>

#include<fstream>

#include<string\_view>

#include<sstream>

class ReaderWriter{

private:

std::istream& input;

std::ostream& output;

public:

ReaderWriter(std::istream& in, std::ostream& out):

input(in),

output(out)

{

}

~ReaderWriter(){

output << '\n';

}

char GetNextChar(){

char c;

input >> c;

if(IsEOF()) return '\0';

return c;

}

char GetAndWriteNextChar(){

char c;

input >> c;

if(IsEOF()) return '\0';

output << c;

return c;

}

bool IsEOF(){

return input.eof();

}

void WriteChar(char c){

output << c;

}

void WriteString(std::string\_view str){

output << str;

}

};

class Analyzer{

private:

ReaderWriter\* reader\_writer;

char last\_char;

std::string error\_msg;

bool CheckListParam(){

if(CheckParam()){

char next = reader\_writer->GetAndWriteNextChar();

if(next == ',') return CheckListParam();

else if(next == ')'){

last\_char = next;

return true;

}

else if(!next) return true;

error\_msg = std::string("List of parameters Error: Expected character \',\', but were given \'") + next + "\'.\n";

return false;

}

return false;

}

bool CheckParam(){

if(CheckName()){

char next = reader\_writer->GetAndWriteNextChar();

if(next == '='){

next = reader\_writer->GetAndWriteNextChar();

if(next == '('){

if(CheckListParam()){

if(last\_char == ')'){

last\_char = 0;

return true;

}

next = reader\_writer->GetAndWriteNextChar();

if(next == ')') return true;

if(next) error\_msg = std::string("Parameter Error: Expected character \')\', but were given \'") + next + "\'.\n";

else error\_msg = "Parameter Error: Expected character \')\', but end of input were reached.\n";

return false;

}

return false;

}else{

if(!(next >= '0' && next <='9')){

if(next) error\_msg = std::string("Parameter Error: Invalid define, character \'") + next + "\' is not digit.\n";

else error\_msg = "Parameter Error: Invalid define, expected digit, but end of input were reached.\n";

return false;

}

next = reader\_writer->GetAndWriteNextChar();

if(!(next >= '0' && next <='9')){

if(next) error\_msg = std::string("Parameter Error: Invalid define, character \'") + next + "\' is not digit.\n";

else error\_msg = "Parameter Error: Invalid define, expected digit, but end of input were reached.\n";

return false;

}

return true;

}

}

if(next) error\_msg = std::string("Parameter Error: Expected character \'=\', but were given \'") + next + "\'.\n";

else error\_msg = "Parameter Error: Expected character \'=\', but end of input were reached.\n";

return false;

}

return false;

}

bool CheckName(){

int count = 0;

while(count < 3){

char next = reader\_writer->GetAndWriteNextChar();

if(!(next >= 'A' && next <= 'z')){

if(next) error\_msg = std::string("Name Error: Invalid name, character \'") + next + "\' is not letter.\n";

else error\_msg ="Name Error: Invalid name, excepted letter, but end of input were reached.\n";

return false;

}

count++;

}

return true;

}

public:

Analyzer(ReaderWriter\* reader\_writer){

this->reader\_writer = reader\_writer;

last\_char = 0;

}

~Analyzer(){}

void StartAnalyz(){

bool correct = CheckListParam();

char next = reader\_writer->GetNextChar();

if(last\_char != 0 && error\_msg.empty()){

error\_msg = std::string("List of parameters Error: Expected character \',\', but were given \'") + last\_char + "\'.\n";

correct = false;

}

if(!correct){

reader\_writer->WriteString("\nIncorrect. ");

reader\_writer->WriteString(error\_msg);

}else reader\_writer->WriteString("\nCorrect.\n");

}

void AnalyzString(const std::string& str){

ReaderWriter\* old\_reader\_writer = reader\_writer;

std::istringstream stream\_str(str);

ReaderWriter new\_reader\_writer(stream\_str, std::cout);

reader\_writer = &new\_reader\_writer;

StartAnalyz();

reader\_writer = old\_reader\_writer;

}

};

int main(int argc, char\*\* argv){

if(argc < 3){

std::cout << "Usage: param\_analyzer /path/to/input /path/to/output\n";

return 1;

}

try{

std::ifstream in(argv[1]);

std::ofstream out(argv[2]);

if(!in.is\_open()){

std::cout << "Can't open file " << argv[1] << "\n";

return 2;

}

if(!out.is\_open()){

std::cout << "Can't open/create file " << argv[2] << "\n";

return 2;

}

/\* Example with string \*/

// std::stringstream in("AAA = 15\n");

{

ReaderWriter rw(in, out);

Analyzer analyzer(&rw);

analyzer.StartAnalyz();

}

in.close();

out.close();

}catch(std::exception& e){

std::cout << e.what() << "\n";

return 3;

}

return 0;

}

# Приложение Б Тестирование

Результаты тестирования представлены в таблице Б.1. Нужно больше слов. Нужно больше слов. Нужно больше слов. Нужно больше слов. Нужно больше слов. Нужно больше слов. Нужно больше слов. Нужно больше слов

Таблица Б.2 - Примеры тестовых случаев

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Коментарии |
|  | AAA = 15 | AAA=15  Correct. | Этот случай проверяет, что программа корректно проверит тривиальный случай |
|  | AA4 = 23 | AA4  Incorrect. Name Error: Invalid name, character '4' is not letter. | Этот случай проверяет, что программа найдёт ошибку в понятии имя |
|  | AAw = 2P | AAw=2P  Incorrect. Parameter Error: Invalid define, character 'P' is not digit. | Этот случай проверяет, что программа найдёт в понятии параметр, а конкретнее в цифрах |
|  | BBB=  (RRR =  56) | BBB=(RRR=56)  Correct. | Этот случай проверяет, что программа корректно проводит проверку понятия параметр=(список\_параметров) |