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 _____ Микулик Д.П.

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 14

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

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:

Pair – 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 isAtomic (является ли атомом текущий элемент), union Node. Объединение также включает в себя 2 поля: base atom (хранит информацию, которая содержится в элементе-атоме), Pair pair.

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

List* head() – возвращает «голову» текущего элемента.
List* tail() – возвращает «хвост» текущего элемента.
bool isAtom() – true, если элемента списка атом.
bool isEmpty() – true, если элемент представляет пустой список.
List* reverse() – создает новый список, который был развернут на всех уровнях вложения.

Также были реализованы следующие функции (не получилось реализовать корректно как методы):
List* setElem(List* headNew, List* tailNew) – создает новый вложенный список.
List* setData(const base data) – создает элемент-атом.
List* reverseRec(List* head, List* tail) – разворачивает список.

void readList(List* &y) – главная функция считывания списка.
void readElem(base prev, List* &y) – считывает элемент-атом списка.
void readSeq(List* &y) – считывает вложенный список.

void writeList(List* y) – главная функция печати списка.
void writeSeq(List* y) – печатает элемент не атом.

void readListFromFile(List* &y, string& line, int& cur, int& len) – главная функция считывания списка из файла.
void readElemFromFile(List* &y, string& line, int& cur, int& len) – считывает элемент-атом списка из файла.

void readSeqFromFile(List* &y, string& line, int& cur, int& len) – считывает вложенный список из файла.

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 несколькими способами: через консоль или ввод из файла. Было проведено тестирование программы, результаты тестирования содержатся в отчете. 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 <fstream>
#include <string>
#include "list.h"

int main(){
    List* instance;
    int inputFlag = 0;
     cout << "Enter 0 or 1, 0 means console input, 1 means file
input: " << endl;
    cin >> inputFlag;
    cin.ignore(32767, '\n');

    string file_name;
    if (inputFlag == 1){
        cout << "Enter the name of an input file: " << endl;
        getline(cin, file_name);
        ifstream input(file_name);
        if (!input){
             cout << "You haven't entered correct input file." <<
endl;
        }
        else{
            cout << "Reverse list program: " << endl;
            string line;
            while(getline(input, line)){
                int len = line.length();
                int cur = 0;
                readListFromFile(instance, line, cur, len);
                cout << "Result input:" << endl;
                writeList(instance);
                cout << "\n";
                List* reversed = instance -> reverse();
                cout << "Result reversed:" << endl;
                writeList(reversed);
                cout << "\n";
            }
        }
    }else if(inputFlag == 0){
        cout << "Enter" << endl;
        readList(instance);
        cout << "Result input:" << endl;
        writeList(instance);
        cout << "\n";
        List* reversed = instance -> reverse();
        cout << "Result reversed:" << endl;
        writeList(reversed);
        cout << "\n";
    }
    else{
```

```
            cout << "Error" << endl;
        }
        return 0;
    }
```

Название файла: list.h

```
#ifndef LIST_H
#define LIST_H

#include <iostream>
#include <string>

using namespace std;

typedef char base;

class List;

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


class List{
public:
    bool isAtomic;
    union{
        base atom;
        Pair pair;
    } Node;


    List* head();
    List* tail();
    bool isAtom();
    bool isEmpty();
    List* reverse();

};

List* setElem(List* headNew, List* tailNew);
List* setData(const base data);
void readList(List* &y);
void readElem(base prev, List* &y);
void readSeq(List* &y);
void writeList(List* y);
void writeSeq(List* y);
List* reverseRec(List* head, List* tail);

void readListFromFile(List* &y, string& line, int& cur, int& len);
void readElemFromFile(List* &y, string& line, int& cur, int& len);
void readSeqFromFile(List* &y, string& line, int& cur, int& len);
```

```
#endif //LIST_H
```

Название файла: list.cpp

```
#include "list.h"

List* setElem(List* headNew, List* tailNew){
    List* newElem;
    if(tailNew -> isAtom()){
        cout << "Error, tail is atomic" << endl;
        exit(1);
    }
    else{
        newElem = new List;
        newElem->isAtomic = false;
        newElem->Node.pair.head = headNew;
        newElem->Node.pair.tail = tailNew;
        return newElem;
    }

}


List* setData(const base data){
    List* newElem = new List;
    newElem->isAtomic = true;
    newElem->Node.atom = data;
    return newElem;
}

List* List::reverse(){
    return reverseRec(this, nullptr);
}

List* reverseRec(List* head, List* tail){
    if(head->isEmpty()){
        return tail;
    }else if(head->head()->isAtom()){
                return  (reverseRec(head->tail(),  setElem(head->head(),
tail)));
    }else{
             return  (reverseRec(head->tail(),  setElem(reverseRec(head-
>head(), nullptr), tail)));
    }
}



List* List::head(){
    if(this != nullptr){
        if (this->isAtom() == false){
            return this->Node.pair.head;
        }
        else{
            cout << "Your element is atom!" << endl;
            exit(1);
```

```
        }
    }
    else{
        cout << "Head is empty!" << endl;
        exit(1);
    }
}

List* List::tail(){
    if (this != nullptr){
        if(this->isAtom() == false){
            return this->Node.pair.tail;
        }
        else{
            cout << "Your element is atom!" << endl;
            exit(1);
        }
    }
    else{
        cout << "Tail is empty!" << endl;
        exit(1);
    }
}

bool List::isAtom(){
    if (this->isEmpty())
        return false;
    return this->isAtomic;
}

bool List::isEmpty(){
    return this == nullptr;
}



void readListFromFile(List* &y, string& line, int& cur, int& len){
    while(line[cur] == ' '){
        cur++;
    }
    readElemFromFile(y, line, cur, len);
}

void readElemFromFile(List* &y, string& line, int& cur, int& len){
    base prev = line[cur];
    if ( prev == ')'){
        cout << "Error" << endl;
        exit(1);
    }else if(prev != '('){
        y = setData(prev);
    }else{
        readSeqFromFile(y, line, cur, len);
    }
}

void readSeqFromFile(List* &y, string& line, int& cur, int& len){
```

```
    //base x;
    List* firstElem;
    List* secondElem;

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

void readList(List* &y){
    base x;
    do cin >> x; while(x == ' ');
    readElem(x, y);
}

void readElem(base prev, List* &y){
    if ( prev == ')'){
        cout << "Error" << endl;
        exit(1);
    }else if(prev != '('){
        y = setData(prev);
    }else{
        readSeq(y);
    }
}

void readSeq(List* &y){
    base x;
    List* firstElem;
    List* secondElem;

    if(!(cin >> x)){
        cout << "Error" << endl;
        exit(1);
    }
    else{
        while( x == ' ') cin >> x;
        if ( x == ')'){
            y = nullptr;
        }
        else{
            readElem(x, firstElem);
            readSeq(secondElem);
```

```
            y = setElem(firstElem, secondElem);
        }
    }
}

void writeList(List* y){
    if(y->isEmpty()){
        cout << "()" << endl;
    }
    else if(y->isAtom()){
        cout << y->Node.atom;
    }
    else{
        cout << "(";
        writeSeq(y);
        cout << ")";
    }
}

void writeSeq(List* y){
    if(!y->isEmpty()){
        writeList(y->head());
        writeSeq(y->tail());
    }
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Файл со входными данными: input.txt

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	1 test.txt (a(b(cd))efg)	Enter 0 or 1, 0 means console input, 1 means file input: 1 Enter the name of an input file: test.txt Reverse list program: Result input: (a(b(cd))efg) Result reversed: (gfe((dc)b)a)	
2.	1 test.txt (a(b(c(d(e(f))))))	Result input: (a(b(c(d(e(f)))))) Result reversed: ((((((f)e)d)c)b)a)	
3.	1 test.txt (abcd(efg(hi)k)lm)	Result input: (abcd(efg(hi)k)lm) Result reversed: (ml(k(ih)gfe)dcba)	
4.	1 test.txt (a)	Result input: (a) Result reversed: (a)	
5. (тест на некорректных данных)	1 test.txt	Result input: a	

	a	Your element is atom!	
6. (тест на консольный ввод)	0 (a(bc)d)	Enter 0 or 1, 0 means console input, 1 means file input: 0 Enter (a(bc)d) Result input: (a(bc)d) Result reversed: (d(cb)a)	
7. (тест на некорректном имени файла)	1 asdasd	Enter 0 or 1, 0 means console input, 1 means file input: 1 Enter the name of an input file: asdasd You haven't entered correct input file.	