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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Алгоритмы и структуры данных»
Тема:Иерархические списки

Студент гр. 9304 Прокофьев М.Д.

Преподаватель Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Узнать о иерархическом списке и о его использовании в практике

Задание.

Логическое, проверка синтаксической корректности, добавить 4-ую операцию (которая может принимать 2 аргумента), префиксная форма

Выполнение работы.

Для выполнения работы был создан класс *Node*, содержащий элементы иерархического списка, и рекурсивная функция *found_prefix*, которая выполняет операцию проверки на синтаксис у заданного префиксного логического выражения.

Класс *Node* содержит в себе такие поля, как *next* и *value*. Поле *next* указывает на следующий элемент списка, а поле *value* указывает на другой список(на "новый уровень" того же списка). При создании иерархического списка был использован *variant* для избавления от потребности создавать отдельный флаг, показывающий, что находится в поле. Кроме того, в программе используются умные указатели во избежании утечек памяти.

Функция *found_prefix* работает следующим образом: Она принимает строку в качестве аргумента. При нахождении скобки происходит создание "нового уровня" списка, если нашлась буква, то она(буква) добавляется в список текущего уровня. Но кроме того, для создания "нового уровня" есть дополнительное и тем не менее обязательное условие: от этой скобки(соответственно, включая ее) должна идти подстрока таких типов, как класса *string*). Соответственно, это "операции" в логическом выражении. Также была добавлена 4-ая операция, которая идет в программе также в виде подстроки, которая используется в проверке в функции *found_prefix*, это операция *XOR*, которая принимает два аргумента(в программе представляется в виде подстроки должно быть определенное количество, и, кроме того, при каждом новом уровне должна быть своя "закрывающаяся скобка". При успешной проверке всех

вышеописанных условий в принимающейся строке, итератор, использующийся в функции, "доходит до конца строки", что означает что строка синтаксически верна, как префиксное логическое выражение, после чего в поток выводится сообщение *"Success"*. В любом ином случае, выводится *"Error"*.

Разработанный программный код см. в приложении А.

Тестирование.

Был написан python-скрипт для проведения тестов(testing.py). Посредством testing.py, в argv[](“входной” строке в программе) последовательно передаются строки из файлов SuccessTests.txt и ErrorTests.txt. Таким образом, при “запуске” Makefile, программа обрабатывает строки из вышеуказанных файлов(SuccessTests.txt и ErrorTests.txt), после чего выдает ответы в консоли.

Результаты тестирования изложены в приложении В.

Выводы.

Написана программа с использованием иерархического списка. Кроме того, в программе использовался *variant* и умные указатели. Использование иерархического списка при написании программы являлось необходимым, поскольку "обозначение" скобок в префиксном логическом выражении рациональнее представить в виде "списков, включающие в себе другие списки".

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: main.cpp

```
#include <iostream>
#include <variant>
#include <memory>

class Node {
    using NodePtr = std::shared_ptr<Node>;

public:
    NodePtr next{ nullptr };
    std::variant<char, NodePtr> value;
    Node() = default;
    ~Node() = default;
};

std::shared_ptr<Node> found_prefix(std::string& string, int& iterator)
{
    using NodePtr = std::shared_ptr<Node>;
    int counter = 0;
    short int arg = 3;
    NodePtr head_local = std::make_shared<Node>();
    NodePtr curr_local = head_local;
    if (!string.compare(iterator, 5, "(NOT "))
        arg = 1;
    head_local->value = string[iterator];
    while ((iterator != string.length()) && (counter < arg)) {
        if (!string.compare(iterator, 5, "(AND ") || !string.compare(iterator, 5,
"(NOT ") || !string.compare(iterator, 5, "(XOR "))
```

```
            iterator += 5;
        else if (!string.compare(iterator, 4, "(OR "))
            iterator += 4;
        else if (string[iterator + 1] == ' ')
            iterator += 2;
        else
            iterator++;
        if (!string.compare(iterator, 1, "(")) {
            curr_local->next = std::make_shared<Node>();
            curr_local = curr_local->next;
            curr_local->value = found_prefix(string, iterator);
            counter++;
        }
        else {
            if ((string[iterator] >= 'a') && (string[iterator] <= 'z') || (string[iterator]
== ')')) {
                curr_local->next = std::make_shared<Node>();
                curr_local = curr_local->next;
                curr_local->value = string[iterator];
                counter++;
            }
            else {
                return head_local;
            }
        }
    }
    if (string[iterator] != ')')
        iterator++;
    return head_local;
}
```

```
int main(int argc, char* argv[])
{
    std::shared_ptr<Node> head;
    int iterator = 0;
    int i = 0;
    std::string text;
    while(argv[++i]){
        text += std::string(argv[i]);
    }
    head = found_prefix(text, iterator);
    if (iterator == text.length() - 1)
        std::cout << "Success\n";
    else
        std::cout << "Error\n";
    return 0;
}
```

ПРИЛОЖЕНИЕ В

ТЕСТИРОВАНИЕ

Результаты тестирования представлены в таблице Б.1

Таблица Б.1 — Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	(AND a (NOT b))	Success	Правильное простое префиксное логическое выражение
2.	(AND a (NOT b)	Error	Неправильное выражение из-за отсутствия скобки справа
3.	(AND a(AND b(XOR c(OR d(AND f(OR e(AND x(OR y(NOT z))))))))	Success	Правильное длинное выражение
4.	(AND a(AND b(XOR c(OR d(AND f(OR e(AND x(OR y(NOT z b))))))))	Error	Неправильное длинное выражение, при NOT не должно быть двух аргументов
5.	(OR a)(AND b)(XOR c)	Error	Некорректное выражение, оно поскольку не является префиксной формой записи
6.	(a b AND)	Error	Некорректное выражение, оно поскольку не является префиксной формой записи
7.	(AND (OR a (XOR (NOT x)(AND y z)))(NOT c))	Success	Верное логическое выражение, содержащее, как список, “много уровней”, находящихся в разных местах
8.	(AND a b c d e f)	Error	Неправильное выражение, дано слишком много аргументов для AND
9.	(XOR a_b)	Error	Наличие недопустимых символов
10.	0.5	Error	Неверно заданное выражение