

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №2**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Иерархические списки**

Студент гр. 7383

\_\_\_\_\_

Кирсанов А.Я.

Преподаватель

\_\_\_\_\_

Размочева Н.В.

Санкт-Петербург

2018

## Содержание

Цель работы .....	3
Реализация задачи .....	3
Тестирование .....	4
Вывод.....	4
Приложение А. Тестовые случаи.....	5
Приложение Б. Исходный код программы .....	7

## **Цель работы**

Цель работы: научиться реализовывать иерархический список, освоить базовые функции работы с ним, научиться писать свои функции на основе рекурсии для работы с иерархическим списком.

Формулировка задачи: заменить в иерархическом списке все вхождения заданного элемента (атома) *x* на заданный элемент (атом) *y*.

## **Реализация задачи**

Программа состоит из функции `main` и функций работы с иерархическим списком.

Список состоит из структур. Каждая структура может содержать либо атом (в нашем случае атом – это символ), либо указатель на следующую и предыдущую структуры.

В функции `main` реализованы два способа ввода списка: с клавиатуры и из файла. Введенная или считанная из файла строка списка передается в поток `strstream`, затем вызывается функция `read_lisp`. Она пропускает пробелы и вызывает функцию `read_s_expr`, передавая в нее первый символ, не являющийся пробелом. `read_s_expr` создает в списке атом если переданный символ не '(' или вызывает функцию `read_seq`, которая создает две структуры списка. В каждую из них, посредством вызова функций `read_seq`, `read_s_expr` и `cons` может быть записан либо атом, либо указатель `NULL`, либо указатели на следующие две структуры. Процесс записи происходит до окончания списка, затем в `main` вызывается функция печати списка `write_lisp`. Эта функция печатает атом, если структура списка является им, либо вызывает функцию `write_seq` печати содержимого внутри скобок.

Функция `her` принимает на вход список и два атома, введенные на предыдущем шаге. Если список пуст, функция возвращает `false`. Если список указывает на атом, функция сравнивает его с заменяемым и заменяет его на атом *y*, если сравнение истинно. Иначе возвращает `false`. Если в голове нет указателя на атом, функция возвращает логическую сумму функций `her`, где в

качестве первого аргумента указатели на хвост или на голову. Если найдется искомый атом, он будет заменен и функция вернет в `main` `true`. В таком случае `main` выведет "Replaced successfully" и выведет измененный список. Иначе `main` выведет "No replacement items found".

## **Тестирование**

Сборка и тестирование программы производилось в среде разработки QT на Linux Ubuntu 16.04 LTS.

В ходе тестирования были использованы различные выражения, заведомо правильные или неправильные. Результаты тестирования представлены в приложении А.

При тестировании программы было обнаружено, что функция `rep` не всегда находит все заменяемые атомы. Ошибка возникла из-за того, что при рекурсивном вызове передавался только указатель `head` и некоторые атомы не были проверены из-за этого. Решением было возвращать логическую сумму функций `rep` с указателями на `head` и на `tail` элемента.

## **Вывод**

В ходе работы были изучены иерархические списки. Получен опыт работы с базовыми функциями таких списков. Реализована рекурсивная функция поиска заданного атома для последующей замены его определенным значением. Найдены и исправлены ошибки в работе функции.

## ПРИЛОЖЕНИЕ А.

### ТЕСТОВЫЕ СЛУЧАИ

Таблица 1 — Тестовые случаи

Входные данные	Вывод программы
1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program. 2 Enter a list: List entered: ( ) Enter x y a d	No replacement items found.
1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program. 2 Enter a list: (dsa) List entered: ( d s a ) Enter x y d i	Replaced successfully. Processed list: ( i s a )
1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program. 1 Enter file name: input Input file not open! ( ) Enter x y a d	No replacement items found.
1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program. 2 Enter a list: (s(ks)sqew(as(sdd)sd)xzxz) List entered: ( s ( k s ) s q e w ( a s ( s d d ) s d ) x z x z ) Enter x y z u	Replaced successfully. Processed list: ( s ( k s ) s q e w ( a s ( s d d ) s d ) x u x u )

1 - Reading from file, 2 - Keyboard input, 3 - Exit from the program. 1 Enter file name: input ( a s d f g h j g f d s ) Enter x y g t	Replaced successfully. Processed list: ( a s d f t h j t f d s )
--	--

## ПРИЛОЖЕНИЕ Б.

### ИСХОДНЫЙ КОД ПРОГРАММЫ

```
main.cpp:
#include "l_intrfc.h"
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <cstdlib>
#include <strstream>
#define N 1000

using namespace h_list;
using namespace std;

int main()
{
    base x, y;
    lisp s = NULL;
    strstream st;
    int k = 0;
    char str[N];

    while(k != 3){
        cout << endl << "1 - Reading from file, 2 - Keyboard input, 3
- Exit from the program." << endl;
        cin >> k;
        switch (k) {
            case 1:{
                cout << "Enter file name:" << endl;
                cin >> str;
                ifstream outfile(str);
                if (!outfile) { cout << "Input file not open!" <<
endl; break; }
                outfile.read(str, N);
                outfile.close();
                st << str;
                break;
            }
            case 2:{
                cout << "Enter a list:" << endl;
                cin.get();
                cin.getline(str, N);
                cout << "List entered: " << endl;
            }
        }
    }
}
```

```

        st << str;
        break;
    }
    case 3:{ cout << "Press Enter\n"; destroy(s); return 0; }
}

read_lisp(s, st);
write_lisp (s);
cout << endl;
cout << "Enter x y\n";
cin >> x >> y;
if(rep(s, x, y)){
    cout << "Replaced successfully." << endl;
    cout << "Processed list:" << endl;
    write_lisp (s);
}
else cout << "No replacement items found." << endl;

    cout << endl;
}
return 0;
}

```

l\_impl.cpp:

```

#include <iostream>
#include <sstream>
#include "l_intrfc.h"
using namespace std;
namespace h_list
{
    bool rep(lisp s, base x, base y){
        if(isNull(s)) return false;
        if(isAtom(s)){
            if(getAtom(s) == x){ s->node.atom = y; return true; }
            return false;
        }
        else return rep(head(s), x, y) + rep(tail(s), x, y);
    }
    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;
    };
    }

    base getAtom (const lisp s)
    {
        if (!isAtom(s)) { cerr << "Error: getAtom(s) for !isAtom(s)
\n"; exit(1);}
        else return (s->node.atom);
    }

    void read_lisp ( lisp& y, strstream &st) // ввод списка с консоли
        { base x;

        do{ st >> x; } while (x==' ');

        if(!(y == NULL))
            read_s_expr (x, y, st);
        }

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

    void read_seq ( lisp& y, strstream &st)
        { base x;
        lisp p1, p2;

```

```

        if (!(st >> x)) {cerr << " ! List.Error 2 " << endl; exit(1);}
        else {
            while ( x==' ' ){ st >> x; }
            if ( x == ')' ) y = NULL;
            else {
                read_s_expr ( x, p1, st);
                read_seq ( p2, st);
                y = cons (p1, p2);
            }
        }
    }

// 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 << " )";
    }
}

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));
    }
}

}

l_intrfc.h:
// интерфейс АД "Иерархический Список"
#include <sstream>
#define N 1000
using namespace std;
namespace h_list
{
    typedef char base;

    struct s_expr;

```

```

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

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

    typedef s_expr *lisp;

bool rep(lisp s, base x, base y);
    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);

    base getAtom (const lisp s);

void read_lisp ( lisp& y, strstream &st);
void read_s_expr (base prev, lisp& y, strstream &st);
void read_seq ( lisp& y, strstream &st);

void write_lisp (const lisp x);
    void write_seq (const lisp x);

}

```