

**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

\_\_\_\_\_

Ковалёв П. Д.

Преподаватель

\_\_\_\_\_

Филатов А. Ю.

Санкт-Петербург

2020

### **Цель работы.**

Изучить иерархические списки, написать программу, которая взаимодействует с ними.

### **Задание.**

#### **Вариант 11**

Сформировать линейный список атомов исходного иерархического списка таким образом, что скобочная запись полученного линейного списка будет совпадать с сокращённой скобочной записью исходного иерархического списка после устранения всех внутренних скобок;

### **Выполнение работы.**

Сначала были написаны структуры *Elem* и *Node*, а после класс *linkedList*. У данного класса есть поля: *std::string argLine*, *Node\* head*, *Elem\* uniquePtr*, *int length*, *int error*, *int counter*. Также у класса есть методы: конструктор *linkedList(std::string& st)*, метод, проводящий проверку концов полученной строки *bool listEndsChecker(const std::string& s) const*, метод, создающий иерархический список *void addNode(Node\*\* ptr, int& iter, const std::string& s, int& err, int& deep)*, метод, помещающий все атомы списка в строку *void getAtoms(Node\* ptr, std::string& argLine)*, метод, ведущий подсчет числа атомов списка *void counting(std::string& s, int& num)*, метод, создающий линейный список *Elem\* createLinearList(std::string& args, int& count)*, метод, создающий узлы линейного списка *Elem\* addLinElem(std::string& s)*, методы очистки памяти списков *void deleter(Node\* tmp)* и *void listDeleter(Elem\* tmp)*, метод вывода линейного списка на экран *void printer(Elem\* ptr)*. Структура *Node* является узлом иерархического списка, а структура *Elem* — узлом линейного списка.

Метод *bool listEndsChecker(const std::string& s) const* проверяет первый и последний символы входной строки. Если эти символы не скобки, то метод вернет *false* и работа программы будет прекращена.

Метод *void addNode(Node\*\* ptr, int& iter, const std::string& s, int& err, int& deep)* создает иерархический список. В нем происходит анализ полученной строки и в случае, если ее элементы — буквы латинского алфавита, будет создан иерархический список, иначе — нет. Так же там происходит проверка на корректность строки. Анализ строки происходит посимвольно, причем если в какой-то момент пользователь передал не одну букву, а целое слово — оно будет считано и записано в соответствующий ему узел.

Метод *void getAtoms(Node\* ptr, std::string& argLine)* проходит по всему иерархическому списку и копирует его атомы в строку *argLine*, разделяя их пробелом.

Метод *void counting(std::string& s, int& num)* принимает строку с атомами иерархического списка и вычисляет их количество.

Метод *Elem\* createLinearList(std::string& args, int& count)* создает линейный список, и наполняет его атомами иерархического списка, которые он выделяет из строки с атомами *args*. В результате функция возвращает указатель на голову списка.

Метод *void printer(Elem\* ptr)* проходится по линейному списку и выводит на экран его элементы через пробел.

Метод *void deleter(Node\* tmp)* используется для очистки памяти, занимаемой иерархическим списком. Он вызывает сам себя до тех пор, пока не дойдет до конца списка, а далее удаляет узел и возвращается на шаг назад и удаление узла повторяется.

Метод *void listDeleter(Elem\* tmp)* используется для очистки памяти, занимаемой линейным списком. Он вызывает сам себя до тех пор, пока не

дойдет до конца списка, а далее удаляет узел и возвращается на шаг назад и удаление узла повторяется.

### **Тестирование.**

Запуск программы начинается с запуска команды *make* в терминале, что приведет к созданию исполняемого файла *lab2*. Запуск программы начинается с ввода команды *./lab2* в терминале в директории *lab2*. Тестирование же проводится с помощью скрипта *tester.py*, который запускается командой *python3 tester.py* в командной строке в директории *lab1*. В файле *test1.txt* лежат входные данные, которые обязательно должны удовлетворять условию задачи, а в файле *test2.txt* лежат входные данные, которые обязательно должны не удовлетворять условию задачи. Таким образом, происходит проверка того, сможет ли программа корректно обработать правильные и неправильные входные данные. Также скрипт в результате тестирования выводит входные данные, и то, что вывела программа. Подавать на вход программе нужно последовательность символов и слов в виде иерархического списка, символы со словами должны быть разделены между собой пробелами, а также отделены пробелами от скобок. Допускается возможность ввода пустого списка.

Результаты тестирования представлены в приложении Б.

### **Выводы.**

Ознакомились с иерархическими списками и особенностями их программирования на языке программирования C++, используя знания о рекурсии, а также знания об объектно-ориентированном программировании.

Реализовали программу, которая считывает входную строку в виде иерархического списка и выводит саму строку в виде линейного списка. Использование иерархического списка в данном случае удобно, т. к. входная

последовательность символов представляет собой список, внутри которого могут быть другие списки.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <cstdlib>

struct Node{
    bool isDown;
    Node* next;
    Node* down;
    std::string val;
};

struct Elem{

    Elem* next;
    std::string data;

};

class linkedList{

    std::string argLine;
    Node* head;
    Elem* uniquePtr;
    int length;
    int error;
    int counter;

public:

    linkedList(std::string& st){

        this->head = nullptr;
        this->uniquePtr = nullptr;
        this->argLine = "";
        this->length = st.length();
        this->counter = 0;

        if(listEndsChecker(st)){
            int iter = 1, counter = 0;
            this->error = 0;
            addNode(&this->head, iter, st, error);
            if(!error) {
                getAtoms(this->head, this->argLine);
                if(this->argLine != "") {
                    counting(this->argLine, this->counter);

                    this->uniquePtr = createLinearList(argLine,
this->counter);

                    printer(uniquePtr);
```

```

        listDeleter(this->uniquePtr);

        deleter(this->head);
    }else{
        std::cout << "Correct!\n";
    }
}
}
}

bool listEndsChecker(const std::string& s) const{
    if(s[0] == '(' && s[length - 1] == ')') {
        return true;
    }
    else{
        std::cout << "Wrong expression!" << '\n';
        return false;
    }
}

void addNode(Node** ptr, int& iter, const std::string& s,
int& err) {
    if(iter < this->length) {
        if (s[iter] != ')') {
            if (s[iter] == ' ') {
                iter = iter + 1;
            }
            if (s[iter] != '(') {
                if (s[iter] == ' ') {
                    iter = iter + 1;
                }
                if (isalpha(s[iter])){

                    std::string el = "";
                    int end = 0;
                    while(isalpha(s[iter + end])){
                        el = el + s[iter + end];
                        end++;
                    }

                    Node *tmp = new Node;
                    tmp->next = nullptr;
                    tmp->val = el;
                    tmp->isDown = false;
                    tmp->down = nullptr;

                    *ptr = tmp;
                    iter = iter + end;
                    addNode(&((*ptr)->next), iter, s, err);
                }
            }
        }
    }
}

```





```

    }
    num = num - spaceCounter;
    num++;
}

Elem* createLinearList(std::string& args, int& count){
    std::string el = "";
    int it = 0;
    Elem* head = nullptr;
    Elem* tail = new Elem;

    while(args[it] != ' '){
        el = el + args[it];
        it++;
    }
    it++;
    tail->data = el;
    el.clear();
    tail->next = nullptr;
    head = tail;

    for(int i = 0; i < count - 1; i++){
        while(args[it] != ' '){
            el = el + args[it];
            it++;
        }
        it++;
        tail->next = addLinElem(el);
        el.clear();
        tail = tail->next;
    }

    return head;
}

Elem* addLinElem(std::string& s){
    Elem* tmp = new Elem;
    tmp->data = s;
    tmp->next = nullptr;
    return tmp;
}

void printer(Elem* ptr){
    while(ptr){
        std::cout << ptr->data << ' ';
        ptr = ptr->next;
    }
    std::cout << '\n';
    std::cout << "Correct!\n";
}

void deleter(Node* tmp){
    if(tmp->next){
        deleter(tmp->next);
    }
    if(tmp->isDown){

```

```

        deleter(tmp->down);
    }
    delete tmp;
}

void listDeleter(Elem* tmp){
    if(tmp->next){
        listDeleter(tmp->next);
    }
    delete tmp;
}

};

int main(int argc, char* argv[]) {
    if(argc == 1){
        std::cout << "Wrong expression!" << '\n';
        return 0;
    }
    std::string s = "";
    int i = 1;
    while(argv[i]){
        s = s + std::string(argv[i]) + ' ';
        i++;
    }
    s.erase(s.length() - 1);
    linkedList list(s);
    return 0;
}

```

## ПРИЛОЖЕНИЕ Б

### ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	( a )	a Correct!	Последовательность правильная
2.	( a b c )	a b c Correct!	Последовательность правильная
3.	( )	Correct!	Последовательность правильная
4.	( a b c ( ) )	a b c Correct!	Последовательность правильная
5.	( a b c d ( e f ) )	a b c d e f Correct!	Последовательность правильная
6.	agjpsdjfh	Wrong expression!	Последовательность неправильная
7.	a b c d ( e f ) )	Wrong expression!	Последовательность неправильная
8.	( a b c ( d e f ) g h i )	a b c d e f g h i Correct!	Последовательность правильная
9.	( a b c ( d e f ( g h i ) ) )	a b c d e f g h i Correct!	Последовательность правильная
10.		Wrong expression!	Последовательность неправильная (введена пустая строка).