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 иерархичских списков

Студент гр. 9303 _____ Ефимов М. Ю.

Преподаватель _____ Филатов А. Ю.

Санкт-Петербург

2020

Цель работы.

Познакомиться с одной из часто используемых на практике нелинейных конструкций, способами её организации и рекурсивной обработки.

Задание.

Вариант 8. Заменить в иерархическом списке все вхождения заданного элемента (атома) x на заданный элемент (атом) y;

Основные теоретические положения.

В практических приложениях возникает необходимость работы с болеесложными, 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) > ::= < At):omic (El) > | < L_list): (S_expr (El)) >,

< At):omic (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) >

Выполнение работы.

Для выполнения работы были использованны базовые функции ,данные нам , для работы со списком. Была написана рекурсивавная функция change_item для смены данных в атоме. Для этого использовалась данная функция для нахождения ссылок на следующие элементы head и tail. Так же использовалась данная функция isAtom. Так же выводятся промежуточные данные для отображения колличества фактических смен и общее колличество итераций.

Код программы смотрите в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	(a b c) c q	Колличество итераций:7 Колличество смен:1 (a b q)	Программа работает корректно
2.	() q w	Колличество итераций:1 Колличество смен:0 ()	Программа работает корректно
3.	(a a d)(a) a p	-	Программа работает не корректоно,из-за базовой фукнции, которая способна принять только один иерархический список

Выводы.

Было изучено понятие иерархические списки и принципы работы с ним. Были приобретены навыки по работе с иерархическими списками с помощью рекурсивных функций. Была реализована программа, включающая в себя рекурсивную функции для считывания, вывода и обработки иерархического списка. Также было проведено тестирование программы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdlib>
#include <locale>
using namespace std;

typedef char base;  // базовый тип элементов (атомов)

struct s_expr;
struct  two_ptr
{
    s_expr *hd;
    s_expr *tl;
};  //end two_ptr;

struct s_expr {
    bool tag; // true: atom, false: pair
    union
    {
        base atom;
        two_ptr pair;
    } node;     //end union node
};          //end s_expr

typedef s_expr *lisp;
lisp head(const lisp s);
lisp tail(const lisp s);
lisp cons(const lisp h, const lisp t);
lisp make_atom(const base x);
bool isAtom(const lisp s);
bool isNull(const lisp s);
```

```
void destroy(lisp s);
// функции ввода:
void read_lisp(lisp& y);            // основная
void read_s_expr(base prev, lisp& y);
void read_seq(lisp& y);
// функции вывода:
void write_lisp(const lisp x);      // основная
void write_seq(const lisp x);
//.....................................

lisp head(const lisp s)
{// PreCondition: not null (s)
    if (s != NULL) if (!isAtom(s))  return s->node.pair.hd;
    else { cerr << "Error: Head(atom) \n"; exit(1); }
    else {
        cerr << "Error: Head(nil) \n";
        exit(1);
    }
}
//.....................................
bool isAtom(const lisp s)
{
    if (s == NULL) return false;
    else return (s->tag);
}
//.....................................
bool isNull(const lisp s)
{
    return s == NULL;
}
//.....................................
lisp tail(const lisp s)
{// PreCondition: not null (s)
    if (s != NULL) if (!isAtom(s))  return s->node.pair.tl;
    else { cerr << "Error: Tail(atom) \n"; exit(1); }
```

```
        else {
            cerr << "Error: Tail(nil) \n";
            exit(1);
        }
    }
    //.....................................
    lisp cons(const lisp h, const lisp t)
    // PreCondition: not isAtom (t)
    {
        lisp p;
        if (isAtom(t)) { cerr << "Error: Tail(nil) \n"; exit(1); }
        else {
            p = new s_expr;
            if (p == NULL) { cerr << "Memory not enough\n";
exit(1); }
            else {
                p->tag = false;
                p->node.pair.hd = h;
                p->node.pair.tl = t;
                return p;
            }
        }
    }
    //.........................
    lisp make_atom(const base x)
    {
        lisp s;
        s = new s_expr;
        s->tag = true;
        s->node.atom = x;
        return s;
    }

    //.........................
    void destroy(lisp s)
```

```
    {
        if (s != NULL) {
            if (!isAtom(s)) {
                destroy(head(s));
                destroy(tail(s));
            }
            delete s;
            // s = NULL;
        };
    }
    //.........................
    // ввод списка с консоли
    void read_lisp(lisp& y)
    {
        base x;
        cin >> x;
        if (x == '(')
            read_s_expr(x, y);
        else{
            cerr << " ! List.Error " << endl; exit(1);
        }
    } //end read_lisp
      //.........................

    void read_s_expr(base prev, lisp& y)
    { //prev - ранее прочитанный символ}
        if (prev == ')') { cerr << " ! List.Error 1 " << endl;
exit(1); }
        else if (prev != '(') y = make_atom(prev);
        else read_seq(y);
    } //end read_s_expr
      //.........................
    void read_seq(lisp& y)
    {
        base x;
```

```
        lisp p1, p2;

        if (!(cin >> x)) { cerr << " ! List.Error 2 " << endl;
exit(1); }
        else {
            if (x == ')') y = NULL;
            else {
                read_s_expr(x, p1);
                read_seq(p2);
                y = cons(p1, p2);
            }
        }
    }
    //end read_seq
      //.........................
      // Процедура вывода списка с обрамляющими его скобками -
write_lisp,
      //  а без обрамляющих скобок - write_seq

    void write_lisp(const lisp x)
    {
        if (isNull(x)) cout << " ()";
        else if (isAtom(x)) cout << ' ' << x->node.atom;
        else
        { //непустой список}
            cout << " (";
            write_seq(x);
            cout << " )";
        }
    } // end write_lisp
      //.........................

    void write_seq(const lisp x)
    {//выводит последовательность элементов списка без обрамляющих
его скобок
```

```
        if (!isNull(x)) {
            write_lisp(head(x));
            write_seq(tail(x));
        }
    }

    void change_item (const lisp x, base item_1,base item_2,int &
change, int& count){
        count++;
        if(isAtom(x)){
            if(x->node.atom == item_1)
            {
            x->node.atom = item_2;
            change++;
            }
        }
        else{
            if(!isNull(x)){
                change_item  (head(x),  item_1,  item_2  ,  change,
count);
                change_item  (tail(x),  item_1,  item_2,  change,
count);
            }
        }
    }
    int main()
    {
        setlocale(LC_ALL, "Russian");
        lisp s1;
        bool flag = false;
        base item_1,item_2;
        cout << "введите первый список:" << endl;
        read_lisp(s1);
        write_lisp(s1);
        cout<<endl;
```

```
        cout << "введите элемент для замены:" << endl;
        cin>>item_1;
         cout << "введите элемент на который будем менять:" << endl;
        cin>>item_2;
        cout<<endl;
        int change = 0;
        int count = 0;
        change_item(s1,item_1, item_2, change, count);
        cout << "Колличество итераций:" << count << endl
<<"Колличество смен:" << change << endl;
        write_lisp(s1);
        destroy(s1);
        return 0;
    }
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Если результаты тестирования велики (больше 1 страницы), то их выносят в приложение.

Процесс тестирования можно представить в виде таблицы, например:

Таблица Б.2 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
4.			
5.			
6.			
...			

Обратите внимание, что в нумерации таблицы в приложении обязательно должен быть в качестве префикса номер самого приложения: А.