

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. 9382

Кузьмин Д. И.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Познакомиться 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.

Основные теоретические положения.

Иерархическим списком называется структура данных, элементы которой могут находиться как на разных уровнях иерархии, так и на одном. Ссылка на следующий элемент называется «хвост», а ссылка на элемент, лежащий на более низком уровне иерархии «голова». «Голова» может являться атомом, при этом «хвост» — нет. Структура непустого иерархического списка — это элемент размеченного объединения множества атомов и множества пар «голова-хвост».

Задание.

Вариант 12.

проверить идентичность двух иерархических списков.

Выполнение работы.

1) Первым шагом было создание структур, определяющую элемент списка. Эта структура состоит из `bool tag` — показывает, является ли элемент атомом; `base atom` — атомарное значение элемента, если он атом; `s_expr* hd` — ссылка на «голову»; `s_expr* tl` — ссылка на хвост.

2) Затем были созданы функции для работы со списком. Среди них: `lisp head(const lisp s)` - возвращается голову списка; `lisp tail(const lisp s)` - возвращается хвост списка; `lisp cons(const lisp h, const lisp t)` - добавляет элемент(не атом) в список; `lisp make_atom(const base x)` - добавляет атом в список - `bool isAtom(const lisp s)` проверка является ли элемент атомом; `bool isNull(const lisp s)` — проверка является ли список пустым; `void destroy(lisp s)` - удаляет список; `base getAtom(const lisp s)` — получает значение атома; `void`

read_lisp(lisp& y, FILE* f = nullptr) - считывание всего списка; void read_s_expr(base prev, lisp& y, FILE* f = nullptr) — считывание отдельно взятого элемента; void read_seq(lisp& y, FILE* f = nullptr); считывание подписка; void write_lisp(const lisp x); вывод всего списка; void write_seq(const lisp x) — вывод отдельного элемента

4) Далее была реализована функция bool lisp_cmp(lisp l1, lisp l2), проверяющая идентичность двух списков. Эта функция сначала проверяет являются ли сравниваемые элементы атомами или нет. Затем, если два элемента атомы — она проверяет их значение, и возвращается соответственно true, если они равны и false, если нет. В случае, если оба элемента не атомы, то та же функция вызывается рекурсивно, но уже для элементов, расположенных на следующей ступени иерархии. В случае, если сравниваемые элементы имеют разную «природу», например один — атом, другой — нет, функция возвращает false. В случае, если сравнивается два пустых списка, возвращается true.

Исходный код см. в приложении А.

Графическое представление списка, используемого в работе см. в приложении В.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 — результаты тестирования.

№ п/п	Входные данные	Выходные данные	Комментарий
0	a a	Списки идентичны	Простейший случай. Оба списка атомы, и их значения равны.
1	(a b c)	Списки	Оба списка состоят из

	(a b c)	идентичны	трех равных элементов.
2	() ()	Списки идентичны	Оба списка пусты, поэтому идентичны.
3	(a b (c d)) (a b c (d))	Списки не идентичны	Элемент b в первом списке имеет хвост (c d), а во втором тот же элемент имеет хвост (c (d)).
4	((((a)))) (((a)))	Списки не идентичны	Один и тот же элемент, но находится на разных уровнях иерархии.
5	(a a a) (a (a (a)))	Списки не идентичны	Три элемента, но в первом случае — на одном уровне, во втором — на разных.
6	(a b c) (a b c)	Списки идентичны	Пробелы в изначальном вводе не влияют на список.
7	(c b a) (a b c)	Списки не идентичны.	Равные элементы, но их порядок противоположный

Выводы.

Был изучен принцип устройства иерархических списков. Получены навыки разработки программы, работающей с иерархическими списками.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <cstdio>
#include <conio.h>

using namespace std;

typedef char base; // базовый тип атомов
struct s_expr;
struct s_expr {
    bool tag;
    base atom;
    s_expr* hd;
    s_expr* tl;
};

typedef s_expr* lisp;

lisp head(const lisp s); // возвращается голову списка
lisp tail(const lisp s); // возвращается хвост списка
lisp cons(const lisp h, const lisp t); // добавляет элемент (не атом) в
список
lisp make_atom(const base x); // добавляет атом в список
bool isAtom(const lisp s);
bool isNull(const lisp s);
void destroy(lisp s); // уничтожает список
base getAtom(const lisp s);
// функции ввода:
void read_lisp(lisp& y, FILE* f = nullptr); // основная (весь список)
void read_s_expr(base prev, lisp& y, FILE* f = nullptr); // отдельно взятый
элемент
void read_seq(lisp& y, FILE* f = nullptr); // для элементов, не являющимися
атомами
// функции вывода:
void write_lisp(const lisp x); // основная (весь список)
void write_seq(const lisp x); // для элемента

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

    else {

        return nullptr;

    }
}

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

bool isNull(const lisp s)
{

```

```

        return s == nullptr;
    }

lisp tail(const lisp s)
{
    if (s != nullptr)
        if (!isAtom(s)) return s->tl;

    else{

        return nullptr;

    }
}

lisp cons(const lisp h, const lisp t)
{
    lisp p;
    if (isAtom(t)){
        cout << "Хвост не может являться атомом";
    }
    else {

        p = new s_expr;
        p->tag = false;
        p->hd = h;
        p->tl = t;
        return p;

    }
}

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

void destroy(lisp s) // удаление списка и освобождение памяти
{
    if (s != nullptr) {
        if (!isAtom(s)) {
            destroy(head(s));
            destroy(tail(s));
        }
        delete s;
    }
};

base getAtom(const lisp s) // получение значения атома элемента
{
    if (!isAtom(s))
        cout << "не атом\n";
    else return (s->atom);
}

```

```

}

void read_lisp(lisp& y, FILE* f) // считывание списка из файла либо
консоли
{
    base x = fgetc(f);
    cout << "Считывание списка: " << endl;

    while (x == ' ' || x == '\n') {

        x = (char)getc(f);

    }

    read_s_expr(x, y, f);
}

void read_s_expr(base prev, lisp& y, FILE* f) // считывание следующего
элемента и проверка является ли он атомом или нет
{
    if (prev == ')') cout << "Первый символ не может быть \")\"";
    else if (prev != '(') {
        y = make_atom(prev);
    }
    else read_seq(y, f);
}

void read_seq(lisp& y, FILE* f) // считывание отдельного элемента
списка(не атома)
{
    base x = getc(f);
    lisp p1, p2;
    if (x == '\n') cout << " the end";
    else {
        while (x == ' ') x = fgetc(f);
        if (x == ')') y = nullptr;
        else {
            read_s_expr(x, p1, f); //рекурсивный вызов для
следующего элемента
            read_seq(p2, f);
            y = cons(p1, p2);
        }
    }
}

void write_lisp(const lisp x) // вывод всего списка
{
    if (isNull(x)) cout << "()";
    else if (isAtom(x)) cout << ' ' << x->atom;
    else {
        cout << " (";
        write_seq(x);
        cout << " )";
    }
}

```

```

void write_seq(const lisp x) //вывод отдельного элемента без скобок
{
    if (!isNull(x)) {
        write_lisp(head(x));
        write_seq(tail(x));
    }
}
//.....
bool lisp_cmp(lisp l1, lisp l2) {

    if (l1 == nullptr && l2 == nullptr) { //проверка, являются ли оба
        списка пустыми
        return true;
    }
    else if (isAtom(l1) && isAtom(l2)) { // проверка, являются ли оба
        элемента атомами, если да, проверяется их значения
        cout << "Проверка значений атомов элементов " << getAtom(l1)
        << " и " << getAtom(l2) << "\n";
        return (getAtom(l1) == getAtom(l2));
    }
    else if (isAtom(l1) != isAtom(l2)) {
        cout << "Один элемент является атомом, другой - нет\n";
        return false;
    }
    else if ( !isAtom(l1) && !isAtom(l2) ) { //в случае, если элементы
        не являются атомами, осуществляется проверка головы и хвоста
        cout << "Элементы ";
        write_lisp(l1);
        cout << " и ";
        write_lisp(l2);
        cout << "не являются атомами. Проверка головы и хвоста
        элементов\n";

        if (lisp_cmp(head(l1), head(l2))) {
            cout << "Проверка хвостов ";
            write_lisp(tail(l1));
            cout << " и ";
            write_lisp(tail(l2));
            cout << endl;
            if (lisp_cmp(tail(l1), tail(l2)))
                return true;
        }
    }

}

int main() {

    setlocale(LC_ALL, "Russian");
    lisp l1, l2;
    FILE* f1 = fopen("file.txt", "r+");
    cout << "Проверка идентичности двух иерархических списков. Введите
    1 для ввода списков с консоли или любой другой символ для ввода с
    файла\n";
    char a = getc(stdin);
    if (a == '1') {
        read_lisp(l1, stdin);

```



```

        read_lisp(l2, stdin);
    }
    else {
        read_lisp(l1, f1);
        read_lisp(l2, f1);
    }
    write_lisp(l1);
    cout << endl;
    write_lisp(l2);
    cout << endl;

    if (lisp_cmp(l1, l2)) {

        printf("Списки идентичны\n");
    }
    else {

        printf("Списки не идентичны\n");
    }

    system("pause");
    return 0;
}

```

ПРИЛОЖЕНИЕ Б. ГРАФИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ СПИСКА.

Данная схема соответствует списку (a (b c))

