

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МОЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №2**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Рекурсивная обработка иерархических списков.**

Студент гр. 9384

\_\_\_\_\_

Прашутинский К.И.

Преподаватель

\_\_\_\_\_

Ефремов М.А.

Санкт-Петербург

2020

### Цель работы.

Создать иерархический список. Проверить созданный ранее список на синтаксическую корректность и обратиться список на всех уровнях вложения.

### Задание.

#### ВАРИАНТ 14.

Обратить иерархический список на всех уровнях вложения; например, для исходного списка (a (b c) d) результатом обращения будет список (d (c b) a).

### Выполнение работы.

Была реализована релизована функция обращения списка `lisp rev(const lisp s, const lisp z)`, принимающая два иерархических списка, обходящая их по глубине и ширине и возвращает указатель на обращенный список. В `main()` ввод был реализован при помощи считывания из файла. Программа считывает с консоли строку и выводит резьтат или же обращенный на всех уровнях вложения список.

### Экспериментальные результаты.

Ввод:

```
((accgr()ds)sdda(sdf fa))))
```

Вывод:

```
(( (( a f f d s ) a d d s ( s d ( ) r g c c a ) ) ) )
```

```
( a ( b ) c ( d ) e )
```

```
( e ( d ) c ( b ) a )
```

```
( )
```

```
( )
```

### Выводы.

В ходе выполнения лабораторный работы был создан иерархический список. Также были использованы алгоритмы для обхода иерархического списка и его обработка.

## ПРОТОКОЛ

Название файла: main.cpp

```
#include <iostream>

#include <cstdlib>
#include "h_list.h"
#include <windows.h>

using namespace std;
using namespace h_list;

int main()
{
    lisp expr;
    cout << "Enter list:" << "\n";
    read_lisp(expr);
    cout << "Enter list: " << "\n";
    write_lisp(expr);
    cout << "\n";
    expr = h_list::reverse(expr);
    cout << "reversed list:" << "\n";
    write_lisp(expr); cout << "\n";
    system("pause");
    return 0;
}
```

Название файла: h\_list.h

```
#pragma once

namespace h_list
{
    typedef char base;
    struct s_expr;
    struct two_ptr
    {
        s_expr* hd;
        s_expr* tl;
    };
}
```

```

}; //end two_ptr;
struct s_expr {
    bool IsAtom;
    union
    {
        base atom;
        two_ptr pair;
    } 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);
base getAtom(const lisp s);

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

void write_lisp(const lisp x);
void write_seq(const lisp x);
//.....
lisp reverse(const lisp s);
lisp rev(const lisp s, const lisp z);
}

```

Название файла: h\_list.cpp

```

#include "h_list.h"

#include <iostream>
#include <cstdlib>
using namespace std;
namespace h_list
{
    //.....
    lisp head(const lisp 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->IsAtom);
    }
    //.....

    bool isNull(const lisp s)
    {
        return s == NULL;
    }
    //.....
    lisp tail(const lisp 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: Cons(*, atom)\n"; exit(1); }
        else {
            p = new s_expr;
            if (p == NULL) { cerr << "Memory not enough\n"; exit(1); }
            else {
                p->IsAtom = 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->IsAtom = true;
    s->node.atom = x;
    return s;
}
//.....
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)
{
    base x;
    do cin >> x; while (x == ' ');
    read_s_expr(x, y);
}
//.....
void read_s_expr(base prev, lisp& y)
{
    if (prev == ')') { cerr << " ! List.Error 1 " << endl; exit(1); }
    else if (prev != '(') y = make_atom(prev);
    else read_seq(y);
}
//.....
void read_seq(lisp& y)
{
    base x;
    lisp p1, p2;
    if (!(cin >> x)) { cerr << " ! List.Error 2 " << endl; exit(1); }
    else {
        while (x == ' ') cin >> x;
        if (x == ')') y = NULL;
        else {
            read_s_expr(x, p1);
            read_seq(p2);
            y = cons(p1, p2);
        }
    }
}

```

```

        }
    }
    //.....

void write_lisp(const lisp x)
{
    if (isNull(x)) cout << " ()";
    else if (isAtom(x)) cout << ' ' << x->node.atom;
    else {
        cout << " (";
        write_seq(x);
        cout << ")";
    }
}
//.....
void write_seq(const lisp x)
{
    if (!isNull(x)) {
        write_lisp(head(x));
        write_seq(tail(x));
    }
}
// -----
lisp reverse(const lisp s)
{
    return(rev(s, NULL));
}
//.....
lisp rev(const lisp s, const lisp z)
{
    if (isNull(s)) return(z);
    else if (isAtom(head(s))) return(rev(tail(s), cons(head(s), z)));
    else return(rev(tail(s), cons(rev(head(s), NULL), z)));
}
}

```