

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

Зуев Д.В.

Преподаватель

Размочева Н.В.

Санкт-Петербург

2018

ОГЛАВЛЕНИЕ

Цель работы.....	3
Реализация задачи.....	3
Тестирование.....	5
Вывод.....	5
Приложение А. Тестовые случаи.....	6
Приложение Б. Исходный код программы.....	7

Цель работы

Цель работы: познакомиться с иерархическими списками, получить навыки использования их в практических задачах на языке программирования C++.

Формулировка задачи: проверить иерархический список на наличие в нем заданного элемента (атома) x ;

Реализация задачи

В данной работе используются структуры `two_ptr` и `s_expr`. В первой содержится указатель на голову списка и на его хвост. Во второй содержится флаг показывающий, является ли элемент атомом и объединение в котором хранится атом или пара (указатели на голову и хвост).

В функции `main` выводится приглашение выбрать способ ввода входных данных либо выйти из программы. В случае выбора файла, программа считывает текст из файла и записывает его в поток ввода, объявленный как глобальная переменная, после этого закрывает файл. В случае ввода информации с консоли функция `main` считывает строку с консоли, записывает эту строку в этот же поток ввода.

Если файл пустой, то `main` выводит ошибку и начинает выполнение программы заново. Иначе вызывает функцию `res`.

Функция `res` проверяет не является ли пустым список, если пустой то возвращает `false`. Если элемент является атомом то функция сравнивает его с искомым элементом, если атом является искомым, функция возвращает `true`. Если элемент пара, то функция возвращает логическую сумму значений применения самой себя к голове и хвосту пары.

Функция `head` проверяет, является ли полученный на вход элемент списка пустым, если является, тогда выводит ошибку и завершает программу, иначе проверяет, является ли элемент атомом, если является тогда функция выводит ошибку и завершает программу, иначе возвращает указатель на голову списка.

Функция `isAtom` проверяет, является ли полученный на вход элемент

списка атомом. Функция возвращает `s->tag`.

Функция `isNull` проверяет является ли полученный на вход элемент списка пустым. Если является то функция возвращает `true`, иначе `false`.

Функция `tail` проверяет, является ли полученный на вход элемент списка пустым, если является, тогда выводит ошибку и завершает программу, иначе проверяет, является ли элемент атомом, если является тогда функция выводит ошибку и завершает программу, иначе возвращает указатель на хвост списка.

Функция `cons` создает новый элемент списка из головы и хвоста полученных на вход. При получении на вход атома вместо хвоста или при недостатке памяти выдает ошибку.

Функция `make_atom` принимает на вход символ и создает элемент списка являющийся атомом.

Функция `destroy` удаляет все элементы списка вызывая рекуррентно саму себя для хвоста и для головы.

Функция `getAtom` возвращает символ, если полученный на вход элемент является атомом, иначе выводит ошибку.

Функция `read_lisp` считывает символы из потока ввода пока не встретит символ, не являющийся пробелом, после, если символ не пустой вызывает `read_s_expr`.

Функция `read_s_expr` выводит ошибку если символ полученный на вход является закрывающей скобкой, иначе вызывает `read_seq` для создания списка одного уровня, если символ является открывающей скобкой, вызывает `make_atom` для создания атома, если символ не является круглой скобкой.

Функция `read_seq` выводит ошибку если поток входных данных пуст. Создает пустой список, если следующий символ в потоке, не являющийся пробелом закрывающая скобка. В остальных случаях считывает голову функцией `read_s_expr`, а хвост функцией `read_seq`.

Функция `write_lisp` выводит список. Если элемент списка — атом то выводит его. Если элемент — пара, то выводит список внутри круглых скобок

функцией `write_seq`.

Функция `write_seq` выводит голову списка функцией `write_lisp`, а хвост функцией `write_seq`, если элемент списка полученный на вход не является пустым.

Тестирование

Программа была собрана в компиляторе G++ в среде разработки Qt creator в OS Linux Ubuntu 16.04 LTS.

В ходе тестирования была найдена ошибка. Программа не обновляла список при повторном вводе с клавиатуры.

Для исправления программы в главную функцию было добавлено обновление потока ввода.

Корректные тестовые случаи представлены в приложении А.

Вывод

В ходе работы были получены навыки работы с иерархическими списками на языке C++. Поскольку структура иерархического списка определяется рекурсивно, то и функции используемые для работы со списком рекурсивные.

ПРИЛОЖЕНИЕ А.

ТЕСТОВЫЕ СЛУЧАИ

Таблица 1 — Корректные случаи

Входные данные	Выходные данные
d 1 (qwer(sdf)sdf)	Введен list1: (q w e r (s d f) s d f) Произведен поиск элемента: Элемент X найден.
4	Введите верное число
d 2	Введен list1: (a s (f g (f g) () f j (d) e r) t r) Произведен поиск элемента: Элемент X найден.
3	
d 1 (qwer(sdf)sdf)	Введен list1: (q w e r (s d f) s d f) Произведен поиск элемента: Элемент X не найден.
s 1 ((d	Error: there is no closing bracket
e 1)dfg	Error: the initial brace is closing
t 1	Введен list1: () Произведен поиск элемента: Элемент X не найден.

ПРИЛОЖЕНИЕ Б.

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <sstream>
#include <fstream>

using namespace std;

stringstream xstream;
typedef char base;
struct s_expr;
struct two_ptr
{
    s_expr *hd;
    s_expr *tl;
};
struct s_expr
{
    bool tag;
    union
    {
        base 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 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);
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);

bool res(lisp s, char x){
    if (isNull(s)) return false;
    if (isAtom(s))
        return x == s->node.atom;
    else
        return res(tail(s),x)+res(head(s),x);
}

int main()
```

```

{
    bool b = 1;
    lisp s = NULL;
    char x;
    string str;
    char str0[100];
    short int tmp = 0;
    while(tmp != 3)
    {
        xstream.str("");
        xstream.clear();
        cout<<"Введите искомый элемент X\n";
        cin >> x;
        cout<<"Введите 1, если желаете вводить выражение с
клавиатуры.\n"
        "Введите 2, если желаете брать выражение из файла
test.txt.\n"
        "Введите, 3 если хотите закончить работу."<<endl;
        cin>>tmp;
        switch(tmp){
            case 1:
            {
                cout << "Введите list1: \n";
                cin.get();
                cin.getline(str0, 1000);
                xstream << str0;
                read_lisp(s);
                b = 1;
                break;
            }
            case 2:
            {
                ifstream outfile;
                outfile.open("test.txt");
                if (!outfile)
                {
                    cout << "Входной файл не открыт!\n";
                    b = 0;
                    break;
                }
                outfile.read(str0, 1000);
                outfile.close();
                xstream << str0;
                read_lisp (s);
                b = 1;
                break;
            }
            case 3:
                break;
            default:
            {
                cout<<"Введите верное число\n";

```



```

        break;
    }
}
if(b)
{
    cout << "Введен list1: \n";
    write_lisp (s);
    cout<<endl;
    cout << "Произведен поиск элемента: \n";
    if(res(s,x))
        cout<<"Элемент X найден.\n";
    else
        cout<<"Элемент X не найден.\n";
    destroy(s);
}
}
return 0;
}

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 -> tag);
}

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)
{
    lisp p;
    if (isAtom(t))
    {
        cerr << "Error: Tail is head \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;
    }
}

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
        xstream >> x;
    while (x==' ');
    if(x)
        read_s_expr ( x, y);
}

void read_s_expr (base prev, lisp& y)
{
    if ( prev == ')' )
    {
        cerr << "Error: the initial brace is closing\n";
        exit(1);
    }
    else
        if ( prev != '(' )
            y = make_atom (prev);
        else read_seq (y);
}

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

    if (!(xstream >> x))
    {
        cerr << "Error: there is no closing bracket\n";
        exit(1);
    }
}

```

```

else
{
    while ( x==' ' )
        xstream >> 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));
    }
}

```