

**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: Иерархические списки**

Студент гр. 8304

\_\_\_\_\_

Барышев А.А.

Преподаватель

\_\_\_\_\_

Фирсов К.В.

Санкт-Петербург

2019

## **Цель работы.**

Написать программу, которая будет проверять идентичность двух иерархических списков.

## **Описание программы.**

Настоящая программа принимает два иерархических списка с помощью функции `void read_lisp(lisp&)`. Списки передаются в функцию `bool compareS_expr(const lisp, const lisp, bool&)`, которая проверит их на идентичность. Параметр `bool&` отвечает за прерывание рекурсивного сравнения двух иерархических списков, то есть, как только в процессе сравнения элементов иерархических списков находится отличие, то флаг устанавливается в `false` и рекурсия прекращается. Все копии данной функции, порождённые рекурсией, завершаются со значением `false`. В другом случае, если ни разу не будет выявлено несовпадения, то функция вернёт `true` (значение `bool&` по умолчанию) .

## **Функции**

Прототипы основных функций данной программы:

```
bool compareS_expr(const lisp lispFirst, const lisp lispSecond, bool &flag);
```

```
bool helpCompare(const lisp x, const lisp y, bool &flag);
```

Данные функции представляют собой взаимную рекурсию, так как `compareS_expr` вызывает `helpCompare`, а `helpCompare` вызывает `compareS_expr`.

Прочие функции, участвовавшие в основных функциях сравнения списков, относились к стандартному интерфейсу, через который возможен доступ к элементам данного АТД. Прототипы таких функций:

```
lisp head (const lisp);
```

```
lisp tail (const lisp);
```

```
lisp cons (const lisp h, const lisp);
```

```
lisp make_atom (const base);
```

```
bool isAtom (const lisp);
bool isNull (const lisp);
void destroy (lisp);
void read_lisp ( lisp&);
void read_s_expr (base, lisp&);
void read_seq ( lisp&);
```

## **Тесты.**

Здесь приведены результаты тестирования: пары списков и результат их сравнения. Для каждой пары соответствующий результат сравнения приведён ниже.

1-st lisp is: (bcx)  
2-nd lisp is: (adwdadaw)  
Result: They are not equal

1-st lisp is: (adwdadaw)  
2-nd lisp is: (bcx)  
Result: They are not equal

1-st lisp is: (a)  
2-nd lisp is: (a)  
Result: They are equal

1-st lisp is: (a(d)zx)  
2-nd lisp is: (a(d)zx)  
Result: They are equal

1-st lisp is: (a(d)zx)  
2-nd lisp is: (a(d)z(x))  
Result: They are not equal

1-st lisp is: (a(d)zx)  
2-nd lisp is: (a(vd()))  
Result: They are not equal

1-st lisp is: (a(vd()))  
2-nd lisp is: (a(vd()))  
Result: They are equal

1-st lisp is: (a(vd(x))

2-nd lisp is: (a(vd()z))

Result: They are not equal

1-st lisp is: (((cda)))

2-nd lisp is: (((cda)))

Result: They are equal

1-st lisp is: (((cda)daw)dz(()))

2-nd lisp is: (((cda)daw)dz(()))

Result: They are equal

1-st lisp is: (((cda)daw)dz(()cx)z)

2-nd lisp is: (((cda)daw)dz(()xc)z)

Result: They are not equal

1-st lisp is: (((cda)daw)dz(()cx)z)

2-nd lisp is: (((cda)daw)dz(()cx)z)

Result: They are equal

1-st lisp is: (dawdhwau(hd(za(xc(qd)gr))d)dhwuo(ad)aw)

2-nd lisp is: (dawdhwau(hd(za(xc(qd)gr))d)dhwuo(ad)aw)

Result: They are equal

1-st lisp is: (dawdhwau(hd(zaxc(qd)gr))d)dhwuo(ad)aw)

2-nd lisp is: (dawdhwau(hd(za(xc(qd)gr))d)dhwuo(ad)aw)

Result: They are not equal

## **Выводы.**

Был получен опыт обработки иерархических списков.

## КОД ПРОГРАММЫ.

```
#include <iostream>

typedef char base;
struct s_expr;

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

struct s_expr{
    bool tag;
    base atom;
    ptr pair;
};

typedef s_expr* lisp;
void read_lisp (lisp& y);
void read_s_expr(base prev, lisp& y);
void read_seq(lisp& y);
void write_seq(const lisp x);
bool compareS_expr(const lisp lispFirst, const lisp lispSecond, bool &flag);
bool helpCompare(const lisp x, const lisp y, bool &flag);

bool isAtom (const lisp s) {
    if(s == NULL)
        return false;
    else return
        (s -> tag);
}

bool isNull (const lisp s) {
    return s == NULL;
}

lisp head(const lisp s) {
    if (s != NULL)
        if (!isAtom(s))
            return s->pair.hd;
    else {
```

```

        std::cerr << "Error: Head(atom)" << std::endl << std::endl;
        exit(1);
    }
    else {
        std::cerr << "Error: Head(nil)" << std::endl << std::endl;
        exit(1);
    }
}

```

```

lisp tail(const lisp s) {
    if (s != NULL)
        if (!lisAtom(s))
            return s->pair.tl;
        else {
            std::cerr << "Error: Tail(atom)" << std::endl << std::endl;
            exit(1);
        }
    else {
        std::cerr << "Error: Tail(nil)" << std::endl << std::endl;
        exit(1);
    }
}

```

```

void destroy(lisp s) {
    if (s != NULL) {
        if (!lisAtom(s)) {
            destroy(head(s));
            destroy(tail(s));
        }
        delete s;
    }
}

```

```

lisp cons(lisp const h, lisp const t) {
    lisp p;
    if (isAtom(t)) {
        std::cerr << "Error: cons(*, atom) \n";
        exit(1);
    }
    else {
        p = new s_expr;
        p->tag = false;
    }
}

```

```

        p->pair.hd = h;
        p->pair.tl = t;
        return p;
    }
}

```

```

lisp make_atom(char const x) {
    lisp s;
    s = new s_expr;
    s->tag = true;
    s->atom = x;
    return s;
}

```

```

void read_lisp(lisp& y) {
    base x;
    do{
        std::cin >> x;
    }
    while(x==' ');
    read_s_expr (x, y);
}

```

```

void read_s_expr(base prev, lisp& y) {
    if(prev != '(') {
        y = make_atom(prev);
    }
    else{
        read_seq(y);
    }
}

```

```

void read_seq(lisp& y) {
    base x;
    lisp p1, p2;
    std::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))
        std::cout << "()";
    else if(isAtom(x))
        std::cout << x->atom;
    else{
        std::cout << "(";
        write_seq(x);
        std::cout << ")";
    }
}

```

```

void write_seq (const lisp x) {
    if (!isNull(x)){
        write_lisp(head (x));
        write_seq(tail (x));
    }
}

```

```

bool compareS_expr(const lisp lispFirst, const lisp lispSecond, bool &flag) {
    if(flag == false)
        return false;
    if(isAtom(lispFirst) || isAtom(lispSecond)) {
        if(isAtom(lispSecond) && isAtom(lispSecond)) {
            if(!(lispSecond->atom == lispFirst->atom)) {
                flag = false;
            }
        }
        else{
            flag = false;
        }
    }
    else if(!(isAtom(lispFirst)) && !(isAtom(lispSecond))) {
        if(helpCompare(lispFirst, lispSecond, flag) == false)
            flag = false;
    }
    else{

```



```

        flag = false;
    }
    return flag;
}

bool helpCompare(const lisp x, const lisp y, bool &flag) {
    if(!isNull(x)) {
        if(isNull(x) || isNull(y))
            return false;
        compareS_expr(head(x), head(y), flag);
        if(isNull(x) || isNull(y))
            return false;
        helpCompare(tail(x), tail(y), flag);
    }
    return true;
}

int main() {
    lisp lst1 = new s_expr;
    lisp lst2 = new s_expr;
    bool compareHierarchicalResult = true;
    bool *compareHierarchicalResultPointer = &compareHierarchicalResult;

    read_lisp(lst1);
    read_lisp(lst2);
    std::cout << "1-st lisp is: ";
    write_lisp(lst1);
    std::cout << std::endl << "2-nd lisp is: ";
    write_lisp(lst2);

    if(compareS_expr(lst1, lst2, *compareHierarchicalResultPointer))
        std::cout << std::endl << "Result: They are equal" << std::endl;
    else
        std::cout << std::endl << "Result: They are not equal" << std::endl;

    destroy(lst1);
    destroy(lst2);

    return 0;
}

```