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: Программирование рекурсивных алгоритмов

Студент гр. 9303 _____ Молодцев Д.А.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург

2020

Цель работы.

Создание базового функционала для работы с иерархическим списком, а также рекурсивной функции, которая разворачивает иерархический список на всех уровнях вложения. Также, работа подразумевает дальнейшее тестирование программы.

Задание.

Вариант 15

Проверить *структурную* идентичность двух иерархических списков (списки структурно идентичны, если их устройство (скобочная структура и количество элементов в соответствующих (под)списках) одинаково, при этом атомы могут отличаться);

Основные теоретические положения.

Рекурсивным называется объект, содержащий сам себя или определенный с помощью самого себя. Мощность рекурсии связана с тем, что она позволяет определить бесконечное множество объектов с помощью конечного высказывания. Точно так же бесконечные вычисления можно описать с помощью конечной рекурсивной программы. Рекурсивные алгоритмы лучше всего использовать, когда решаемая задача, вычисляемая функция или обрабатываемая структура данных определены с помощью рекурсии. Если процедура (функция) P содержит явное обращение к самой себе, она называется прямо рекурсивной. Если P содержит обращение к процедуре (функции) Q, которая содержит (прямо или косвенно) обращение к P, то P называется косвенно рекурсивной. Многие известные функции могут быть определены рекурсивно. Например факториал, который присутствует практически во всех учебниках по программированию, а также наибольший общий делитель, числа Фибоначчи, степенная функция и др.

Выполнение работы.

Для представления списка в памяти были реализованы следующие классы и объединение:

two_ptr – данный класс содержит всего 2 поля: List* head, List* tail, которые являются указателями на «голову» и «хвост» текущего элемента иерархического списка.

List – базовый класс, который описывает элемент списка, содержит два поля: bool is_atomic (является ли атомом текущий элемент), union Node. Объединение также включает в себя 2 поля: base atom (хранит информацию, которая содержится в элементе-атоме), two_ptr pair – элемент класса two_ptr.

Для работы со списком для класса List были реализованы следующие методы:

List* Get_head(const List* elem) – возвращает «голову» текущего элемента.

List* Get_end(const List* elem) – возвращает «хвост» текущего элемента.

bool isAtom(const List* elem) – true, если элемента списка атом.

bool isNull(const List* elem) – true, если элемент представляет пустой список.

Bool is_right(const string str) – false, если строка не начинается с открывающейся скобки или имеет лишние скобки(

List* Set_new_list(List* new_head, List* new_end) – создает новый вложенный список.

List* Set_atom(const base atom) – создает элемент-атом.

void ReadList(List* &y, const string& line, int& cur, int& len,vector<int>& v) – главная функция считывания списка.

void ReadElem(List* &y, const string& line, int& cur, int& len,vector<int>& v) – считывает элемент-атом списка.

void ReadBranch(List* &y, const string& line, int& cur, int& len,vector<int>& v) – считывает вложенный список.

void write_list(List* x) – главная функция печати списка.

void write_branch(List* x) – печатает элемент не атом.

void destroy(List* elem) – удаляет список.

Исходный код программы представлен в приложении А. Результаты тестирования включены в приложение Б

Выводы.

Был реализован иерархический список и базовый функционал для работы с ним (создание и считывание списка), ввод данных в программе может осуществляться через ввод из файла, а строка для сравнения считывается из консоли. Было проведено тестирование программы, результаты тестирования содержатся в приложении Б. Стоит отметить, что само понимание работы с иерархическим списком вызвало некоторую сложность из-за его рекурсивной природы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>


using namespace std;

typedef char base;

class List;

class two_ptr{
public:
    List* head;
    List* tail;
};


class List{
public:
    bool is_atomic;
    union{
        base atom;
        two_ptr pair;
    } Node;
};

List* Get_head(const List* s);//
List* Get_end(const List* s);
bool isAtom(const List* s);
```

```
bool isNull(const List* s);
List* Set_new_list(List* new_head, List* new_end);//
List* Set_atom(const base atom);//

void ReadList(List* &y,const string& line, int& cur, int&
len,vector<int>& v);//
void ReadElem(List* &y,const string& line, int& cur, int&
len,vector<int>& v);//
void ReadBranch(List* &y,const string& line, int& cur, int&
len,vector<int>& v);//

void write_list(const List* x);//
void write_branch(const List* x);//

void destroy(List* elem);//

bool is_right(const string &str){
    int flag=0;
    if(str[0]!='('){
        return false;
    }
    for(int i=0;i<str.length();i++){
        if(str[i]=='('){
            flag++;
        }
        if(str[i]==')'){
            flag--;
        }
    }
    return !flag;
}

int main() {
    string path;
```

```
cout<<"Enter the file name.\n";
getline(cin,path);
ifstream input(path);
if(!input){
    cout<<"Wrong file name, try again!\n";
}else{
    string analized_str;
    List* analized_list;
    vector<int> vect1;
    cout<<"Enter the analized string.\n";
    getline(cin,analized_str);

    int len = analized_str.length();
    int cur = 0;
    if(!is_right(analized_str)){
        cerr<<"Wrong analized string.\n";
        exit(1);
    }

    ReadList(analized_list,analized_str,cur,len,vect1);

    string file_line;
    while(getline(input, file_line)){
        if(!is_right(file_line)){
          cout<<"String "<<file_line<<" is wrong string.\n";
            continue;
        }
        vector<int> vect2;
        len = file_line.length();
        cur = 0;
        List* list_file;
        ReadList(list_file, file_line, cur, len,vect2);
        if(vect1==vect2){
            cout<<"List ";
```

```
                write_list(list_file);
                cout<<" and ";
                write_list(analized_list);
                cout<<" has the same structure.\n";
            }else{
                cout<<"List ";
                write_list(list_file);
                cout<<" and ";
                write_list(analized_list);
                cout<<" has the different structure.\n";
            }
        }
    }
    return 0;
}

List* Set_new_list(List* new_head, List* new_end){
    List* elem;
    if(isAtom(new_end)){
        cerr << "Error, tail is atomic" << endl;
        exit(1);
    }
    else{
        elem = new List;
        elem->is_atomic = false;
        elem->Node.pair.head = new_head;
        elem->Node.pair.tail = new_end;
        return elem;
    }
}

List* Set_atom(const base data){
    List* elem = new List;
    elem->is_atomic = true;
```

```
    elem->Node.atom = data;
    return elem;
}

List* Get_head(const List* elem){// PreCondition: not null (s)
    if (elem != nullptr){
        if (!isAtom(elem)){
            return elem->Node.pair.head;
        }else {
            cerr << "Error: Head(atom) \n";
            exit(1);
        }
    }
    else {
        cerr << "Error: Head(nullptr) \n";
        exit(1);
    }
}

List* Get_end(const List* elem){
    if (elem != nullptr){
        if (!isAtom(elem)){
            return elem->Node.pair.tail;
        }
        else {
            cerr << "Error: Tail(atom) \n";
            exit(1);
        }
    }
    else {
        cerr << "Error: Tail(nullptr) \n";
        exit(1);
    }
}
```

```
bool isAtom(const List* elem){
    if (elem == NULL){
        return false;
    }
    else {
        return (elem->is_atomic);
    }
}

bool isNull(const List* elem){
    return elem == NULL;
}

void ReadList(List* &y,const string& line, int& cur, int& len,
vector<int>& v){
    while(line[cur] == ' '){
        cur++;
    }
    ReadElem(y, line, cur, len,v);
}

void ReadElem(List* &y,const string& line, int& cur, int& len,
vector<int>& v){
    v.push_back(1);
    base prev = line[cur];
    if ( prev == ')'){
        cout << "Error" << endl;
        exit(1);
    }else if(prev != '('){
        y = Set_atom(prev);
    }else{
        ReadBranch(y, line, cur, len,v);
    }
```

```
}

void ReadBranch(List* &y,const string& line, int& cur, int& len,
vector<int>& v){
    v.push_back(2);
    List* firstElem;
    List* secondElem;

    if(cur >= len){
        cout << "Error" << endl;
        exit(1);
    }
    else{
        cur++;
        while( line[cur] == ' ') cur++;
        if ( line[cur] == ')'){
            y = nullptr;
        }
        else{
            ReadElem(firstElem, line, cur, len,v);
            ReadBranch(secondElem, line, cur, len,v);
            y = Set_new_list(firstElem, secondElem);
        }
    }
}

void destroy(List* elem){
    if(elem!=nullptr){
        if (!isAtom(elem)) {
            destroy(Get_head(elem));
            destroy(Get_end(elem));
        }
        delete elem;
    }
```

```
}

void write_list(const List* elem){
    if(isNull(elem)){
        cout<<"()";
    }else if(isAtom(elem)){
        cout<<"("<<elem->Node.atom<<")";
    }else{
        cout<<"(";
        write_branch(elem);
        cout<<")";
    }
}

void write_branch(const List* elem){
    if(!isNull(elem)){
        write_list(Get_head(elem));
        write_branch(Get_end(elem));
    }
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Файл со входными данными: input.txt

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	input.txt (a(s))	List (a(s)) and (a(s)) has the same structure.	
2.	input.txt (aa(lsd)14)	List ((h)((e)(l)((p))(m))(e)) and ((a)(a)((l)(s)(d))(1)(4)) has the different structure.	
3.	input.txt (11(000(qw)4)45)	List ((q)(w)((e)(f)(w)((h)(i))(k))(l)(m)) and ((1)(1)((0)(0)(0)((q)(w))(4))(4)(5)) has the same structure.	
4.	input_2.txt (aa)	List ((a)(e)(e)(e)) and ((a)(a)) has the different structure.	
5. (тест на некорректных данных)	input.txt dString d is wrong string.	String d is wrong string.	
6. (тест на различные корректные имена файла)	input_2.txt (q(w)0)	List ((a)(e)(e)(e)) and ((q)((w))(0)) has the different structure.	
7. (тест на некорректном имени файла)	rarely	Enter the file name. rarely Wrong file name, try again!	
8.	Input_2.txt	List ((I)((w)(a)	

	(0(0000)00(0(0(000))))	(n)(t))(t)(o)((s)((l)((e)(e)(p))))) and ((0)((0)(0)(0)(0))(0)(0)((0)((0)((0)(0)(0))))) has the same structure.	
9.	input.txt (aardu	Wrong analized string.	