

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

Студентка гр. 7381

Кушкочева А.О.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2018

Цель работы.

Познакомиться с иерархическими списками и использованием их в практических задачах на языке программирования C++.

Формулировка задачи.

Сформировать линейный список атомов исходного иерархического списка таким образом, что скобочная запись полученного линейного списка будет совпадать с сокращенной скобочной записью исходного иерархического списка после устранения всех внутренних скобок.

Основные теоретические положения.

Определим соответствующий тип данных $S_expr(El)$ рекурсивно, используя определение линейного списка (типа L_list):

$$\langle S_expr(El) \rangle ::= \langle Atomic(El) \rangle \mid \langle L_list(S_expr(El)) \rangle,$$
$$\langle Atomic(E) \rangle ::= \langle El \rangle.$$
$$\langle L_list(El) \rangle ::= \langle Null_list \rangle \mid \langle Non_null_list(El) \rangle$$
$$\langle Null_list \rangle ::= Nil$$
$$\langle Non_null_list(El) \rangle ::= \langle Pair(El) \rangle$$
$$\langle Pair(El) \rangle ::= (\langle Head_l(El) \rangle . \langle Tail_l(El) \rangle)$$
$$\langle Head_l(El) \rangle ::= \langle El \rangle$$
$$\langle Tail_l(El) \rangle ::= \langle L_list(El) \rangle$$

Переход от полной скобочной записи к сокращенной производится путем отбрасывания внутренних скобок.

Реализация задачи.

Базовым типом данных для данной задачи является тип `char`. Для реализации иерархического списка использовались две структуры: `struct s_expr` и `struct two_ptr`. Структура `struct two_ptr` содержит в себе два указателя `s_expr *hd` и `s_expr *tl`. Структура `struct s_expr` содержит переменную `bool tag`, которая в зависимости от того, является элемент атомом или подписанием списка присваивает значение `true` и `false` соответственно. Также эта структура содержит объединение двух типов, `base atom` и `two_ptr pair`.

Функции-селекторы: `head` и `tail`, выделяющие «голову» и «хвост» списка соответственно. Если «голова» списка не атом, то функция `head` возвращает список, на который указывает голова пары, т.е. подсписок, находящийся на следующем уровне иерархии. Если же «голова» списка — атом, то выводится сообщение об ошибке и функция прекращает работу. Функция `tail` работает аналогично функции `head`, но только для «хвоста».

Функции-конструкторы: `cons`, создающая точечную пару (новый список из «головы» и «хвоста»), и `make_atom`, создающая атомарное выражение. При создании нового выражения требуется выделение памяти. Если памяти нет, то `p == NULL` и это приводит к выводу соответствующего сообщения об ошибке. Если «хвост» — не атом, то для его присоединения к «голове» требуется создать новый узел (элемент), головная ссылка которого будет ссылкой на «голову» этого «хвоста», а хвостовая часть элемента — ссылкой на его «хвост»

Функции-предикаты: `isNull`, проверяющая список на отсутствие в нем элементов, и `isAtom`, проверяющая, является ли список атомом. Если элемент — атом, тогда функция возвращает значение `test`, которое равно `true`, и значение `False`, если «голова-хвост». В случае пустого списка значение предиката `False`.

Функции-деструкторы: `delete` и `destroy`. Функция `delete` удаляет текущий элемент из списка, а функция `destroy` удаляет весь список путем вызова функции `delete` и вызова самой себя.

Функция `getAtom` возвращает нам значение атома.

Функция `copy_lisp` — функция копирования списка.

Функция `concat` — функция для соединения двух списков. Создает новый иерархический список из копий атомов, входящих в соединяемые списки.

Функция `l_list` — функция для выравнивания иерархического списка, то есть формирования из него линейного списка путем удаления из сокращенной скобочной записи иерархического списка всех внутренних скобок.

Функция `write_lisp` — функция вывода списка с внешними скобками.

Функция `write_seq` – функция вывода последовательности элементов списка без скобок.

Тестирование.

Программа была собрана в компиляторе `g++` в OS Linux Ubuntu 16.04 LTS.

В ходе тестирования ошибки не были найдены.

Некорректные случаи представлены в табл. 1, где была написана неправильная форма записи иерархического списка.

Таблица 1 — Некорректные случаи в синтаксисе.

Входные данные	Результат
)dqwqdIncorrect	Incorrect using a closing bracket
(ask)(da)	Enter the list: (ask)(da) Input list: a s k Linear list = (a s k) (s k) (k) (a s k) Freeing memory: End!

Во втором примере вторая внешняя скобка не рассматривается, так как не объединена с первой.

Корректные тестовые случаи представлены в приложении А.

Выводы.

В ходе работы были получены навыки работы с иерархическими списками. Поскольку структура иерархического списка определена рекурсивно, рекурсивный подход является простым и удобным способом поиска решения.

ПРИЛОЖЕНИЕ А. ТЕСТОВЫЕ СЛУЧАИ

Таблица 2 — Корректные тестовые случаи

Входные данные	Результат
(a (f f g d (f w e f w r (q e w r))))	<p>Test 3: (a(ffgd(fwefwr(qewr))))</p> <hr/> <p>Testing:</p> <p>Enter the list: Input list: a (f f g d (f w e f w r (q e w r))) Linear list = (a (f f g d (f w e f w r (q e w r)))) ((f f g d (f w e f w r (q e w r)))) (f f g d (f w e f w r (q e w r))) (f g d (f w e f w r (q e w r))) (g d (f w e f w r (q e w r))) (d (f w e f w r (q e w r))) ((f w e f w r (q e w r))) (f w e f w r (q e w r)) (w e f w r (q e w r)) (e f w r (q e w r)) (f w r (q e w r)) (w r (q e w r)) (r (q e w r)) ((q e w r))</p>

	<p>(q e w r)</p> <p>(e w r)</p> <p>(w r)</p> <p>(r)</p> <p>(a f f g d f w e f w r q e w r)</p> <p>Freeing memory:</p> <p>End!</p>
(q w d q (d q w d q d q w (D q w q d q)))	<p>(qwdq(dqwqdqw(Dqwqdq)))</p> <hr/> <p>Testing:</p> <p>Enter the list:</p> <p>Input list:</p> <p>q w d q(d q w q d q w(D q w q d q))</p> <p>Linear list =</p> <p>(q w d q(d q w q d q w(D q w q d q)))</p> <p>(w d q(d q w q d q w(D q w q d q)))</p> <p>(d q(d q w q d q w(D q w q d q)))</p> <p>(q(d q w q d q w(D q w q d q)))</p> <p>((d q w q d q w(D q w q d q)))</p> <p>(d q w q d q w(D q w q d q))</p> <p>(q w q d q w(D q w q d q))</p> <p>(w q d q w(D q w q d q))</p> <p>(q d q w(D q w q d q))</p>

	(d q w(D q w q d q)) (q w(D q w q d q)) (w(D q w q d q)) ((D q w q d q)) (D q w q d q) (q w q d q) (w q d q) (q d q) (d q) (q) (q w d q d q w q d q w D q w q d q) Freeing memory: End!
(((((a))))))	((((a)))) <hr/> Testing: Enter the list: Input list: ((((a)))) Linear list = ((((a)))) ((((a)))) (((a))) ((a)) (a) (a) Freeing memory: End!

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <cstdlib>
#include "list.hpp"
using namespace std;

using namespace h_list;
lisp concat(const lisp y, const lisp z);
lisp l_list(const lisp s);

int main() {
    lisp s1, s2;
    cout << "Enter the list:" << endl;
    read_lisp(s1);
    cout << "Input list: " << endl;
    write_seq(s1);
    cout << endl;

    cout << "Linear list = " << endl;
    s2 = l_list(s1);
    write_lisp(s2);
    cout << endl;
    cout << "Freeing memory: " << endl;
    destroy(s2);
    cout << "End!" << endl;
    return 0;
}

lisp concat(const lisp y, const lisp z) {
    if (isNull(y)) return copy_lisp(z);
    else return cons(copy_lisp(head(y)), concat(tail(y), z));
}

lisp l_list(const lisp s) {
    if (isNull(s)) return NULL;
    else if (isAtom(s)) {

        cout << s->node.atom << endl;
        return cons(make_atom(getAtom(s)), NULL);
    }
    else //s - not empty list
    if (isAtom(head(s))) {
        cout << "(";
        write_seq(s);
        cout << ")" << endl;
    }
}
```

```

        cout << endl;
        return cons(make_atom(getAtom(head(s))), l_list(tail(s)));
    }
    else {
        cout << "(";
        write_seq(s);
        cout << ")" << endl;
        cout << endl;
        return concat(l_list(head(s)), l_list(tail(s)));
    }
}

namespace h_list {
lisp head(const lisp s) {
    if (s != NULL)
        if (!isAtom(s))      return s->node.pair.head;
        else
        {
            cout << "Atom\n";
            exit(1);
        }
    else
    {
        cout << "The list is empty\n";
        exit(1);
    }
}

bool isAtom(const lisp s) {
    атомарен ли список
    if (s == NULL) return false;
    else return (s->test);
}

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

lisp tail(const lisp s) {
    if (s != NULL)
        if (!isAtom(s))
            return s->node.pair.tail;
        else
        {
            cout << "The list is empty\n";
            exit(1);
        }
    }

lisp cons(const lisp h, const lisp t) {

```

//функция проверяет

//функция проверяет список на

```

lisp p;
if (isAtom(t))
{
    cout << "Atom\n";
    exit(1);
}
else {
    p = new s_expr;
    if (p == NULL) {
        cerr << "Not enough memory\n";
        exit(1);
    }
    else {
        p->test = false;
        p->node.pair.head = h;
        p->node.pair.tail = t;
        return p;
    }
}
}
}
lisp make_atom(const char x) {
lisp s;
s = new s_expr;
s->test = true;
s->node.atom = x;
return s;
}

void destroy(lisp s) {                                     //функция освобождения
    памяти
    if (s != NULL) {
        if (!isAtom(s)) {
            destroy(head(s));
            destroy(tail(s));
        }
        delete s;
    };
}

char getAtom(const lisp s) {                               //функция дает
    возможность получить значение атомарного выражения
    if (isAtom(s))
    {
        cout << " Error - it's not an atom \n";
        exit(1);
    }
    else return (s->node.atom);
}

```

```

namespace h_list {                                     //создание структуры иерархического
списка
    struct s_expr;
    struct two_ptr {
        s_expr *head;                                //указатель на начало списка
        s_expr *tail;                                //указатель на конец списка
    };

    struct s_expr {
        bool test;                                    // если true: atom, иначе false: pair
        union {
            char atom;
            two_ptr pair;
        } node;
    };
    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 char x);
    bool isAtom(const lisp s);
    bool isNull(const lisp s);
    void destroy(lisp s);
    char getAtom(const lisp s);
    void read_lisp(lisp& y);
    void read_s_expr(char prev, lisp& y);
    void read_seq(lisp& y);
    void write_lisp(const lisp x);
    void write_seq(const lisp x);
    lisp copy_lisp(const lisp x);
} void read_lisp(lisp& y) {                             //функция считывания
списка
char x;
do
    cin >> x;
while (x == ' ');
read_s_expr(x, y);
}

void read_s_expr(char prev, lisp& y) {
//вспомогательная процедура для read_lisp
if (prev == ')')
{
    cout << " Incorrect using a closing bracket\n" << endl;
    exit(1);
}
else if (prev != '(') y = make_atom(prev);

```

```

else read_seq(y);
}

void read_seq(lisp& y) {                                     //вспомогательная процедура
для read_lisp
char x;
lisp p1, p2;
if (!(cin >> x))
{
    cout << "Error\n" << 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) {                                //функция вывода
списка с внешними скобками
if (isNull(x)) cout << "()";
else if (isAtom(x)) cout << ' ' << x->node.atom;
else {
    cout << "(";
    write_seq(x);
    cout << ")";
}
}

void write_seq(const lisp x) {                                  //функция вывода
последовательности элементов списка без скобок
if (!isNull(x)) {
    write_lisp(head(x));
    write_seq(tail(x));
}
}

lisp copy_lisp(const lisp x) {
if (isNull(x)) return NULL;
else if (isAtom(x)) return make_atom(x->node.atom);
else return cons(copy_lisp(head(x)), copy_lisp(tail(x)));
}

```

}

ПРИЛОЖЕНИЕ В. ОБЪЯВЛЕНИЕ СТРУКТУРЫ И ПРОТОТИПОВ ФУНКЦИЙ

```
namespace h_list {                                //создание структуры
иерархического списка
    struct s_expr;
    struct two_ptr {
        s_expr *head;                            //указатель на начало
списка
        s_expr *tail;                            //указатель на конец
списка
    };

    struct s_expr {
        bool test;                                // если true: atom, иначе
false: pair
        union {
            char atom;
            two_ptr pair;
        } node;
    };
    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 char x);
    bool isAtom(const lisp s);
    bool isNull(const lisp s);
    void destroy(lisp s);
    char getAtom(const lisp s);
    void read_lisp(lisp& y);
    void read_s_expr(char prev, lisp& y);
    void read_seq(lisp& y);
    void write_lisp(const lisp x);
    void write_seq(const lisp x);
    lisp copy_lisp(const lisp x);
}
```