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 _____ Паутова Ю.В.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с понятием иерархического списка, получить навыки программирования функций для работы с иерархическими списками. Реализовать иерархический список на языке программирования C++.

Задание.

Вариант 6

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

Описание алгоритма работы.

Программе с помощью аргументов командной строки подаются два файла: файл1, содержащий строку-список и элемент(фтом), и файл2, в который будет записан результат работы программы. Сначала проверяем были ли переданы программе файлы, если да, то проверяем, открылись ли они для считывания/записи. Если оба файла были открыты, тогда создаются: объект класса string: list, объект класса H_List: h_list и переменная типа char: elem. Строка-список записывается в list, после чего проверяется на корректность функцией isCorrect().

Если строка-список корректна, то вызывается метод класса H_List read_h_list(), который создает иерархический список с помощью метода read_node, который рекурсивно пробегается по строке-списку list и создает узел (в значении лежит указатель) или атом (в значении лежит значение).

Затем считывается элемент(атом) и записывается в переменную elem, которая затем передается в метод класса H_List set_element(), который заполняет поля elem и is_elem. Файл1 закрывается.

После вызывается метод класса H_List browes(), который рекурсивно пробегается по иерархическому списку и сравнивает значения атомов с заданным значением. Если значения совпали, то переменной is_elem присваивается значение true.

В зависимости от значения поля is_elem(true/false) в файл2 записывается строка: “<elem> is/isn’t found”. Вызывается метод класса H_List free_list(), после чего файл2 закрывается и программа завершается.

Формат входных и выходных данных.

Входные данные представлены в виде строки, которая является краткой записью иерархического списка, и элемента (атома), наличие которого в этом списке нужно проверить.

Выходные данные представлены в виде строки, в которой говорится был найдем заданный элемент (атом) или нет.

Описание основных структур данных и функций.

Структуры данных:

- Class Node – элемент иерархического списка: узел или атом.
  - Поле Node* next – указатель на следующий элемент списка.
  - Поле std::variant<Node*, char> value – значение, лежащее в узле: указатель на начало списка-ответвления в иерархическом списке или символ.
- Class H_List – реализация иерархического списока
  - Поле char elem – хранит значение элемента(атома), наличие которого нужно проверить.
  - Поле bool is_elem – хранит true или false в зависимости от того, есть заданный элемент в списке или его нет.
  - Поле Node* head – указатель на голову иерархического списка.
  - Метод set_element() – принимает переменную типа char и передает ее значение полю elem, также передает полю is_elem значение false.
  - Метод get_is_elem() – возвращает значение поля is_elem.
  - Метод read_node() – принимает ссылку на объект класса string и тип данных int, создает узел или атом списка.

- Метод read_h_list() – принимает ссылку на объект класса string и тип данных int, создает иерархический список.
- Метод print() – принимает указатель на объект класса Node, выводит иерархический список.
- Метод browse() – принимает указатель на объект класса Node, пробегается по списку и сравнивает значение атомов со значением поля elem. Если значения совпали, то переменной is_elem присваивается значение true.
- Метод free_list() – принимает указатель на объект класса Node, пробегается по списку и очищает память.

Функции:

- bool isCorrect() – принимает ссылку на объект класса string, и проверяет корректность строки с краткой записью списка.

Разработанный программный код см. в приложении А.

Тестирование.

Тестирование происходит с помощью bash-скрипта. Он с помощью команд терминала запускает программу, подавая на вход файлы с тестами из директории Tests, и выводит результат. Также запускает программу без указания файлов и с указанием несуществующего файла test7.txt. Для запуска тестирования в консоли используется команда *make run_tests*.

Результаты тестирования см. в приложении В.

Выводы.

Произошло ознакомление с понятием иерархического списка, были получены навыки программирования функций для работы с иерархическими списками. Был реализован иерархический список на языке программирования C++.

Была разработана программа, считывающая краткую запись иерархического списка, создающая на её основе иерархический список и проверяющая список на вхождение заданного элемента (атома). Проведено тестирование работы программы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: **source/main.cpp**

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
#include <stack>
#include <variant>

#include "H_List.h"

bool isCorrect(const std::string& list);

int main(int argc, char** argv){
    if (argc < 3){
        std::cout << "Команда для запуска программы: ./lab2 /path/to/i
nput /path/to/output" << std::endl;
        return 1;
    }
    else{
        std::ifstream in(argv[1]);
        if (!in.is_open()){
            std::cout << argv[1] << ": No such file or directory" << s
td::endl;
            return 1;
        }
        std::ofstream out(argv[2]);
        if (!out.is_open()){
            std::cout << argv[2] << ": No such file or directory" << s
td::endl;
            return 1;
        }
        std::string list;
        H_List h_list;
        char elem;
        while(getline(in, list)){
            if (list.find('(') != std::string::npos){
                if (!isCorrect(list)){
                    out << "The list isn't correct." << std::endl;
                    in.close();
                    out.close();
                    return 1;
                }
                int i = 0;
                h_list.read_h_list(list,i);
            }
            else{
                elem = list[list.length()-1];
                h_list.set_element(elem);
            }
        }
        in.close();

        h_list.browse(h_list.head);
```

```
        if (h_list.get_is_elem()){
            out << elem << " is found\n";
        }
        else{
            out << elem << " isn't found\n";
        }

        h_list.free_list(h_list.head);
        out.close();
    }
    return 0;
}

bool isCorrect(const std::string& list){
    std::stack<char> Stack;
    if (list[0] != '('){
        return false;
    }
    for (char i : list){
        if (i == '('){
            Stack.push(i);
        }
        if (i == ')'){
            if (Stack.empty()){
                return false;
            }
            Stack.pop();
        }
    }
    return Stack.empty();
}
```

Название файла: **Node.h**

```
#ifndef NODE_H
#define NODE_H

#include <variant>

class Node{
public:
    Node * next;
    std::variant<Node*, char> value;
};

#endif
```

Название файла: **H_List.h**

```
#ifndef H_LIST_H
#define H_LIST_H

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

```
#include <variant>

#include "Node.h"

class H_List{
    char elem;
    bool is_elem;
public:
    Node* head;

    void set_element(char elem);
    char get_is_elem();
    Node* read_node(std::string& list, int& i);
    void read_h_list(std::string& list, int& i);
    void print(Node* cur);
    void browse(Node* cur);
    void free_list(Node* cur);
};

#endif
```

Название файла: **H_List.cpp**

```
#include "H_List.h"

void H_List::set_element(char elem){
    this->elem = elem;
    this->is_elem = false;
}
char H_List::get_is_elem(){
    return this->is_elem;
}
Node* H_List::read_node(std::string& list, int& i){
    Node* tmp = new Node;
    if(list[i] == ')'){
        delete tmp;
        return nullptr;
    } else
        if(list[i] == '('){
            i++;
            tmp->value = read_node(list, i);
            if(std::get<Node*>(tmp->value) == nullptr){
            return tmp;
            }
            i++;
            Node* tmp1 = std::get<Node*>(tmp->value);
            while (list[i] != ')'){
                if( list[i] != ' ') {
                    tmp1->next = read_node(list, i);
                    tmp1->next->next = nullptr;
                    tmp1 = tmp1->next;
                }
                i++;
            }
            return tmp;
        }
        else{
```

```
            tmp->value = list[i];
            tmp->next = nullptr;
        }
        return tmp;
}

void H_List::read_h_list(std::string& list, int& i){
    if(list[i] != ' ' && list[i] != ')'){
        if(list[i] == '('
            i++;
        head = read_node(list, i);
        Node* tmp = head;
        i++;
        while (list[i] != ')'){
            if( list[i] != ' ') {
                tmp->next = read_node(list, i);
                tmp->next->next = nullptr;
                tmp = tmp->next;
            }
            i++;
        }
    }
}

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

void H_List::browse(Node* cur){
    if (std::holds_alternative<Node *>(cur->value)) {
        if (std::get<Node*>(cur->value) != nullptr) {
            browse(std::get<Node*>(cur->value));
        }
    }
    else{
        if(std::get<char>(cur->value) == this->elem){
            this->is_elem = true;
            return;
         }
    }
    if (cur->next != nullptr) {
        browse(cur->next);
    }
}

void H_List::free_list(Node* cur){
```

```
    if (std::holds_alternative<Node *>(cur->value)) {
        if (std::get<Node*>(cur->value) != nullptr) {
            free_list(std::get<Node*>(cur->value));
            delete cur;
            return;
        }
    }
    if (cur->next != nullptr) {
        free_list(cur->next);
    }
    delete cur;
}
```

Название файла: **Makefile**

```
lab2: main.o h_list.o
    g++ -std=c++17 main.o h_list.o -o lab2

main.o: source/main.cpp source/H_List.h
    g++ -std=c++17 -c source/main.cpp

h_list.o: source/H_List.cpp source/H_List.h source/Node.h
    g++ -std=c++17 -c source/H_List.cpp

run_tests: lab2
    ./myscript

clean:
    rm -rf *.o
```

Название файла: **./myscript**

```
#!/bin/bash
for n in {1..6}
do
    arg=$(cat Tests/test$n.txt)
    echo -e "\nTest $n"
    echo "list = $arg"
    ./lab2 Tests/test$n.txt result$n
    res=$(cat result$n)
    echo "$res"
done
echo -e "\nTest 7\nВеденная команда: ./lab2"
./lab2
echo -e "\nTest 8"
./lab2 Tests/test8.txt result8
echo -e
```

ПРИЛОЖЕНИЕ В
ТЕСТИРОВАНИЕ

Таблица В.1 – Результаты тестирования

№	Входные данные	Выходные данные	Комментарии
1	(1(()23)4) Element to search for: 2	2 is found	Корректный список, содержащий заданный элемент (атом)
2	(a b (c d (e f (h g)))) Element to search for: f	f is found	Корректный список, содержащий заданный элемент (атом)
3	(1(()23)(56)7) Element to search for: 0	0 isn’t found	Корректный список, не содержащий заданный элемент (атом)
4	(f(t(asd)a) Element to search for: x	The list isn't correct.	Некорректный список
5	2(a(s(d)e)3)0) Element to search for: w	The list isn't correct.	Некорректный список
6	(1(()23)4(25)) Element to search for: 2	2 is found	Корректный список, содержащий заданный элемент (атом)
7	./lab2	Команда для запуска программы: ./lab2 /path/to/input /path/to/output	Программа запущена без аргументов (файлов для считывания и записи данных)
8	./lab1 Tests/test8.txt result8	Tests/test8.txt: No such file or directory	Программе передан несуществующий файл