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 _____ Куршев Е.О

Преподаватель _____ Филатов Ар.Ю

Санкт-Петербург

2020

Цель работы.

Создание базовых функций для работы с иерархическим списком, а также создание рекурсивной функции, просматривает весь иерархический список на всех уровнях вложения. Работа подразумевает дальнейшее тестирование программы.

Задание

Вариант 10.

Подсчитать число различных атомов в иерархическом списке; сформировать из них линейный список;

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

В ходе выполнения работы были реализованы следующие функции и структуры:

1. class List; // класс, описывающий элемент поля;
2. class two_ptr; // класс содержащий оказатели;
3. void insert_at_bottom(char x, stack<char>& st);// вспомогательная функция для reverse;
4. void reverse(stack<char>& st);// функция разворота стека;
5. List* head(const List* s);//проверка головы;
6. List* end(const List* s);//проверка последнего элемента;
7. bool isAtom(const List* s);// проверка на атом;
8. bool isNull(const List* s);//проверка на нулевой элемент;
9. List* new_list(List* new_head, List* new_end);//создание нового списка;
10. List* make_atom(base atom);//создание атома;
11. void ReadList(List* &L, const string& str, int& cur, int& len,vector<char>& v, stack<char> &st, vector<char>& v2);//одна из функций для создания иерархического списка
12. void ReadElement(List* &L, const string& str, int& cur, int& len,vector<char>& v, stack<char> &st, vector<char>& v2);//одна из функций для создания иерархического списка
13. void ReadLine(List* &L, const string& str, int& cur, int& len,vector<char>& v, stack<char> &st, vector<char>& v2);//одна из функций для создания иерархического списка
14. void write_list(const List* x);//одна из функций для вывода списка;
15. void write_branch(const List* x);//одна из функций для вывода списка;
16. void destroy(List* elem);
17. bool isRight(const string &str);

Выводы.

Был реализован иерархический список и базовые функции для работы с ним, ввод данных в программе осуществляется из файла,. 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 <algorithm>
#include <string>
#include <fstream>
#include <vector>
#include <stack>

using namespace std;

typedef char base;

class List;

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

class List{
public:
    bool tag;
    union{
        base atom;
        two_ptr pair;
    } node;
};

void insert_at_bottom(char x, stack<char>& st);
void reverse(stack<char>& st);
List* head(const List* s);
List* end(const List* s);
```

```
    bool isAtom(const List* s);
    bool isNull(const List* s);
    List* new_list(List* new_head, List* new_end);
    List* make_atom(base atom);

    void ReadList(List* &L, const string& str, int& cur, int&
len,vector<char>& v, stack<char> &st, vector<char>& v2);//
    void ReadElement(List* &L, const string& str, int& cur, int&
len,vector<char>& v, stack<char> &st, vector<char>& v2);//
    void ReadLine(List* &L, const string& str, int& cur, int&
len,vector<char>& v, stack<char> &st, vector<char>& v2);//

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

    void destroy(List* elem);
    bool isRight(const string &str);

    int main() {
        string path;
        cout<<"Enter the file name:\n";
        getline(cin,path);
        ifstream input(path);
        if(!input)
            cout<<"Wrong file name!\n";
        else{
            cout << '\n';
            string line;
            while(getline(input, line)){
                if(!isRight(line)){
                    cout << line <<" is wrong string!!\n\n";
                    continue;
                }
                vector<char> vect, vect2;
                stack<char> st;
```

```
                int len = line.length();
                int cur = 0;
                List* L;
                ReadList(L, line, cur, len, vect, st, vect2);
                reverse(st);
                stack<char> tmp = st;
                cout << "Hierarchical list ";
                write_list(L);
                cout << " has " << vect2.size() << " different
atoms" << '\n';
                cout << "Linear list = (";
                while(!tmp.empty()){
                    cout << tmp.top();
                    tmp.pop();
                }
                cout << ")\n\n";

            }
        }
        return 0;
    }

    List* new_list(List* new_head, List* new_end){
        List* elem;
        if(isAtom(new_end)){
            cerr << "Error, tail = atomic!" << endl;
            exit(1);
        }
        else{
            elem = new List;
            elem->tag = false;
            elem->node.pair.head = new_head;
            elem->node.pair.tail = new_end;
            return elem;
        }
```

```
}

List* make_atom(base atom){
    List* elem = new List;
    elem->tag = true;
    elem->node.atom = atom;
    return elem;
}

List* head(const List* s){
    if (s != nullptr){
        if (!isAtom(s)){
            return s->node.pair.head;
        }else {
            cerr << "Error: Head(atom) \n";
            exit(1);
        }
    }
    else {
        cerr << "Error: Head(nullptr) \n";
        exit(1);
    }
}

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

```
        cerr << "Error: Tail(nullptr) \n";
        exit(1);
    }
}

bool isAtom(const List* s){
    if (s == nullptr){
        return false;
    }
    else {
        return (s->tag);
    }
}

bool isNull(const List* s){
    return s == nullptr;
}

void destroy(List* elem){
    if(elem != nullptr){
        if (!isAtom(elem)) {
            destroy(head(elem));
            destroy(end(elem));
        }
        delete elem;
    }
}

void write_list(const List* x){
    if(isNull(x))
        cout << "()";
    else if(isAtom(x))
        cout << "(" << x->node.atom << ")";
    else{
        cout << "(";
```

```
            write_branch(x);
            cout << ")";
        }
    }

    void write_branch(const List* x){
        if(!isNull(x)){
            write_list(head(x));
            write_branch(end(x));
        }
    }

    void ReadList(List* &L, const string& str, int& cur, int& len,
vector<char>& v, stack<char> &st, vector<char>& v2){
        while(str[cur] == ' ')
            cur += 1;
        ReadElement(L, str, cur, len, v, st, v2);
    }

    void ReadElement(List* &L, const string& str, int& cur, int&
len, vector<char>& v, stack<char> &st, vector<char>& v2){
        base prev = str[cur];
        v.push_back(prev);
        if(prev == ')'){
            cout << "Error" << endl;
            exit(1);
        }else if(prev != '('){
            if(find(v2.begin(), v2.end(), str[cur]) == v2.end())
                v2.push_back(prev);
            st.push(prev);
            L = make_atom(prev);
        }else{
            ReadLine(L, str, cur, len, v, st, v2);
        }
    }
```

```
    void ReadLine(List* &L, const string& str, int& cur, int& len,
vector<char>&v , stack<char> &st, vector<char>& v2){
        List* firstElem;
        List* secondElem;
        if(cur >= len){
            cout << "Error!\n" << endl;
            exit(1);
        }
        else{
            cur++;
            while(str[cur] == ' ') cur++;
            if (str[cur] == ')'){
                v.push_back(str[cur]);
                L = nullptr;
            }
            else{
                ReadElement(firstElem, str, cur, len, v, st, v2);
                ReadLine(secondElem, str, cur, len, v, st, v2);
                L = new_list(firstElem, secondElem);
            }
        }
    }

    void insert_at_bottom(char x, stack<char>& st){
        if(st.empty())
            st.push(x);
        else{
            char a = st.top();
            st.pop();
            insert_at_bottom(x, st);
            st.push(a);
        }
    }
```

```
void reverse(stack<char>& st){
    if(!st.empty()){
        char x = st.top();
        st.pop();
        reverse(st);
        insert_at_bottom(x, st);
    }
}

bool isRight(const string &str){
    int errors = 0;
    if(str[0] != '(')
        return false;
    for(char i : str){
        if(i == '(')
            errors++;
        else if(i == ')')
            errors--;
    }
    if(!errors)
        return true;
    else
        return false;
}
```

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Входной файл: input.txt

```
aaaaaaa
(a(b(c(a))))
(a(() (bc)d)e)
(h(e(l(l(o(w(o(r(l(d))))))))))
(dsf(sdf)
```

Результаты работы программы:

```
aaaaaaa is wrong string!!

Hierarchical list ((a)((b)((c)((a))))) has 3 different atoms
Linear list = (abca)

Hierarchical list ((a)(()((b)(c))(d))(e)) has 5 different
atoms
Linear list = (abcde)

Hierarchical list ((h)((e)((l)((l)((o)((w)((o)((r)((l)
((d))))))))))) has 7 different atoms
Linear list = (helloworld)

(dsf(sdf) is wrong string!!
```