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: Иерархические списки

Студент гр. 9304 _____ Попов Д.С.

Преподаватель _____ Филатов А.Ю

Санкт-Петербург

2020

Цель работы.

Изучить понятие иерархического списка.

Реализовывать программу с использованием иерархического списка в языке C++.

Задание.

Вариант 12.

Проверить идентичность двух иерархических списков.

Выполнение работы.

Программа принимает в качестве аргументов два параметра: текстовые файлы со структурами и значениями атомов. Происходит считывание информации из файлов в отдельные строки, которые выступают аргументами для конструктора класса *List.* Все необходимые преобразования и проверки происходят внутри класса. Сначала приватный метод *checkValue()* преобразует строку со значениями атомов в вектор целочисленных значений *valueAtom*, все символы не относящиеся к цифрам будут проигнорированы. Затем происходит проверка строки содержащей структуру списка методом *checkStruct()*, проверяется корректность количества открывающих и закрывающих скоб, их расположение, а так же соответствие количества значений атомов к самим атомам. В случае некорректной структуры, первый элемент списка *firstElement* приравнивается к *nullptr,* программа выдает ошибку и заканчивает работу. Если проверка структуры пройдена, то метод *createList()* производит «сборку» данного списка и записывает все адреса узлов в вектор *NodePtr*.

Для проверки на идентичность используется перегруженный оператор == , который изначально проверяет структуры обоих списков, если они окажутся идентичны, то программа сообщит об этом и перейдет к проверке узлов — атомов. Каждая проверка сопровождается выводом сравниваемых

значений в консоль. Если структуры и значения атомов идентичны — списки идентичны.

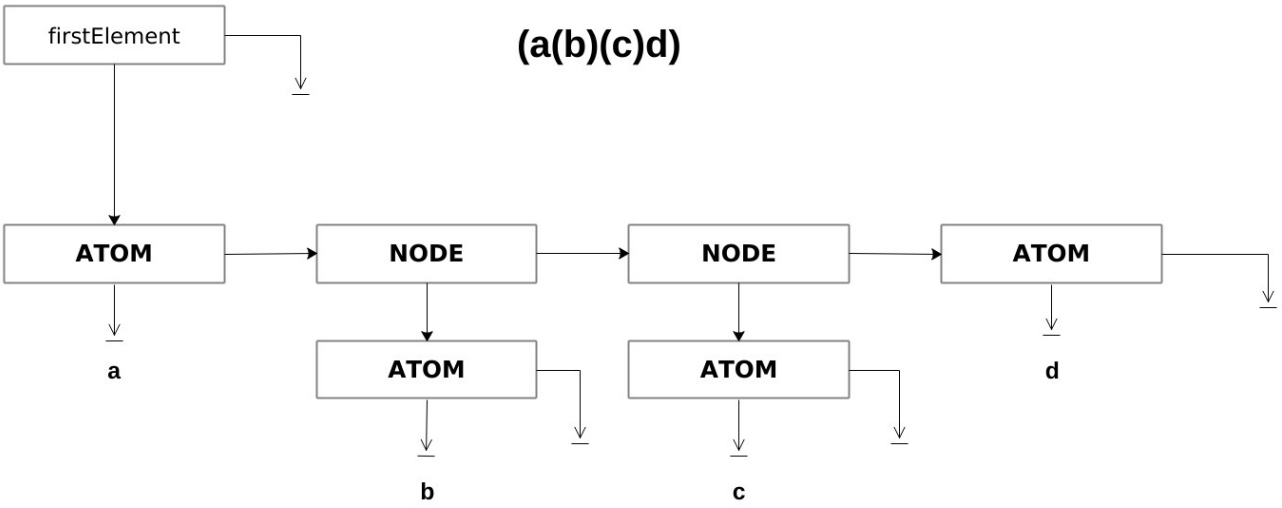
Разработанный программный код см. в приложении А.

Формат входных и выходных данных.

На вход программе подается два текстовых файла содержащие 2 строки: структура списка и значения атомов.

Программа должна определить идентичны ли они.

Ниже представлена визуализация одного из тестируемых иерархических списков.



Тестирование.

Для проведения тестирования был написан bash-скрипт ./script .Скрипт запускает программу где в качестве входных аргументов служат заранее подготовленные файлы, расположенные в папке ./Tests

Результаты тестировния см. в приложении Б.

Выводы.

Было изучено понятие иерархического списка. Был реализован иерархический список с помощью языка программирования C++, с использованием std::variant.

Была реализована программа, создающая два иерархических списка и проверяющая их идентичность.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <variant>
#include <vector>
#include <cctype>

#include "Node.h"
#include "List.h"



int main(int argc, char* argv[]){

    setlocale(LC_ALL, "ru");

    if(argc != 3)
    {
        std::cout << "Неверное кол-во аргументов!" << std::endl;
        return 1;
    }

    std::fstream file1(argv[1]);

    if(!file1)
    {
         std::cout << "Файл " << argv[1] << " не может быть открыт
на чтение!" << std::endl;
        return 1;
    }

    std::fstream file2(argv[2]);

    if(!file2)
    {
         std::cout << "Файл " << argv[2] << " не может быть открыт
на чтение!" << std::endl;
        return 1;
    }

     std::string structList1 {};    // Строка со структурой первого
списка.
    std::string valueAtom1 {};     // Строка со значениями атомов.

    std::string structList2 {};
    std::string valueAtom2 {};

    getline(file1, structList1);
    getline(file1, valueAtom1);

    getline(file2, structList2);
```

```
        getline(file2, valueAtom2);

        List *list1 = new List(structList1, valueAtom1);
        List *list2 = new List(structList2, valueAtom2);

        if(list1->getFirstElement() == nullptr){

                std::cout << "Неверная структура или кол-во аргументов
списка!" << std::endl;
            return 1;
        }

        if(list2->getFirstElement() == nullptr){

                std::cout << "Неверная структура или кол-во аргументов
списка!" << std::endl;
            return 1;
        }

        if(*list1 == *list2){
            std::cout << "Списки идентичны!" << std::endl;
        }else{
            std::cout << "Это разные списки!" << std::endl;
        }

        delete list1;
        delete list2;

        return 0;
     }
```

Название файла: List.cpp

```
#include "List.h"
#include "Node.h"

#include <iostream>



List::List(std::string newStructList, std::string valueAtom){

    structList = newStructList;

    checkValue(valueAtom);

    if(!checkStruct()){
        firstElement = nullptr;
    }else{
        createList();
    }
}



List::~List(){
```

```
    if(firstElement){
        delete firstElement;
    }
}



void List::checkValue(std::string stringValue){

    // Преобразуем строку в целочисленный вектор значений атомов.
    // Игнорируются все буквы, знаки препинания и тд.

    bool flag = false;

    for(int i = 0; stringValue[i] != '\0'; i++){
        if(isdigit(stringValue[i])){
            if(flag){
                continue;
            }else{
                valueAtom.emplace_back(atoi(&stringValue[i]));
                flag = true;
            }
        }else{
            flag = false;
        }
    }
}



Node* List::getFirstElement(){
    return firstElement;
}



bool List::checkStruct(){

    // Проверка строки на корректность.

    int level = 0;
    size_t countAtom = 0;
    size_t count = valueAtom.size();

    if(structList.size() == 0){
        return 0;
    }

    for(size_t i = 0; i < structList.size(); i++){

        if(structList[i] == '('){
            level++;
        }

        if(isalpha(structList[i])){
            countAtom++;
        }
```

```
        if(structList[i] == ')'){
            level--;
        }

        if(level < 0){
            return 0;
        }
    }

    if(level != 0){
        return 0;
    }

    if(countAtom != count){
        return 0;
    }

    return 1;
}



void List::createList(){

    // Создание списка на основе полученных данных.

    size_t count = 0;
    size_t first = 0;
    size_t second = 0;
    std::string str = structList;
    std::vector<int> vec = valueAtom;
    std::vector<Node*> ptr = NodePtr;
    firstElement = new Node;
    ptr.emplace_back(firstElement);

     auto PR = [&second, &str, &vec, &count, &ptr](size_t first, Node*
next, auto &&PR){

        second++;

        if(str[first] == '(' && str[second] == ')'){
            if(second - first == 1){
                next->setValue(nullptr);
            }
            return;
        }

        if(str[first] == '(' && str[second] == '('){
            if(second - first == 1){
                next->setValue(new Node);
                ptr.emplace_back(std::get<Node*>(next->getValue()));
                PR(second, std::get<Node*>(next->getValue()), PR);
            }else{
                Node* tmp = std::get<Node*>(next->getValue());
                for(;;){
                    if(tmp->getNextPtr() == nullptr){
                        break;
                    }else{
```

```
                    tmp = tmp->getNextPtr();
                }
            }
            tmp->setNextPtr(new Node);
            ptr.emplace_back(tmp->getNextPtr());
            PR(second, tmp->getNextPtr(), PR);
        }
    }

    if(str[first] == '(' && isalpha(str[second])){
        if(second - first == 1){
            next->setValue(new Node(vec[count]));
            count++;
            ptr.emplace_back(std::get<Node*>(next->getValue()));
        }else{
            Node* tmp = std::get<Node*>(next->getValue());
            for(;;){
                if(tmp->getNextPtr() == nullptr){
                    break;
                }else{
                    tmp = tmp->getNextPtr();
                }
            }
            tmp->setNextPtr(new Node(vec[count]));
            count++;
            ptr.emplace_back(tmp->getNextPtr());
        }
    }

    PR(first, next, PR);
};

PR(first, firstElement, PR);
NodePtr = ptr;
}



bool operator==(const List &l1, const List &l2){

    std::string s1 = l1.structList;
    std::string s2 = l2.structList;
    size_t count1 = 0;
    size_t count2 = 0;

    if(s1.size() == s2.size()){

        for(size_t i = 0; i < s1.size(); i++){

            if(isalpha(s1[i])){
                s1[i] = 'a';
                count1++;
            }
        }

        for(size_t i = 0; i < s2.size(); i++){

            if(isalpha(s2[i])){
```

```
                s2[i] = 'a';
            }
        }

        if(s1 == s2){

            std::cout << "Идентичные структуры!\n";
            bool flag = 1;

            for(size_t i = 0; i < l1.NodePtr.size(); i++){

                if(std::holds_alternative<int>(l1.NodePtr[i]->getValue())
+ std::holds_alternative<int>(l2.NodePtr[i]->getValue()) == 2){

                    if(flag){
                         std::cout << "Кол-во атомов = " << count1 << '\
n';
                        std::cout << "Значения атомов:\n";
                        flag = 0;
                    }

                          if(std::get<int>(l1.NodePtr[i]->getValue()) ==
std::get<int>(l2.NodePtr[i]->getValue())){
                                std::cout << std::get<int>(l1.NodePtr[i]-
>getValue()) << " == " << std::get<int>(l2.NodePtr[i]->getValue()) << '\
n';
                        count2++;
                    }else{
                                std::cout << std::get<int>(l1.NodePtr[i]-
>getValue()) << " != " << std::get<int>(l2.NodePtr[i]->getValue()) << '\
n';
                    }

                }
            }

        }else{
            std::cout << "Различные структуры!\n";
            return 0;
        }

    }else{
        std::cout << "Различные структуры!\n";
        return 0;
    }

    if(count1 == count2){
        return 1;
    }

    return 0;
}



bool operator!=(const List &l1, const List &l2){
    return !(l1 == l2);
}
```

Название файла: List.h

```
#pragma once

#include <vector>
#include <string>
#include <iostream>



class Node;

class List{
    Node* firstElement;             // Адрес первого элемента.
    std::vector<Node*> NodePtr;     // Адреса узлов.
    std::string structList;         // Структура списка.
    std::vector <int> valueAtom;    // Значения атомов.
    void checkValue(std::string stringValue);
    void createList();
    bool checkStruct();
public:
    List(std::string structList, std::string valueAtom);
    ~List();
    Node* getFirstElement();
    friend bool operator==(const List &l1, const List &l2);
    friend bool operator!=(const List &l1, const List &l2);
};
```

Название файла: Node.cpp

```
#include "Node.h"
#include <iostream>


Node::Node(){
    next = nullptr;
}



Node::Node(std::variant<Node*, int> newValue){
    next = nullptr;
    value = newValue;
}




Node::~Node(){
    if(next)
    {
```

```
        delete next;
    }

    if(std::holds_alternative<Node*>(value))
    {
        delete std::get<Node*>(value);
    }
}



Node* Node::getNextPtr(){
    return next;
}



std::variant<Node*, int> Node::getValue(){
    return value;
}



void Node::setNextPtr(Node* newPtr){
    next = newPtr;
}



void Node::setValue(std::variant<Node*, int> newValue){
    value = newValue;
}
```

Название файла: Node.h

```
#pragma once

#include "variant"



class Node {
    std::variant <Node*, int> value;
    Node* next;
public:
    Node();
    Node(std::variant<Node*, int>);
    ~Node();
    Node* getNextPtr();
    std::variant<Node*, int> getValue();
    void setNextPtr(Node* newPtr);
    void setValue(std::variant<Node*, int> newValue);
};
```

Название файла: script

```
#!/bin/bash

arg1=$(cat Tests/test1-1.txt)
arg2=$(cat Tests/test1-2.txt)
echo -e "_____________________________________________\n"
echo -e "Test 1:\n"
echo " argument 1 ="
echo "$arg1"
echo " argument 2 ="
echo -e "$arg2\n"
./lab2 ./Tests/test1-1.txt ./Tests/test1-2.txt

arg1=$(cat Tests/test2-1.txt)
arg2=$(cat Tests/test2-2.txt)
echo -e "_____________________________________________\n"
echo -e "Test 2:\n"
echo " argument 1 ="
echo "$arg1"
echo " argument 2 ="
echo -e "$arg2\n"
./lab2 ./Tests/test2-1.txt ./Tests/test2-2.txt

arg1=$(cat Tests/test3-1.txt)
arg2=$(cat Tests/test3-2.txt)
echo -e "_____________________________________________\n"
echo -e "Test 3:\n"
echo " argument 1 ="
echo "$arg1"
echo " argument 2 ="
echo -e "$arg2\n"
./lab2 ./Tests/test3-1.txt ./Tests/test3-2.txt

arg1=$(cat Tests/test4-1.txt)
arg2=$(cat Tests/test4-2.txt)
echo -e "_____________________________________________\n"
echo -e "Test 4:\n"
echo " argument 1 ="
echo "$arg1"
echo " argument 2 ="
echo -e "$arg2\n"
./lab2 ./Tests/test4-1.txt ./Tests/test4-2.txt

arg1=$(cat Tests/test5-1.txt)
arg2=$(cat Tests/test5-2.txt)
echo -e "_____________________________________________\n"
echo -e "Test 5:\n"
echo " argument 1 ="
echo "$arg1"
echo " argument 2 ="
echo -e "$arg2\n"
./lab2 ./Tests/test5-1.txt ./Tests/test5-2.txt
echo -e "_____________________________________________\n"
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ ПРОГРАММЫ

Результаты тестирования представлены в табл.1

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	(a) 2 (b) 2	Идентичные структуры! Кол-во атомов = 1 Значения атомов: 2 == 2 Списки идентичны!	Значение атома второго списка == первому
2.	(()()()) (()()())	Идентичные структуры! Списки идентичны!	Значений атомов нет.
3.	(a(b)(c)n) 10 20 30 40 (a(b)(a)y) 1 20 30 40	Идентичные структуры! Кол-во атомов = 4 Значения атомов: 10 != 1 20 == 20 30 == 30 40 == 40 Это разные списки!	Значение первого атома первого списка != второму
4.	)a(15 (a) 15	Неверная структура или кол-во аргументов списка!	Структура первого списка не верна
5.	(a(b)c) 1 2 3 (abc) 1 2 3	Различные структуры! Это разные списки!	