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.

Студентка гр. 9304 _____ Рослова Л.С

Преподаватель _____ Филатов А.Ю

Санкт-Петербург

2020

Цель работы.

Изучить понятие иерархического списка. Реализовывать программу с использованием иерархического списка в языке C++.

Задание.

14) Обратить иерархический список на всех уровнях вложения; Например, для исходного списка (a(bc)d) результатом обращения будет список (d(cb)a).

Выполнение работы.

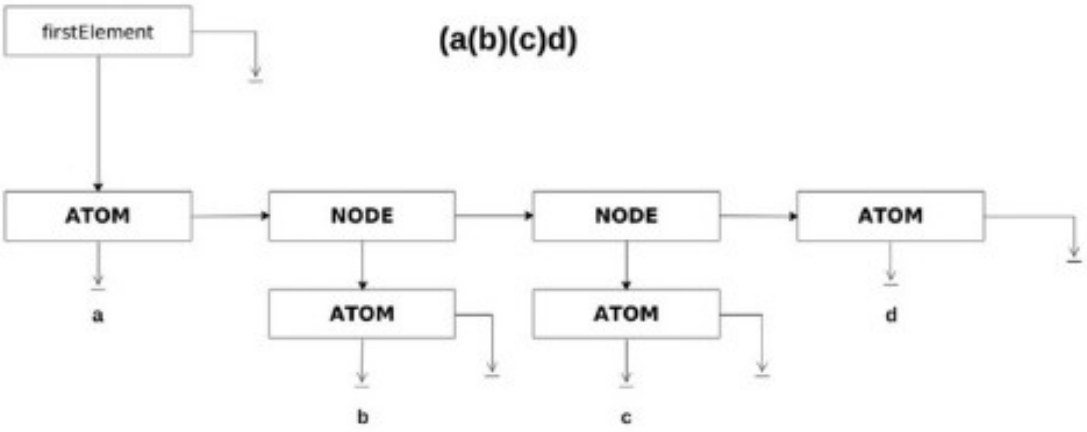
Программа принимает на вход строку со скобочным представлением иерархического списка и передает его в конструктор класса *List*. Конструктор с помощью приватного метода *checkValid* проверяет строку на корректность, в случае нахождение символа не относящегося к скобам и буквам — выбрасывает исключение с последующим описанием ошибки, и инициализирует первый элемент списка *nullptr*. Если структура корректна приватный метод *createList* производит рекурсивное построение списка с помощью лямбды. Каждый элемент списка представляет из себя объект класса *Node*, в котором хранится умный указатель на следующий объект и вариативная строка данных, которая может быть как типом char, так и умным указателем на следующий подуровень. Для чтения иерархического списка был реализован публичный метод *read*, который рекурсивно читает список, а так же создан метод *reverse* для обращения элементов.

Разработанный программный код см. в приложении А.

Формат входных и выходных данных.

На вход программе подается строка со структурным представлением иерархического списка.

Программа должна обратить элементы на каждом уровне.



Тестирование.

Для проведения тестирования был написан bash-скрипт ./script .Скрипт запускает программу где в качестве входных аргументов служат заранее подготовленные файлы, расположенные в папке ./Tests

Результаты тестирования см. в приложении Б.

Выводы.

Было изучено понятие иерархического списка. Реализована программа с использованием иерархического списка на языке C++, с использованием библиотеки *variant* .

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <memory>
#include <iostream>

#include "Node.h"
#include "List.h"



int main(int argc, char* argv[]){

    std::string structList{};
    getline(std::cin, structList);

    std::unique_ptr<List> myList(new List(structList));

    if(myList->getFirstElement()){
        myList->revers();
        myList->read();
    }

    return 0;
}
```

Название файла: List.h

```
#pragma once

#include <memory>



class Node;

class List{
    std::shared_ptr<Node> firstElement;
    void createList(std::string structList);
    void checkValid(std::string structList);

public:
    List(std::string structList);
    std::shared_ptr<Node> getFirstElement();
```

```
    void revers();
    void read();
};
```

Название файла: Node.h

```
#pragma once

#include <memory>
#include <variant>



class Node{
    using NodePtr = std::shared_ptr<Node>;
public:
    Node();
    Node(char newValue);

    NodePtr next;
    std::variant<NodePtr, char> value;
};
```

Название файла: Node.cpp

```
#include "Node.h"



Node::Node(){
    next = nullptr;
}


Node::Node(char newValue){
    value = newValue;;
}
```

Название файла: List.cpp

```
#include <iostream>
#include <vector>

#include "List.h"
#include "Node.h"
```

```
List::List(std::string structList){
    try{
        checkValid(structList);
        createList(structList);
    }catch(const char* strErr){
        std::cerr << strErr << std::endl;
        firstElement = nullptr;
    }
}



std::shared_ptr<Node> List::getFirstElement(){
    return firstElement;
}



void List::revers(){

    auto PR = [](std::shared_ptr<Node>& node, auto&& PR){

        std::shared_ptr<Node> tmp = node;
        std::vector<char> dataAtom {};

        if(!node){
            return;
        }

        while(1){
            if(tmp == nullptr){
                break;
            }
            if(std::holds_alternative<std::shared_ptr<Node>>(tmp->value))
{
                PR(std::get<std::shared_ptr<Node>>(tmp->value), PR);
            }else if(std::holds_alternative<char>(tmp->value)){
                dataAtom.emplace_back(std::get<char>(tmp->value));
            }
            tmp = tmp->next;
        }

        tmp = node;

        while(1){
            if(tmp == nullptr){
                break;
            }
            if(std::holds_alternative<char>(tmp->value)){
                tmp->value = dataAtom.back();
                dataAtom.pop_back();
            }
            tmp = tmp->next;
        }

    };
```

```
    PR(firstElement, PR);

}



void List::read(){

    size_t level = 0;

    auto PR = [&level](std::shared_ptr<Node>& node, auto&& PR){

        std::shared_ptr<Node> tmp = node;

        while(1){
            if(tmp == nullptr){
                if(level){
                    std::cout << ')';
                }
                return;
            }
            if(std::holds_alternative<std::shared_ptr<Node>>(tmp->value))
{
                std::cout << '(';
                level++;
                PR(std::get<std::shared_ptr<Node>>(tmp->value), PR);
                level--;
            }else if(std::holds_alternative<char>(tmp->value)){
                std::cout << std::get<char>(tmp->value);
            }
            tmp = tmp->next;
        }

    };

    PR(firstElement, PR);
    std::cout << '\n';
}



void List::createList(std::string structList){

    size_t iter = 0;
    size_t secondIter = 0;
    firstElement = std::make_shared<Node>();

             auto   PR   =   [&structList,   &secondIter](size_t   iter,
std::shared_ptr<Node>& node, auto&& PR){

        secondIter++;

        if(structList[iter] == '(' && structList[secondIter] == ')'){
            return;
        }

        if(structList[iter] == '(' && structList[secondIter] == '('){
```

```
            if(secondIter - iter == 1){
                node->value = std::make_shared<Node>();
                     PR(secondIter,  std::get<std::shared_ptr<Node>>(node-
>value), PR);
            }else{
                                                std::shared_ptr<Node>   tmp   =
std::get<std::shared_ptr<Node>>(node->value);
                while(tmp->next){
                    tmp = tmp->next;
                }
                tmp->next = std::make_shared<Node>();
                PR(secondIter, tmp->next, PR);
            }

        }

        if(structList[iter] == '(' && isalpha(structList[secondIter])){

            if(secondIter - iter == 1){
                                                                node->value    =
std::make_shared<Node>(structList[secondIter]);
            }else{
                                                std::shared_ptr<Node>   tmp   =
std::get<std::shared_ptr<Node>>(node->value);
                while(tmp->next){
                    tmp = tmp->next;
                }
                                                                  tmp->next    =
std::make_shared<Node>(structList[secondIter]);
            }

        }

        PR(iter, node, PR);
    };

    PR(iter, firstElement, PR);

}



void List::checkValid(std::string structList){

    size_t lB = 0;  // Левая открывающая скобка
    size_t rB = 0;  // Правая закрывающая
    size_t atm = 0; // Кол-во атомов

    for(size_t i = 0; i < structList.size(); i++){

        if(structList[i] == '('){
            lB++;
        }else if(structList[i] == ')'){
            rB++;
        }else if(isalpha(structList[i])){
            atm++;
        }else{
            throw "Unknown symbol!";
```

```
        }

        if(rB > lB){
            throw "Right brackets > Left braskets!";
        }

    }

    if(rB != lB){
        throw "Left braskets < Right brackets!";
    }
    if(structList[0] != '('){
        throw "First symbol != '('!";
    }
    if(structList[structList.size() - 1] != ')'){
        throw "Last symbol != ')'!";
    }

}
```

Название файла: script

```
#!/bin/bash

for n in {1..7}
do
   arg=$(cat Tests/test$n.txt)
   echo -e "\nTest $n:"
   echo "BinTree = $arg"
   ./lab2 < Tests/test$n.txt
done
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ ПРОГРАММЫ

Результаты тестирования представлены в табл.1

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	(avc(def))	(cva(fed))	Пример правильной работы 1
2.	(a(bcd)g(hed)k)	(k(dcb)g(deh)a)	Пример правильной работы 2
3.	(a(b(c(d(e(fgho))))))	(a(b(c(d(e(ohgf))))))	Пример правильной работы 3
4.	a(bcd)	First symbol != '('!	Неверное начало структуры
5.	(j))	Right brackets > Left braskets!	Закрывающих больше чем открывающих
6.	(123)	Unknown symbol!	Символы не являются буквами