**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: Рекурсивная обработка иерархических списков**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6383 |  | Карамышев А.О. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

**Цель работы.**

Ознакомиться с одной из часто используемых на практике нелинейных конструкций, способами её организации и рекурсивной обработки. Получить навыки решения задач обработки иерархических списков, как с использованием базовых функций их рекурсивной обработки, так и без использования рекурсии.

1. **Постановка задачи.**

Задание предполагает самостоятельную разработку студентом одного или нескольких модулей на языке С++, реализующих согласованный с преподавателем набор операций над иерархическими списками, а также главной программы, непосредственно решающей поставленную задачу. Предполагается выполнение задания в двух вариантах: с использованием базовых функций рекурсивной обработки иерархических списков и без использования рекурсии.

Во всех случаях, когда это не оговорено особо, предполагается, что исходные и результирующие списки размещаются в файлах подходящего типа. Для представления иерархических списков рекомендуется использовать сокращенную скобочную запись.

Вариант 9. проверить структурную идентичность двух иерархических списков (списки структурно идентичны, если их устройство (скобочная структура и количество элементов в соответствующих подсписках) одинаково, при этом атомы могут отличаться).

1. **Основные теоретические положения.**

В практических приложениях возникает необходимость работы с более сложными, чем линейные списки, нелинейными конструкциями. Рассмотрим одну из них, называемую иерархическим списком элементов базового типа El или S-выражением.

Определим соответствующий тип данных S\_expr (El) рекурсивно, используя определение линейного списка (типа L\_list):

< S\_expr (El) > ::= < Atomic (El) > | < L\_list (S\_expr (El)) >,

< Atomic (E) > ::= < El >.

< L\_list(El) > ::= < Null\_list > | < Non\_null\_list(El) >

< Null\_lis t> ::= Nil

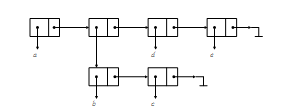
< Non\_null\_list(El) > ::= < Pair(El) >

< Pair(El) > ::= ( < Head\_l(El) > . < Tail\_l(El) > )

< Head\_l(El) > ::= < El >

< Tail\_l(El) > ::= < L\_list(El) >

Традиционно иерархические списки представляют или графически или в виде скобочной записи. На рис. 1 приведен пример графического изображения иерархического списка. Соответствующая этому изображению сокращенная скобочная запись － это (a (b c) d e).



*Рисунок 1. Пример представления иерархического списка в виде двумерного рисунка*

Переход от полной скобочной записи, соответствующей определению иерархического списка, к сокращенной производится путем отбрасывания конструкции . Nill и удаления необходимое число раз пары скобок вместе с предшествующей открывающей скобке точкой. Согласно приведенному определению иерархического списка, структура непустого иерархического списка － это элемент размеченного объединения множества атомов и множества пар «голова-хвост».

1. **Спецификация программы.**

Программа предназначена для проверки структурной идентичности двух списков.

Входными данными для программы являются:

Два иерархических списка в сокращенной скобочной записи.

1. **Пример диалога с пользователем.**

Введите ниже два списка, которые нужно проверить

(ф(фв)мм)

(м(мыв)в)

Bad!

1. **Тестирование.**

|  |  |  |
| --- | --- | --- |
| First | Second | Result |
| (1(1)) | (2(2)) | Cool |
| (()) | ((a)) | Bad! |
| (aaaaaaaaaaa) | (aaaaaaaaa()) | Bad! |
| (a(a(bv)c(cd))c) | (v(d(vc)v(qw))@) | Cool |
| () | ( ) | Bad! |

**Выводы****.**

В результате выполнения лабораторной работы были получены основные сведения о иерархических списках. Ознакомились с базовыми функциями обработки иерархических списков. Была написана функция, позволяющая сравнивать структуру двух иерарзических списков.

**ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД**

#include <iostream>

#include <string>

#include <ostream>

#include <conio.h>

#include <Windows.h>

using namespace std;

class Node {

public:

Node() {};

friend class List;

void destr(Node\* node);

void makeNodes(string& exp);

friend bool equel(Node \*a, Node \*b);

private:

Node \*head = nullptr;

Node \*tail = nullptr;

bool isAtom = false;

char data;

Node\* makeAtom(char data);

};

void Node::makeNodes(string& exp) {

if (exp[0] == ')') {

exp.erase(0, 1);

return;

}

else {

if (exp[0] != '(') {

head = makeAtom(exp[0]);

exp.erase(0, 1);

if (exp[0] == ')') {

exp.erase(0, 1);

return;

}

tail = new Node;

tail->makeNodes(exp);

}

else {

exp.erase(0, 1);

head = new Node;

head->makeNodes(exp);

if (exp[0] == ')') {

exp.erase(0, 1);

return;

}

tail = new Node;

tail->makeNodes(exp);

}

}

}

void Node::destr(Node\* node) {

if (node != nullptr)

{

if (!node->isAtom) {

destr(node->head);

destr(node->tail);

}

delete node;

}

}

Node\* Node::makeAtom(char data) {

Node \*node = new Node;

node->data = data;

node->isAtom = true;

return node;

}

class List {

public:

List(string list);

void destrList();

Node \*HeadEl = nullptr;

};

void List::destrList() {

HeadEl->destr(HeadEl);

};

List::List(string list) {

HeadEl = new Node;

if (list[0] == '(') {

list.erase(0, 1);

}

HeadEl->makeNodes(list);

}

bool equel(Node \*a, Node \*b) {

if (a == b)return true;

else if ((a != nullptr&& b == nullptr) || (a == nullptr && b != nullptr)) return false;

else {

if (!equel(a->head, b->head))return false;

if(!equel(a->tail, b->tail))return false;

return true;

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

cout << "Введите ниже два списка, которые нужно проверить\n";

string str, str1;

getline(cin, str);

List usual(str);

getline(cin, str1);

List usual1(str1);

if (equel(usual.HeadEl,usual1.HeadEl)) cout << "Cool";

else cout << "Bad!";

usual.destrList();

usual1.destrList();

system("pause > nul");

return 0;

}