ММИНОБРНАУКИ 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: Рекурсивная обработка иерархических списков

Студент гр. 9384 _____ Николаев А.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Создать иерархический список. Проверить два иерархических списка на идентичность.

Задание.

ВАРИАНТ 12.

Проверить идентичность двух иерархических списков.

Выполнение работы.

Была реализована функция bool CheckList(s_expr* expA, s_expr* expB) принимающая два иерархических списка, обходящая их по глубине и ширине и сравнивающая значения атомов в списке и их положение. Возвращает 0 когда списки разные и 1 когда идентичные. В main() ввод был реализован при помощи считывания из файла. Программа запрашивает его название и принимает первую строку файла для преобразования ее в иерархический список, а затем забирает вторую и преобразует его во второй список, затем вызывается функция описанная выше и проверяет эти два списка.

Перед главной проверкой, в main() считывается размер введенных строк и в случае, если их размеры разные, то программа заканчивает свою работу досрочно и выводит 0.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

Входные данные	Выходные данные
(()a) и (()b) (()a) и (()a) ((()(a()b)) и ((()(a()b)) ((()(a()b)) и ((()(a()a))	0 1 1 0

Выводы.

В ходе выполнения лабораторный работы был создан иерархический список. Также были использованы алгоритмы для обхода иерархических списков и их сравнения.

ПРИЛОЖЕНИЕ

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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

typedef char Base; // базовый тип элементов (атомов)

struct s_expr;
struct  two_ptr
{
	s_expr* hd;
	s_expr* tl;
};

struct s_expr {
	bool IsAtom; // true: atom, false: pair
	union
	{
		Base atom;
		two_ptr pair;
	} node;
};

s_expr* CreateList(std::string& str, int& pos)
{
	s_expr* hd = nullptr;
	s_expr* st = nullptr;
	s_expr* tl = nullptr;
	do
	{
		if (str[pos] == '(') {//если начинается s_expr и это не atom
			st = new s_expr;
			if (!hd) {//если указатель на голову пуст
				hd = st;
				tl = st;
			}
			else {
				tl->node.pair.tl = st;
				tl = st;
			}
```

```
                pos++;
                st->IsAtom = false;
                st->node.pair.tl = nullptr;
                st->node.pair.hd = CreateList(str, pos);
            }
            else if (str[pos] == ')') {//если s_expr закончилось
                pos++;
                return hd;
            }
            else {//если atom
                if (!hd) {
                    hd = new s_expr;
                    hd->IsAtom = true;
                    hd->node.pair.tl = nullptr;
                    hd->node.atom = str[pos];
                    tl = hd;
                }
                else {
                    st = new s_expr;
                    tl->node.pair.tl = st;
                    st->IsAtom = true;
                    st->node.pair.tl = nullptr;
                    st->node.atom = str[pos];
                    tl = st;
                }
                pos++;
            }
        } while (str[pos]);
        return hd;
}

bool CheckBrackets(const std::string str)
{
        size_t open = std::count(str.begin(), str.end(), '(');
        size_t closed = std::count(str.begin(), str.end(), ')');
        if (open > closed)
        {
            std::cout << "Missing closed bracket" << std::endl;
            return false;
        }
        else if (closed > open)
        {
            std::cout << "Missing open bracket" << std::endl;
            return false;
```

```
        }
        return true;
}

bool CheckList(s_expr* expA, s_expr* expB)
{
        if (expA->IsAtom && expB->IsAtom)
        {
                if (expA->node.pair.tl && expB->node.pair.tl)
                {
                        if (expA->node.atom == expB->node.atom)
                        {
                                return CheckList(expA->node.pair.tl, expB->node.pair.tl);
                        }
                        else
                        {
                                return false;
                        }
                }
                else if (!expA->node.pair.tl && !expB->node.pair.tl)
                {
                        if (expA->node.atom == expB->node.atom)
                        {
                                return true;
                        }
                        else
                        {
                                return false;
                        }
                }
                else
                {
                        return false;
                }
        }
        else if (!expA->IsAtom && !expB->IsAtom)
        {
                if (expA->node.pair.hd && expB->node.pair.hd)
                {
                        if (CheckList(expA->node.pair.hd, expB->node.pair.hd))
                        {
                                if (expA->node.pair.tl && expB->node.pair.tl)
                                {
```

```
                                        return CheckList(expA->node.pair.tl, expB->node.pair.tl);
                                }
                                else if (!expA->node.pair.tl && !expB->node.pair.tl)
                                {
                                        return true;
                                }
                                else
                                {
                                        return false;
                                }
                        }
                }
                else if (!expA->node.pair.hd && !expB->node.pair.hd)
                {
                        if (expA->node.pair.tl && expB->node.pair.tl)
                                return CheckList(expA->node.pair.tl, expB->node.pair.tl);
                        else
                                return true;
                }
                else
                {
                        return false;
                }
        }

        return false;
}

void PrintList(s_expr head /*std::ofstream& out*/) {

        if (head.IsAtom) {
                std::cout << head.node.atom;
                if (head.node.pair.tl)
                        PrintList(*(head.node.pair.tl));
        }
        else if (head.node.pair.hd) {
                std::cout << '(';
                if (head.node.pair.hd)
                        PrintList(*(head.node.pair.hd));
                std::cout << ')';
                if (head.node.pair.tl)
                        PrintList(*(head.node.pair.tl));
        }
```

```
}
int main() {
	int posA = 0;
	int posB = 0;

	std::string str;
	std::ifstream in("in.txt");

	s_expr* expA;
	s_expr* expB;

	std::getline(in, str);
	if (CheckBrackets(str))
		expA = CreateList(str, posA);
	else
	{
		std::cout << "Brackets error!\n";
		return 0;
	}

	std::getline(in, str);
	if (CheckBrackets(str))
		expB = CreateList(str, posB);
	else
	{
		std::cout << "Brackets error!\n";
		return 0;
	}

	if (posA != posB)
	{
		std::cout << false;
		return 0;
	}

	std::cout << CheckList(expA, expB);

	std::cout << '\n';
	return 0;
}
```