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: Рекурсивная обработка иерархических списков

Студент гр. 9384 _____ Соседков К.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Решить поставленную задачу с использованием базовых функций рекурсивной обработки списков.

Задание. (Вариант № 1)

Проверить иерархический список на наличие в нем заданного элемента (атома) x.

Выполнение работы.

Для выполнения работы был разработан класс *List* с методами *createNode, createList* и *contains.*

Метод *createNode* создает новый элемент в списке, который содержит некоторые данные типа *char* и указатели на следующий и предыдущий элемент в списке.

Рекурсивный метод *createList* создает список, который в качестве элементов может иметь как обычные элементы типа *char* так и такие же списки.

Метод *contains* проверяет список на наличие в нем заданного элемента.

Так же была написана вспомогательная функция *isValid* проверяющая список на корректность.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.		String is empty	Ошибка Ввод пустой строки
2.	(a (s w) s s (t t)) w	Character in the list	Все в порядке
3.	(((w)))))	Extra character) at pos 6	Ошибка Много лишних скобок

Выводы.

В ходе выполнения лабораторной работы был реализован иерархический список и рекурсивные методы для работы с ним(создание списка, удаление списка и поиск элементов в списке).

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <algorithm> //std::remove
#include "List.h"

using namespace std;

std::string input();
bool isValid(std::string listString);
string readListFromFile();
string readListFromTerminal();



int main() {
    List* list;
    std::string listString = input();

    if(isValid(listString)) {
        list = new List(listString);

        cout << "Enter the character you want to find\n> ";
        char ch = getchar();

        if(list->contains(ch)) cout << "Character in the list\n";
        else cout << "Character is NOT in the list\n";
    }

    return 0;
}
```

```
std::string input() {
    cout << "Lab 2(1) -> проверить иерархический список на наличие в нем заданного элемента (атома) x;\n";
    cout << "1) Input from file ('file.txt' by default)\n";
    cout << "2) Terminal input\n";
    cout << "Input type (1, 2): \n> ";

    int type = 0;
    do {
        cin >> type;
        if (cin.fail()) {
            cout << "Please enter an integer (1, 2)\n> ";
            cin.clear();
            cin.ignore();
        }
    } while(type != 1 && type != 2);

    std::string str;

    if(type == 1) str = readListFromFile();
    else if(type == 2) str = readListFromTerminal();

    str.erase(remove(str.begin(), str.end(), ' '), str.end());

    return str;
}

string readListFromFile() {
    string line;
    cin.ignore();
    ifstream infile ("file.txt");
    std::getline(infile, line);
    return line;
```

```
}

string readListFromTerminal() {
   cout << "> ";
   string line;
   cin.ignore();
   getline(std::cin, line);
   return line;
}

bool isValid(std::string listString) {
   if(listString.empty()) {
      cout << "String is empty\n";
      return false;
   }

   int openBracketCount = 0;
   bool flag = false;
   for(int i=0; i<listString.size(); i++) {
      if(listString[i] == '(') {
         flag = false;
         openBracketCount++;
      }
      else if(listString[i] == ')') {
         if(flag && openBracketCount-1 >= 0) {
            openBracketCount--;
            if(openBracketCount == 0 && i < listString.size()-1) {
                  cout << "Extra character " << listString[i] << " at pos " << i <<
endl;
               return false;
            }
         }
         else {
```

```
                    cout << "Unexpected character " << listString[i] << " at pos " << i
<< endl;
                    return false;
                }
            }
            else if(isalnum(listString[i])) flag = true;
            else {
                cout << "Unexpected symbol " << listString[i] << " at pos " << i <<
endl;
                return false;
            }
        }
        if(openBracketCount == 0) return true;
        std::cout << "Not enough ')' in string\n";
        return false;
    }
```

Название файла: List.h

```
#pragma once
#include <iostream>
using namespace std;

class List {
private:

    struct Node {
        Node() {};
        Node(char s): data(s) {};
        Node* next = nullptr;
        Node* prev = nullptr;;
        List* list = nullptr;
        char data;
    };
```

```
    Node* head = nullptr;
    Node* tail = nullptr;

public:
    List(std::string listString) {
        createList(listString, this);
    }

    ~List() {
        deleteList();
    }

    void deleteList(List* lst=nullptr) {
        if(lst == nullptr) lst = this;
        Node* tmp = lst->head;
        while(tmp) {
            if(tmp->list) {
                deleteList(tmp->list);
                delete tmp->list;
            }
            tmp = tmp->next;
            delete tmp->prev;
        }
    }

    Node* createNode(List* list, Node* newNode) {
        if(!list->head) {
            list->head = newNode;
            list->tail = list->head;
        }
        else {
            newNode->prev = tail;
            tail->next = newNode;
            tail = newNode;
```

```
    }
    return newNode;
}

void createList(std::string s, List* list) {
    char symbol = readNextChar(s);

    if(symbol == ' ') return;
    else if(symbol == '(') {
        Node* newNode = createNode(list, new Node());
        newNode->list = new List(s);
    }
    else if(isalnum(symbol)) {
        createNode(list, new Node(symbol));
    }
    else if(symbol == ')') {
        return;
    }
    createList(s, list);
}

bool contains(char atom, List* lst = nullptr) {
    if(lst == nullptr) lst = this;
    Node* tmp = lst->head;
    while(tmp) {
        if(tmp->list && contains(atom, tmp->list)) return true;
        else if(!tmp->list && tmp->data == atom) return true;
        tmp = tmp->next;
    }
    return false;
}

static char readNextChar(string s) {
    static int str_size = s.size();
```

```
        static int index = 0;
        static string last_string = s;
        if(last_string != s) {
            str_size = s.size();
            index = 0;
            last_string = s;
        }
        while(s[index] == ' ' && index < str_size) {
            index++;
        }
        if(index >= str_size) return ' ';
        index++;
        return s[index-1];
    }
};
```