**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 Аксёнова Е.А.

Преподаватель Филатов Ар.Ю.

Санкт-Петербург
2020

**Цель работы.**

Ознакомиться с понятием иерархического списка. Реализовать иерархический список для решения задания на языке программирования C++.

**Задание.**

Вариант 7.

Удалить из иерархического списка все вхождения заданного элемента(атома) x.

**Описание алгоритм работы.**

При запуске программы, ей подается строка и символ, которые записываются в переменные std::string str и std::string strDel соответственно. Узнаем размер строки и записываем его в целочисленную переменную size. Если пользователь подал на вход не символ для удаления, а строку, то выделяем из неё первый символ, записываем символ в переменную char symbol. Проверяем строку на валидность и создаем массив символов размера size, в который переписываем все символы из строки str. Создаем объект класса List. И вызываем для него методы deleteAtom() и print(). А затем выделяем очищенную память.

Класс Node:

В нем есть два приватных поля Node* next - указатель на следующий элемент списка и std::variant<Node*, char> data - значение, лежащее в этом узле (либо символ, либо указатель на элемент списка). Реализован конструктор, который инициализирует поля класса, и деструктор, который удаляет объект класса.

Класс List:

В классе есть приватные методы createNode() и deleteNode(). А также публичные поле (Node* head) и методы (deleteAtom() и print()). Также реализованы конструктор (вызывает метод createNode()) и деструктор(вызывает метод deleteNode()).

Метод createNode() рекурсивно проходится по поданной ему строке с помощью индекса, в зависимости, от окружающих в строке элементов создает либо атом (в значении лежит символ), либо узел(в значении лежит указатель).

Метод deleteNode() рекурсивно проходит по списку, если элемент списка - узел, то очищает память списка, на который он указывает в данных, вызывает метод для следующего и удаляет текущий.

Метод deleteAtom() рекурсивно проходит по списку. В нем создаем флаг, который будет нам показывать, было ли удаление или не было. Если элемент - узел и в его данных тоже лежит узел, то вызываем метод ещё раз, но уже для указателя, который лежит в данных. Если же это узел, но своими данным он указывает на атом, то мы проверяем, нужно ли нам удалять этот атом, если да, то создаем временную переменную, куда копируем адрес атома и меняем указатель узла на следующий за наших атомом элемент. И меняем значение флага. Далее проверяем для следующего. Если он узел, то вызываем для него наш метод, если же он атом, то проверяем, подходит ли он под наше условие. Если да, то удаляем, как и в первом случае, только теперь пеняем указатель нынешнего элемента на следующий элемент за следующим. Проверяем флаг, если он изменился, то вызываем наш метод ещё раз для текущего элемента.

Метод print() так же рекурсивно проходимся по списку. Если текущий элемент - узел, то выводим открывающуюся скобочку и вызываем для элемента, который лежит в данных. Делаем это до тех пор, пока вложенный список не закончится, тогда выводим закрывающуюся скобочку. Если элемент - атом, то выводим символ, который в нем лежит. Если указатель на следующий существует, то вызываем метод для него.

Разработанный программный код см. в приложении А.

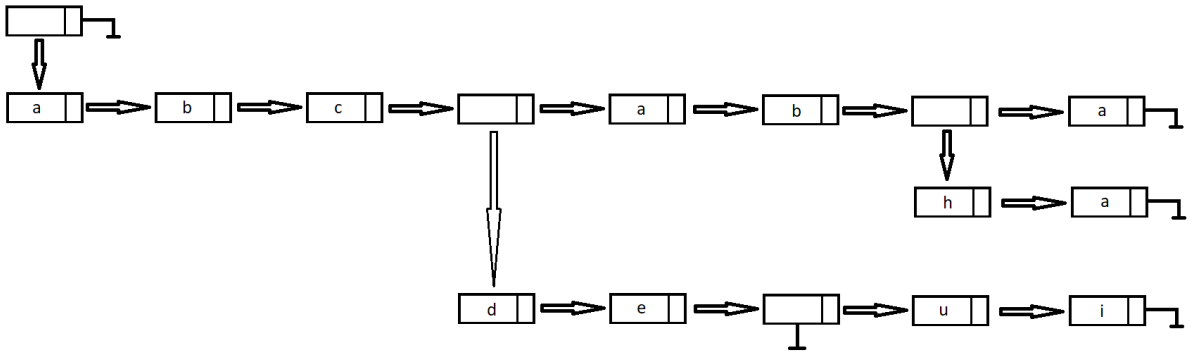
**Формат входных и выходных данных.**

На вход программе подается строка из символов и круглых скобок. А также символ, атом со значением которого мы хотим удалить.

Программа выводит иерархический список без всех вхождений поданного ранее элемента.

Ниже представлена визуализация одного из тестируемых иерархических списков.

**(abc(de()ui)ab(ha)a)**




**Описание основных структур данных и функций.**

- Класс Node - создание узлов и атомов иерархического списка.
- Класс List – создание иерархического списка
    a. Метод print() - вывод иерархического списка
    b. Метод createNode() - создание иерархического списка
    c. Метод deleteNode() - очищение памяти после иерархического списка
    d. Метод deleteAtom() - удаление заданного атома

**Тестирование.**

Для запуска тестирования в консоли прописываем команду make. Тестирование проводится по средством скрипта bash-скрипта. Он подает программе на вход определенные строки, к которым у неё уже есть ответ. Создает файл, в который записывает результат работы программы. Сравнивает строку-результат с данной строкой. Выводит в консоль входные данные, исход

проверки: incorrect!, если результат неверный, и correct, если результаты совпали, а также ожидаемы и реальные результаты Очищает созданные файлы. Результаты тестирования см. в приложении Б.

**Выводы.**

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

Была реализована программа, создающая иерархический список и удаляющая все вхождения заданного атома.

Использование иерархических списков не оправдано, так как, из-за их структуры, приходится тратить много ресурсов на хранение и обработку.

## ПРИЛОЖЕНИЕ А
## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab2.cpp

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

int main(int argc, char *argv[])
{
    std::string str;
    std::string strDel;
    char symbol;
    if(argc<2){
        std::cin >> str;
        std::cin >> strDel;
    }
    int size = str.size();
    if(str[0] != '('){
        std::cerr<<"The first symbol is not correct"<<'\n';
        return 0;
    }
    int counter = 1;
    for(int i = 1; i<size; i++){
        if(str[i]=='('){
            counter+=1;
        }
        if(str[i]==')'){
            counter-=1;
        }
    }
    if(counter){
```

```
            std::cerr<<"Wronq numder of brackets\n";
            return 0;
        }
        if(strDel.size()>1){
            std::cout<<"You entered a strind. I wiil delete only
first symbol"<<'\n';
        }
        symbol = strDel[0];
        char* arr = new char[size];
        for (int i = 0; i < size; i++) {
            arr[i] = str[i];
        }

        List lst(arr, size);
        lst.deleteAtom(lst.head, symbol);
        lst.print(lst.head);

        delete[] arr;
    }
```

Название файла: list.cpp

```
    #include "list.h"

    List::List(char* arrOfBukoff, int sizeArr) {
        head = createNode(arrOfBukoff, sizeArr, 0);
    }

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

    void List::print(Node* cur) {
        if (std::holds_alternative<Node*>(cur->data)) {
            std::cout << '(';
```

```
        if (std::get<Node*>(cur->data) != nullptr) {
            print(std::get<Node*>(cur->data));
        }
        std::cout << ')';
    }
    else {
        std::cout << std::get<char>(cur->data);
    }
    if (cur->next != nullptr) {
        print(cur->next);
    }
}

Node* List::createNode(char* arrOfBukoff, int sizeArr, int
index) {
    if (index < sizeArr - 1) {
        if (arrOfBukoff[index] == '(') {
            int i, count = 1;
            for (i = index + 1; i < sizeArr; i++) {
                if (arrOfBukoff[i] == '(') {
                    count += 1;
                }
                if (arrOfBukoff[i] == ')') {
                    count -= 1;
                }
                if (count == 0) {
                    break;
                }
            }
            if (i == sizeArr - 1 || arrOfBukoff[i + 1] == ')')
{
                Node*    uzel    =    new    Node(nullptr,
createNode(arrOfBukoff, sizeArr, index + 1));
```

```
                return uzel;
            }
            else {
                Node* uzel = new Node(createNode(arrOfBukoff,
sizeArr, i + 1), createNode(arrOfBukoff, sizeArr, index + 1));
                return uzel;
            }
        }
        else if (arrOfBukoff[index] == ')') {
            return nullptr;
        }
        else {
            Node*  atom  =  new  Node(createNode(arrOfBukoff,
sizeArr, index + 1), arrOfBukoff[index]);
            return atom;
        }
    }
    else {
        return nullptr;
    }
}

void List::deleteAtom(Node* cur, char toDelete) {
    if (cur != nullptr) {
        bool flag = false;
        if (std::holds_alternative<Node*>(cur->data)) {
            if (std::get<Node*>(cur->data) != nullptr) {
                if
(std::holds_alternative<Node*>(std::get<Node*>(cur->data)->d
ata)) {
                    deleteAtom(std::get<Node*>(cur->data),
toDelete);
                }
```

```
                else {
                    if
(std::get<char>(std::get<Node*>(cur->data)->data)         ==
toDelete) {
                        Node*             temp             =
std::get<Node*>(cur->data);
                        cur->data                          =
std::get<Node*>(cur->data)->next;
                        delete temp;
                        flag = true;
                    }
                    else {

deleteAtom(std::get<Node*>(cur->data), toDelete);
                    }
                }
            }
        }
        if (cur->next != nullptr) {
            if
(std::holds_alternative<Node*>(cur->next->data)) {
                deleteAtom(cur->next, toDelete);
            }
            else {
                if    (std::get<char>(cur->next->data)    ==
toDelete) {
                    Node* temp = cur->next;
                    cur->next = cur->next->next;
                    delete temp;
                    flag = true;
                }
                else {
                    deleteAtom(cur->next, toDelete);
```

```
                }
            }
        }
        if (flag) {
            deleteAtom(cur, toDelete);
        }
    }
}

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

Название файла: list.h

```
#ifndef LIST_H
#define LIST_H
#include <iostream>
#include <variant>
#include "node.h"
class List
{
    Node* createNode(char*, int, int);
    void deleteNode(Node*);

public:
    Node* head;
    List(char*, int);
    ~List();
```

```
    void print(Node*);
    void deleteAtom(Node*, char);

};
#endif
```

Название файла: node.cpp

```
#include "node.h"

Node::Node(Node* next, std::variant<Node*, char> data) :
next(next), data(data) {
}
Node::~Node() {
}
```

Название файла: node.h

```
#ifndef NODE_H
#define NODE_H
#include <iostream>
#include <variant>

class Node
{
    friend class List;

    Node* next;

    std::variant<Node*, char> data;
public:
    Node(Node*, std::variant<Node*, char>);
    ~Node();

};
#endif
```

Название файла: Makefile

```
all: lab2.o node.o list.o
    g++ - std = c++17 lab2.o node.o list.o - o lab2

lab2.o : Source / lab2.cpp Source / list.h
    g++ - std = c++17 - c Source / lab2.cpp

node.o : Source / node.cpp Source / node.h
    g++ - std = c++17 - c Source / node.cpp

list.o : Source / list.cpp Source / list.h Source / node.h
    g++ - std = c++17 - c Source / list.cpp

clean :
    rm - rf * .o
run_tests : lab2
    . / tests_script
```

# ПРИЛОЖЕНИЕ Б
# ТЕСТИРОВАНИЕ

Результаты тестирования представлены в таблице Б.1

Таблица Б.1 — Результаты тестирования

| № п/п | Входные данные | Выходные данные | Результат проверки |
|---|---|---|---|
| 1. | (a)<br>a | () | correct |
| 2. | (a(a(a(a(a)a)a)a)a)<br>a | (((((()))))) | correct |
| 3. | (ghja)<br>a | (ghj) | correct |
| 4. | (a(Fyhxsjx)()()(f)HJks)<br>f | (a(Fyhxsjx)()()()HJks) | correct |
| 5. | (sankkxnlsajbchasjklk)<br>p | (sankkxnlsajbchasjklk) | correct |
| 6. | ()<br>t | () | correct |
| 7. | (jhbxakj(jhsxkaskj(jnsslakml(idhcnksof)jcd)jncwke))<br>n | (jhbxakj(jhsxkaskj(jsslakml(idhcksof)jcd)jcwke)) | correct |
| 8. | (abc(de()ui)ab(ha)a)<br>a | (bc(de()ui)b(h)) | correct |