**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 Ламбин А.В.

Преподаватель Филатов А.Ю.

Санкт-Петербург
2020

**Цель работы.**

Ознакомиться с понятием иерархического списка, реализовать иерархический список для решения поставленной задачи с помощью языка программирования C++.

**Задание.**

Вариант 30.

Разработать программу, создающую иерархический список, соответствующий иерархическому содержанию, записанному в файле. Соединить 2 заданных соседних элемента (находящихся в одном подсписке). Информация из названий и текстов не должна потеряться. Может потребоваться перенос названия в состав текста или перенумерация элементов, подчинённых одному из объединяемых.

**Выполнение работы.**

При запуске программы, ей передаётся файл с исходными данными. В случае, если файла на входе нет, программа выводит ошибку и завершается. Переданный файл открывается на чтение. Если открыть файл не предоставляется возможности, выводится ошибка, программа завершается.

В динамический массив строк записываются все строки из входного файла, сам файл после этого закрывается.

Создаётся указатель *list* на объект класса *List*. После чего считываются номера строк, которые необходимо соединить. Считанные строки передаются методу *connect()* класса *List*. В конце открывается файл *result.txt* на запись, куда и выводится результат работы программы. Файл закрывается, выделенная память отчищается.

Класс *Node*:

Для реализации узлов иерархического списка создан класс *Node*, приватными полями которого являются массив целый чисел *num*, содержащий

иерархический номер записи, целое число *count* – количество элементов массива *num*, строка *name* – название записи, поле *data*, являющееся строкой или указателем на другую подстроку, и указатель на следующий элемент подсписка *next*.

Конструктору класса *Node* передаются части входных строк, указатель на следующий элемент и данные, которые являются либо строкой с текстом записи, либо указателем на другую подстроку. В конструкторе заполняются поля класса *Node*, в деструкторе – освобождается выделенная динамическая память.

Помимо этого в данном классе перегружен оператор << для вывода экземпляров класса. Сперва выводится *count – 1* символ табуляции, после через точку элементы массива *num*, название записи и её текст.

Класс *List*:

В классе *List* определены приватные поля *head* (указатель на элемент класса Node), *text* (массив строк) и *size* (количество элементов в массиве строк).

В конструкторе этого класса заполняется массив *text* и вызывается рекурсивная функция *createNode()*. В функции *createNode() n*-ая строка массива *text* разделяется на иерархический номер, название записи и её текст, если он имеется. После чего, в зависимости от иерархического номера создаётся либо узел, либо атом иерархического списка.

В деструкторе вызывается рекурсивная функция *deleteNode()*. Из неё вызывается эта же функция для поля *data* экземпляра класса *Node*, если это поле имеет тип *Node **, и эта же функция для следующего элемента подсписка. В конце освобождается память, выделенная под переданный экземпляр класса.

Перегруженный оператора << вызывает перегруженный оператор << для головы списка.

Метод *connect()* объединяет два элемента списка в один, если данные элементы существуют, находятся в одном подсписке и не равны друг другу. Если все условия выполняются, то в элемент с меньшим иерархическим номером добавляется информация из другого, после чего поле *next* предшествующего

элемента большего из сливаемых становится равным полю *next* этого самого элемента. Память, выделенная под этот элемент освобождается. В конце вызывается функция *changeNum()* для перенумерации всех элементов списка, очищается используемая программой память.

Для поиска элементов списка по их иерархическим номерам была написана рекурсивная функция *findNode()*, которая проходится по всем элементам и, в случае совпадения, возвращает этот элемент.

Для перенумерации всех элементов была реализована функция *changeNum()*, заменяющая относительные номера записей на минимально возможные.

Разработанный программный код см. в приложении А.

**Тестирование.**

Запуск программы начинается с ввода команды “make”, что приведёт к компиляции и линковки программы, после чего будет создан исполняемый файл lab2. Запуск программы производится командой “./lab2 <input.txt>”, где input.txt – входной файл. После вводятся иерархические номера сливаемых строк в любом порядке. Вывод программы осуществляется в файл result.txt.

Тестирование программы производится с помощью скрипта script.py, написанного на языке программирования Python. Запуск скрипта осуществляется командой “python3 script.py” в директории tests.

Результаты тестирования см. в приложении Б.

**Выводы.**

Было проведено ознакомление с понятием иерархического списка, был получен навык создания иерархических списков с помощью языка программирования C++ и использования контейнера std::variant.

Была разработана программа, создающая и обрабатывающая иерархический список.

На мой взгляд, использование иерархических списков при решении поставленной задачи не оправдано, поскольку при реализации используется рекурсивная обработка структуры данных, что потребляет больше ресурсов и занимает больше времени.

## ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include "list.h"

int main(int argc, char *argv[]) {
    if (argc < 2) {
        std::cerr << "Error: No input file\n";
        return 0;
    }
    std::ifstream in(argv[1]);
    if (!in.is_open()) {
        std::cerr << "Error: Wrong input file\n";
        return 0;
    }

    unsigned int size = 30, n = 0;
    auto *text = new std::string[size];
    while (std::getline(in, text[n++])) {
        if (n == size) {
            size += 30;
            std::string *buf = text;
            text = new std::string[size];
            for (int i = 0; i < size - 30; i++)
                text[i] = buf[i];
            delete [] buf;
        }
    }
    size = n - 1;
    in.close();

    List *list = new List(text, size);

    std::string num1, num2;
    std::cin >> num1 >> num2;
    list->connect(num1, num2);
    std::ofstream out(argv[2]);
    out << *list;
    out.close();

    delete list;
    delete [] text;
    return 0;
}
```

Название файла: node.h

```
#ifndef NODE_H
#define NODE_H

#include <iostream>
```

```
#include <string>
#include <variant>

class Node {
public:
    Node (std::string, std::string, std::variant<std::string, Node
*>, Node *);
    ~Node ();
    friend std::ostream &operator<< (std::ostream &, const Node *);

private:
    int *num;
    int count;
    std::string name;
    std::variant<std::string, Node *> data;
    Node *next;
    friend class List;

};

#endif //NODE_H
```

Название файла: node.cpp

```
#include "node.h"

Node::Node (std::string number, std::string name,
std::variant<std::string, Node*> data, Node *next) : name(name),
data(data), next(next) {
    int count = 1;
    for (char i : number)
        if (i == '.')
            count++;
    this->num = new int [count];
    this->count = count;
    for (int i = 0; i < count - 1; i++) {
        int pos = number.find('.');
        num[i] = std::stoi(number.substr(0, pos));
        number.erase(0, pos + 1);
    }
    num[count - 1] = std::stoi(number);
}

Node::~Node () {
    delete [] this->num;
}

std::ostream &operator<< (std::ostream &out, const Node *node) {
    if (node->count > 1) {
        for (int i = 0; i < node->count - 1; i++)
            out << '\t';
    }

    if (node->count == 1) {
        out << node->num[0] << ' ';
    } else {
        for (int i = 0; i < node->count - 1; i++)
```

```
            out << node->num[i] << '.';
        out << node->num[node->count - 1] << ' ';
    }

    out << node->name << ' ';

    if (std::holds_alternative<Node *>(node->data)) {
        if (std::get<Node *>(node->data) != nullptr) {
            out << '\n' << std::get<Node *>(node->data);
        }
    } else {
        out << std::get<std::string>(node->data) << '\n';
    }

    if (node->next != nullptr)
        out << node->next;
    return out;
}
```

Название файла: list.h

```
#ifndef LIST_H
#define LIST_H

#include <iostream>
#include <string>
#include "node.h"

class List {
public:
    List (std::string *, unsigned int);
    ~List ();
    friend std::ostream &operator<< (std::ostream &, const List &);
    int connect (std::string, std::string);

private:
    Node *createNode (unsigned int);
    void deleteNode (Node *);
    Node *findNode (Node *, int *, int, int);
    void changeNum (Node *, int *, int);

    Node *head;
    std::string *text;
    unsigned int size;

};

#endif //LIST_H
```

Название файла: list.cpp

```
#include "list.h"

List::List (std::string *text, unsigned int size) : text(text),
size(size) {
    this->text = new std::string [size];
```

```
    for (int i = 0; i < size; i++)
        this->text[i] = text[i];
    this->head = createNode(0);
    delete [] this->text;
}

List::~List () {
    deleteNode(head);
}

std::ostream &operator<< (std::ostream &out, const List &list) {
    out << list.head;
    return out;
}

int List::connect (std::string number1, std::string number2) {
    int count;
    int count1 = 1, count2 = 1;
    for (char i : number1)
        if (i == '.')
            count1++;
    for (char i : number2)
        if (i == '.')
            count2++;
    if (count1 == count2) {
        count = count1;
    } else {
        std::cerr << "Error: elements are not in one sublist\n";
        return 1;
    }

    int *num1 = new int [count];
    int *num2 = new int [count];
    for (int i = 0; i < count - 1; i++) {
        int pos = number1.find('.');
        num1[i] = std::stoi(number1.substr(0, pos));
        number1.erase(0, pos + 1);
        pos = number2.find('.');
        num2[i] = std::stoi(number2.substr(0, pos));
        number2.erase(0, pos + 1);
    }
    num1[count - 1] = std::stoi(number1);
    num2[count - 1] = std::stoi(number2);

    for (int i = 0; i < count; i++) {
        if (i == count - 1) {
            if (num1[i] == num2[i]) {
                std::cerr << "Error: elements are equal\n";
                delete [] num1;
                delete [] num2;
                return 1;
            } else if (num1[i] > num2[i]) {
                int *temp = num1;
                num1 = num2;
                num2 = temp;
                break;
            } else {
                break;
```

```
                }
            }
            if (num1[i] != num2[i]) {
                std::cerr << "Error: elements are not in one sublist\n";
                delete [] num1;
                delete [] num2;
                return 1;
            }
        }

        Node *node1 = findNode(head, num1, count, 0);
        Node *node2 = findNode(head, num2, count, 0);
        if (node1 == nullptr || node2 == nullptr) {
            std::cerr << "Error: no element with such number\n";
            delete [] num1;
            delete [] num2;
            return 0;
        }

        if (std::holds_alternative<Node *>(node2->data)) {
            if (std::holds_alternative<Node *>(node1->data)) {
                Node *temp = std::get<Node *>(node1->data);
                while (temp->next != nullptr)
                    temp = temp->next;
                temp->next = std::get<Node *>(node2->data);
            } else {
                node1->data = std::get<Node *>(node2->data);
            }
        } else {
            if (std::holds_alternative<std::string>(node1->data)) {
                std::string str1, str2;
                str1 = std::get<std::string>(node1->data);
                str2 = std::get<std::string>(node2->data);
                node1->data = str1.erase(str1.size() - 1, 1) + ". " +
str2.erase(0, 1);
            }
        }
        node1->name += " - " + node2->name;

        Node *prev = nullptr;
        int *prevNum = new int [count];
        for (int i = 0; i < count; i++)
            prevNum[i] = num2[i];
        for (int i = num2[count - 1] - 1; i >= 0; i--) {
            prevNum[count - 1] = i;
            if ((prev = findNode(this->head, prevNum, count, 0)) !=
nullptr)
                break;
        }
        prev->next = node2->next;
        delete [] prevNum;
        delete node2;

        int *num = new int [count];
        for (int i = 0; i < count; i++)
            num[i] = 0;
        changeNum(this->head, num, 1);
        delete [] num;
```

```
        delete [] num1;
        delete [] num2;
        return 0;
    }

    Node *List::createNode (unsigned int n) {
        unsigned int pos1 = text[n].find(' '), pos2 = 0;
        std::string num = text[n].substr(0, pos1), name, str;

        if (text[n].find('\"') != std::string::npos) {
            pos2 = text[n].find('\"');
            name = text[n].substr(pos1 + 1, pos2 - pos1 - 2);
            str = text[n].substr(pos2, text[n].size());
        } else {
            name = text[n].substr(pos1 + 1, text[n].size());
            str = "";
        }

        if (n == size - 1) {
            return new Node(num, name, str, nullptr);
        } else {
            int curCount = 0;
            for (char i : num)
                if (i == '.')
                    curCount++;
            int nextCount = 0;
            for (char i : text[n + 1].substr(0, text[n + 1].find(' ')))
                if (i == '.')
                    nextCount++;

            if (nextCount > curCount) { // node
                unsigned int next = 0;
                bool flag = false;
                for (unsigned int j = n + 1; j < size; j++) {
                    int jCount = 0;
                    for (char i : text[j].substr(0, text[j].find(' ')))
                        if (i == '.')
                            jCount++;
                    if (jCount == curCount) {
                        flag = true;
                        next = j;
                        break;
                    }
                    if (jCount < curCount) {
                        break;
                    }
                }

                if (flag) {
                    return new Node(num, name, createNode(n + 1),
createNode(next));
                } else {
                    return new Node(num, name, createNode(n + 1),
nullptr);
                }
            } else { // atom
                if (nextCount < curCount)
```

```
                    return new Node(num, name, str, nullptr);
                else
                    return new Node(num, name, str, createNode(n + 1));
            }
        }
    }

    void List::deleteNode (Node *cur) {
        if (cur == nullptr)
            return;
        if (std::holds_alternative<Node *>(cur->data))
            deleteNode (std::get<Node *>(cur->data));
        deleteNode (cur->next);
        delete cur;
    }

    Node *List::findNode (Node *node, int *num, int count, int n) {
        while (node->num[n] != num[n]) {
            node = node->next;
            if (node == nullptr)
                return nullptr;
        }

        if (n < count - 1)
            if (std::holds_alternative<Node *>(node->data))
                node = findNode(std::get<Node *>(node->data), num, count,
n + 1);
            else
                return nullptr;
        else
            return node;

        return node;
    }

    void List::changeNum (Node *cur, int *num, int count) {
        for (int i = 0; i < count; i++)
            cur->num[i] = num[i];
        if (std::holds_alternative<Node *>(cur->data)) {
            num[count] = 0;
            changeNum(std::get<Node *>(cur->data), num, count + 1);
            num[count] = 0;
        }
        if (cur->next != nullptr) {
            num[count - 1]++;
            changeNum(cur->next, num, count);
        }
    }
```

# ПРИЛОЖЕНИЕ Б

# ТЕСТИРОВАНИЕ

Таблица Б.1 – Примеры тестовых случаев

Входной файл:

```
0 Предисловие
1 Книга I
1.1 Глава 1 "Название первой главы"
1.2 Глава 2 "Название второй главы"
1.3 Глава 3 "Название третьей главы"
2 Книга II
2.1 Глава 1
2.1.1 Подглава 1 "Название первой подглавы"
2.1.2 Подглава 2 "Название второй подглавы"
3. Эпилог
```

| № п/п | Входные данные | Выходные данные | Комментарии |
|---|---|---|---|
| 1. | 0 1.1 | Error: elements are not in one sublist | Элементы из разных подсписков |
| 2. | 1 4 | Error: no element with such number | Обращение к несуществующему элементу |
| 3. | 2 2 | Error: elements are equal | Обращение к одинаковым элементам |
| 4. | 0 1 | 0 Предисловие - Книга I<br>0.0 Глава 1 "Название первой главы"<br>0.1 Глава 2 "Название второй главы"<br>0.2 Глава 3 "Название третьей главы"<br>1 Книга II<br>1.0 Глава 1<br>1.0.0 Подглава 1 "Название первой подглавы"<br>1.0.1 Подглава 2 "Название второй подглавы"<br>2 Эпилог | Корректная работа программы |
| 5. | 2.1.2 2.1.1 | 0 Предисловие<br>1 Книга I<br>1.0 Глава 1 "Название первой главы"<br>1.1 Глава 2 "Название второй главы"<br>1.2 Глава 3 "Название третьей главы"<br>2 Книга II<br>2.0 Глава 1<br>2.0.0 Подглава 1 - Подглава 2 "Название первой подглавы. Название второй подглавы"<br>3 Эпилог | Корректная работа программы |