

**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

Кузнецов Р.В.

Преподаватель

Филатов А.Ю.

Санкт-Петербург

2020

## **Цель работы.**

Ознакомиться с иерархическими списками, научиться реализовывать их на языке C++.

## **Задание.**

Вариант 8.

Заменить в иерархическом списке все вхождения заданного элемента (атома) *x* на заданный элемент (атом) *y*;

## **Описание работы алгоритма.**

На вход программа запрашивает у пользователя иерархический список, состоящий из узлов и атомов, и два атома, разделенные пробелом, где вместо второго атома можно ввести иерархический список. Узлы должны быть представлены в виде открывающихся скобок, атомы в виде символов, не являющимися круглыми скобками. Закрывающая скобка означает конец ветки от последнего узла. Результатом работы программы является выведенный в таком же формате список, в котором все вхождения первого заданного атома заменены на второй введенный атом/иерархический список.

Структурная единица иерархического списка представлена в виде структуры *Node*. Она состоит из двух полей: указателя на следующий элемент списка и значения, которое может быть либо указателем на структуру *Node*, либо символом, что реализовано с помощью гетерогенного контейнера *std::variant*.

В процессе работы программа сначала считывает данные из *stdin*, затем генерирует 1-2 иерархических списка (в зависимости от типа изменяющего элемента) с помощью функции *getList()*, затем изменяет искомый элемент в списке с помощью функции *changeAllEntries()*, и после этого выводит получившийся иерархический список в его строковом представлении, полученном с помощью функции *getRepr()*.

Функция *getList()* принимает по ссылке указатель на ноду, в которую нужно записать список, а также константные итераторы на начало и конец строки, в

которой записан иерархический список. Функция работает итеративно: сначала создается вектор из уровней глубины атомов, в который записываются адреса нод, затем строка посимвольно обрабатывается: если символ, на который указывает итератор, равен открывающейся скобке, создается и подключается новый узел, глубина увеличивается. Если символ является закрывающей скобкой, глубина вложенности уменьшается. Иначе символ является значением атома. В данном случае создается атом с данным значением и подключается к списку. Подключение происходит с помощью лямбда функции *connect()*, принимающей константный указатель на ноду и захватывающей по ссылке вектор уровней и текущую глубину. Также функция следит за скобочным балансом, и в случае его нарушения, завершает работу программы, сообщая об ошибке.

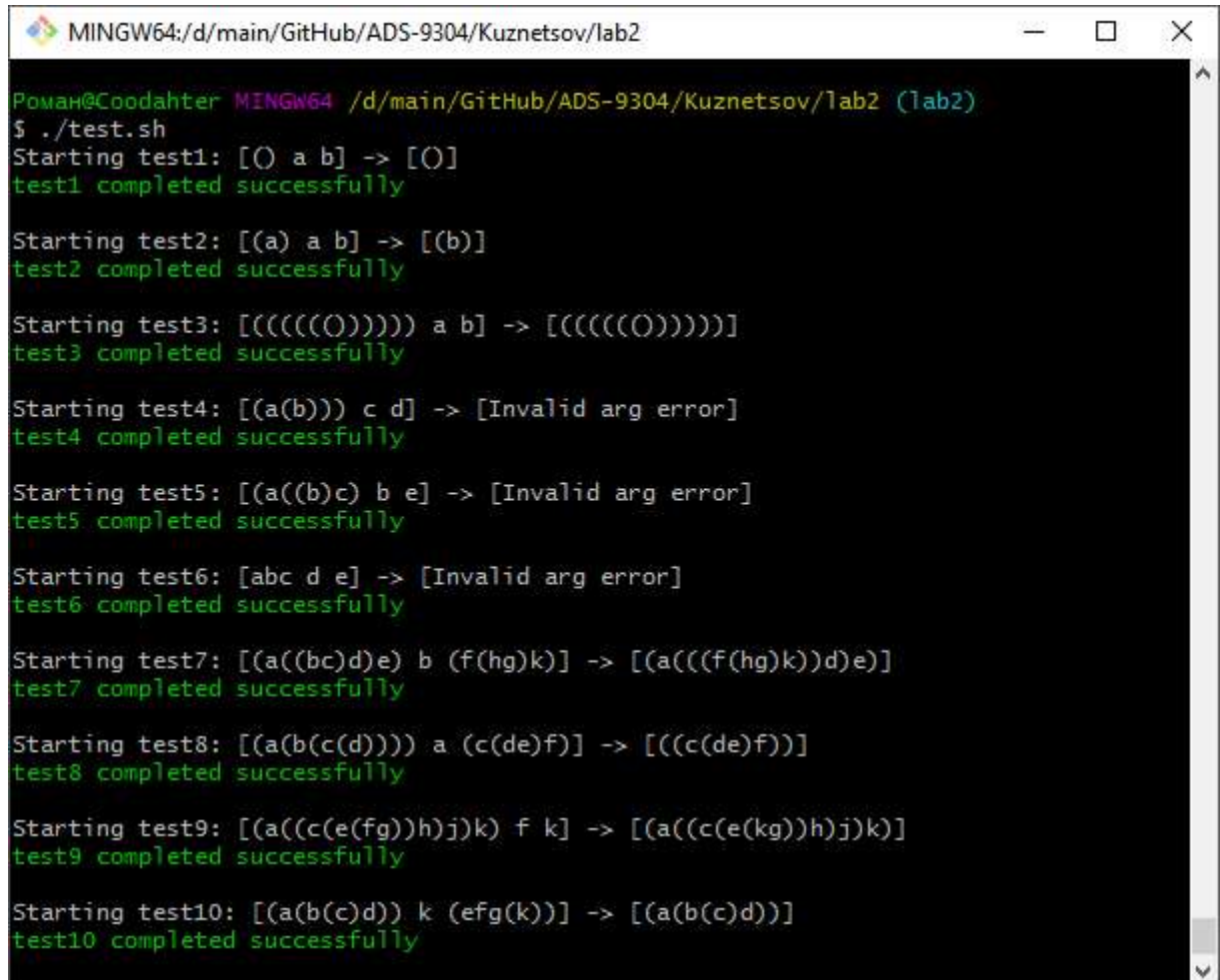
Функция *getRepr()* принимает по ссылке строку, в которую записывается результат, и константный указатель на список. Для удобства создается *std::stringstream*, в который записывается результат прохождения по списку. Алгоритм обхода списка аналогичен поиску в глубину, реализован с помощью рекурсивной лямбда функции *dfs()*, принимающей константный указатель на список и захватывающей строковый поток по ссылке. Полученный строковый поток преобразуется в строку, которая и является результатом работы функции.

Функция *changeAllEntries()* принимает по ссылке указатель на ноду, заменяемый атом и заменяющий атом/иерархический список, в зависимости от принятого значения. Иерархический список обходится по алгоритму, аналогичному используемому в функции *getRepr()*, проверяя встречные атомы на равенство заменяемого. В случае соответствия происходит замена.

### **Тестирование.**

Для тестирования программы был написан python скрипт и 2 bash скрипта. При запуске *test.sh* сначала компилируется основная программа, затем запускается скрипт *resgen.sh*, выводящий результаты работы программы в 10 файлов, по 1 на каждый соответствующий тест, затем запускается python скрипт,

сверяющий результат работы программы со списком строк ans. При каждой итерации проверяющего цикла выводится информация о предстоящем тесте, а на следующей строчке результат проверки, подсвеченный соответствующим цветом. Пример вывода можно увидеть на картинке 1.



```
MINGW64:/d/main/GitHub/ADS-9304/Kuznetsov/lab2
Роман@Coodahter MINGW64 /d/main/GitHub/ADS-9304/Kuznetsov/lab2 (lab2)
$ ./test.sh
Starting test1: [(O a b] -> [O]
test1 completed successfully

Starting test2: [(a) a b] -> [(b)]
test2 completed successfully

Starting test3: [((((((O)))))) a b] -> [((((((O))))))]
test3 completed successfully

Starting test4: [(a(b))) c d] -> [Invalid arg error]
test4 completed successfully

Starting test5: [(a((b)c) b e] -> [Invalid arg error]
test5 completed successfully

Starting test6: [abc d e] -> [Invalid arg error]
test6 completed successfully

Starting test7: [(a((bc)d)e) b (f(hg)k)] -> [(a(((f(hg)k))d)e)]
test7 completed successfully

Starting test8: [(a(b(c(d)))) a (c(de)f)] -> [(((c(de)f))]
test8 completed successfully

Starting test9: [(a((c(e(fg))h)j)k) f k] -> [(a(((c(e(kg))h)j)k)]
test9 completed successfully

Starting test10: [(a(b(c)d)) k (efg(k))] -> [(a(b(c)d))]
test10 completed successfully
```

Картинка 1 – Результат работы тестирующей программы.

В процессе тестирования были рассмотрены все возможные типы входных данных, так как если первый аргумент не является иерархическим списком, программа корректно завершит работу, сообщив об этом пользователю. Если скобочный баланс этой строки валиден, она является иерархическим списком вне зависимости от содержания, так как данные атомов хранятся в типе char, который может корректно работать с любым символом. Второй аргумент, заменяемый атом, также может являться любым символом. Если атом с таким значением присутствует в списке, то значение корректно, иначе ничего не будет заменено и программа продолжит работу. Третий аргумент, заменяющий элемент, может

являться либо атомом, либо иерархическим списком. В первом случае его длина должна быть равна 1, и это должна быть не открывающаяся скобка, иначе это иерархический список, над которым производятся все те же проверки, как и над первым аргументом. Результаты тестирования начинаются на таблице Б.1 и продолжаются на таблице Б.2 в приложении.

Таблица Б.1 – Результат тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	() a b	()	Пустой список
2.	(a) a b	(b)	Минимальный список
3.	(((((O)))))) a b	(((((O))))))	Список без атомов
4.	(a(b))) c d	ERROR: given string list is invalid (there are more ')' than '(' ) string value = [(a(b)))]	Список некорректен, скобочный баланс отрицателен

### Выводы.

В процессе выполнения работы было проведено ознакомление с иерархическими списками, также были получены практические навыки по реализации их на языке C++, навыки работы с гетерогенным контейнером `std::variant`.

Была реализована программа, заменяющая все вхождения заданного атома на вводимый элемент. Также была реализована тестирующая программа на языке Python для проверки корректности выходных данных и пара *bash*-скриптов, упрощающих использование программы.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <variant>
#include <string>
#include <iterator>
#include <memory>
#include <vector>
#include <sstream>

struct Node {
    std::shared_ptr<Node> next {nullptr};
    std::variant<std::shared_ptr<Node>, char> val;
};
using NodePtr = std::shared_ptr<Node>;
#define NewNode std::make_shared<Node>

void getList(NodePtr &node, const std::string::iterator begin, const
std::string::iterator end) {
    unsigned v_capacity = 10;
    std::vector<NodePtr> levels(v_capacity, nullptr);
    int depth = 1;
    auto it = begin;
    if (*it != '('){
        std::cout << "ERROR: given string list is invalid (it
has to start with '(' )\n" \
                                                                "string value = ["
<< std::move(std::string(begin, end)) << "];
        exit(EXIT_FAILURE);
    }
    it++;
    levels[0] = node;

    auto connect = [&levels, &depth](const NodePtr &cur) {
        if (levels[depth]) //if not first at this level
            levels[depth]->next = cur;
        else //if first at this level
            levels[depth - 1]->val = cur;
        levels[depth] = cur;
    };

    while (it != end) {
        if(depth<=0){
            std::cout << "ERROR: given string list is
invalid (there are more ')' than '(' )\n" \
                                                                "string
value = [" << std::move(std::string(begin, end)) << "];
            exit(EXIT_FAILURE);
        }
        switch (*it) {
            case '(':
                {
                    if (depth + 1 == v_capacity) {
                        v_capacity <=<= 1;
```

```

                                levels.resize(v_capacity,
nullptr);

                                }
                                NodePtr cur = NewNode();
                                connect(cur);
                                depth++;
                                }
                                break;
                                case ')':
                                depth--;
                                break;
                                default:
                                {
                                    NodePtr cur = NewNode();
                                    cur->val = *it;
                                    connect(cur);
                                }
                                } //switch
                                it++;
                                } //while
                                if(depth){
                                    std::cout << "ERROR: given string list is invalid
(there are more '(' than ')'\n" \
                                                "string value = ["
<< std::move(std::string(begin, end)) << "];
                                    exit(EXIT_FAILURE);
                                }
                                }

                                void getRepr(std::string &dest, const NodePtr& list) {
                                    std::stringstream sstr;
                                    auto dfs = [&sstr](const NodePtr& list, auto && dfs) {
                                        if (!list){
                                            sstr << ')';
                                            return;
                                        }
                                        if (std::holds_alternative<NodePtr>(list->val)){
                                            sstr << '(';
                                            dfs(std::get<NodePtr>(list->val), dfs);
                                        }
                                        else
                                            sstr << std::get<char>(list->val);
                                        dfs(list->next, dfs);
                                    };
                                    dfs(list, dfs);
                                    dest = std::move(sstr.str());
                                    dest.pop_back();
                                }

                                void changeAllEntries(NodePtr &node, char a, const std::variant<char,
NodePtr>& b){
                                    if (node) {
                                        if (std::holds_alternative<NodePtr>(node->val))
                                            changeAllEntries(std::get<NodePtr>(node->val), a, b);
                                        else if (std::get<char>(node->val) == a)
                                            if(std::holds_alternative<NodePtr>(b))
                                                node = std::get<NodePtr>(b);

```

```

        else
            node->val = std::get<char>(b);
        changeAllEntries(node->next, a, b);
    }
}

int main() {
    std::string input, res, b_str;
    char a;
    std::cin>>input>>a>>b_str;
    NodePtr list = NewNode();
    getList(list, input.begin(), input.end());
    if(b_str.size() == 1 && b_str[0] != '(')
        changeAllEntries(list, a, b_str[0]);
    else {
        NodePtr b_list = NewNode();
        getList(b_list, b_str.begin(), b_str.end());
        changeAllEntries(list, a, b_list);
    }
    getRepr(res, list);
    std::cout<<res;
    return 0;
}

```



## Название файла: tests.py

```
ans = [
    "()", #1
    "(b)", #2
    "(((((((()))", #3
    "ERROR: given string list is invalid (there are more ')' than
'(' )\n\
    string value = [(a(b))]", #4
    "ERROR: given string list is invalid (there are more '(' than
')' )\n\
    string value = [(a((b)c)]", #5
    "ERROR: given string list is invalid (it has to start with
'(' )\n\
    string value = [abc]", #6
    "(a(((f(hg)k))d)e)", #7
    "(c(de)f)", #8
    "(a((k(e(kg))k)j)k)", #9
    "(a(b(c)d))", #10
]

for i in range(1, 11):
    f = open(f"Tests/test{i}.txt", 'r')
    inp = f.read()
    trueRes = ans[i-1]
    print(f"Starting test{i}:", f"[{inp}] ->", f"[{trueRes}]" if
trueRes[0:5] != "ERROR" else f"[Invalid arg error]")
    f.close()
    f = open(f"Tests/testres{i}.txt", 'r')
    myRes = f.read()
    f.close()
    if myRes == trueRes:
        print(f"\033[32mtest{i} completed successfully\033[m\n")
    else:
        print(f"\033[31mtest{i} failed:\033[m", \
            f"your res = [{myRes}]", \
            f"true res = [{trueRes}]", sep='\n', end='\n\n')
```

## ПРИЛОЖЕНИЕ Б

### ТЕСТИРОВАНИЕ

Таблица Б.2 – Результат тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
5.	(a((b)c) b e	ERROR: given string list is invalid (there are more '(' than ')') string value = [(a((b)c)]	Список некорректен, скобочный баланс положителен
6.	abc d e	ERROR: given string list is invalid (it has to start with '(') string value = [abc]	Ввод некорректен: abc не является иерархическим списком.
7.	(a((bc)d)e) b (f(hg)k)	(a(((f(hg)k))d)e)	Вместо заменяющего атома был передан список, замена b на данный список приводит к удалению атома c, так как на b, ссылающийся на него, заменен.
8.	(a(b(c(d)))) a (c(de)f)	((c(de)f))	Коренной атом a заменен на иерархический список, на внутреннюю часть «(b(c(d)))» уже ничего не указывает
9.	(a((c(e(cg))c)j)c) c k	(a((k(e(kg))k)j)k)	Несколько атомов c заменены атомом k
10.	(a(b(c)d)) k (efg(k))	(a(b(c)d))	Замена несуществующего

			<p>элемента на иерархический список не изменила иерархический список.</p>
--	--	--	---