

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: Иерархические списки

Студент гр. 7383

МЕДВЕДЕВ И. С.

Преподаватель

РАЗМОЧАЕВА Н. В.

Санкт-Петербург

2018

Содержание

Цель работы	3
Реализация задачи.....	4
Тестирование.....	7
Вывод	8
Приложение А. Код программы.....	9
Приложение Б. Тестовые случаи	15

1. Цель работы

Познакомиться с иерархическими списками и научиться реализовывать их на языке программирования C++.

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

2. Реализация задачи

В данной работе было написано несколько функций и структур для реализации задачи. Перечень функций и структур:

`struct two_ptr` – структура, которая содержит указатель на голову и на хвост списка.

`struct s_expr` – структура, которая содержит поля *bool tag*, который имеет значения `true`, если элемент списка атом, и поле `union node`. Определили эту структуру с помощью `typedef` как **lisp*.

`union node` – объединение, содержащее поля *char atom* и *two_ptr pair*. Поле `atom` хранит символ, если элемент списка атом, а поле `pair` хранит указатели на хвост и голову если элемент не атом.

`lisp head (const lisp s)` – функция, которая возвращает указатель на голову списка или выдает ошибку, если голова - атом или передан пустой список.

`bool isAtom (const lisp s)` – возвращает значение *s->tag*, тем самым показывает, является ли элемент атомом.

`bool isNull (const lisp s)` – функция получает на вход список и проверяет является ли он пустым.

`lisp tail (const lisp s)` – функция, которая возвращает указатель на хвост списка или выдает ошибку, если хвост - атом или передан пустой список.

`lisp cons (const lisp h, const lisp t)` – функция, получает на вход два списка и «склеивает» второй с первым. Если второй аргумент – атом, сообщает об ошибке.

`lisp make_atom (const char x)` – функция получает на вход элемент создает список, в который записывает этот элемент. Значение *s->tag* этого списка становится `true`, а *s->node.atom* становится равным переданному элементу.

`void destroy (lisp s)` – функция получает на вход список и удаляет его.

`void read_lisp (lisp& y, istream &list)` – получает на вход ссылку на список и элемент типа *istream*. Функция считывает посимвольно из *list*, проверяя на пробелы. Встретив символ отличающийся от ' ', запускает программу `read_s_expr`, передавая ему этот символ.

`void read_s_expr (char prev, lisp& y, istream &list)` – функция получает на вход символ, который встретила функция `read_lisp`, ссылку на список и элемент типа *istream*. Если *prev* равен `') '`, то функция сообщает об ошибке. Если *prev* не равен `' ('`, то делает из *prev* атом, и приравнивает *y* к этому атому. В других случаях, функция вызывает `read_seq`.

`void read_seq (lisp& y, istream &astr)` – функция получает на вход ссылку на список и элемент типа *istream*. В начале функция создает списки *p1* и *p2*, считывает все пробелы, игнорируя их. Если встречает закрывающую скобку приравнивает переданный список к *NULL*. Иначе же запускает функцию `read_s_expr` для первого символа отличного от пробела, списка *p1* и элемента *istream*, с которого считываются символы, затем вызывает функцию `read_seq` для списка *p2*, и элемента *istream*, затем склеивает списки *p1* и *p2*.

`void write_lisp (const lisp x)` – функция получает на вход список, если он пустой, то выводит `"()"`, если он состоит из одного атома, то выводит его значение в скобках, иначе выводит `' ('`, запускает функцию `write_seq` и выводит `') '`.

`void write_seq (const lisp x)` – получает на вход список, вызывает функцию `write_lisp` для головы списка и саму себя для хвоста.

`lisp rev(const lisp s,const lisp z)` – функция получает на вход два списка. Если первый пустой, то возвращает второй список. Если голова первого списка – атом, то запускает саму же себя для хвоста первого списка и «склейки» головы первого списка со вторым списком и возвращает это значение. Иначе запускает саму себя же для хвоста первого списка и склейки двух списков, первым является запуск функции `rev` для головы первого списка и пустого списка, вторым – список *z*.

`int main ()` – головная функция, в которой считывается значение переменной *forSwitch*, для оператора ветвления `switch`. Так же считывается строка, введенная пользователем. В зависимости от выпора пользователя запускается тот или иной алгоритм. Если значение равно единице, то считывается строка, эта строка записывается в буфер и запускается функция

start. Если значение равно двум, то считывается строка из файла и записывается в буфер, запускается функция start. Значение равно нулю – программа завершается. При других значениях сообщается о неверном вводе.

int start(lisp& s1, lisp& s2, istream &astr) – функция создана для избежания дублирования кода. Передается два списка, считанный и пустой, а так же элемент типа *istream*. Функция выводит первый список, переворачивает его и записывает во второй, а затем выводит второй список. В конце удаляет списки.

3. Тестирование

Программа собрана в операционной системе Ubuntu 17.04 с использованием компилятора g++. В других ОС и компиляторах тестирование не проводилось. Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении Б.

4. Вывод

В ходе выполнения лабораторной работы были изучены основные понятия об иерархических списках, получены навыки создания иерархических списков и функций работы со списками на языке программирования C++. Также была написана программа для обращения иерархического списка.

ПРИЛОЖЕНИЕ А.

КОД ПРОГРАММЫ

```
struct
s_expr;

    struct two_ptr{
        s_expr *head;
        s_expr *tail;
    };

    struct s_expr {
        bool tag;
        union{
            char atom;
            two_ptr pair;
        }node;
    };

typedef s_expr *lisp;
using namespace std;
#include <fstream>
#include <iostream>
#include <sstream>
#include <cstdlib>
#include <cstring>

lisp reverse(const lisp s);

lisp rev(const lisp s,const lisp z);

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, istream &astr);

void read_s_expr (char prev, lisp& y, istream &astr);

void read_seq ( lisp& y, istream &astr);

void write_lisp (const lisp x);

void write_seq (const lisp x);

lisp head (const lisp s){
    if (s != NULL)
        if (!isAtom(s))
            return s->node.pair.head;
        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.tail;
        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: Cons(*, atom)\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.head = h;
            p->node.pair.tail = t;
            return p;
        }
    }
}

lisp make_atom (const char 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;
    }
}

```

```

void read_lisp ( lisp& y, istream &list){
    char x;
    do{
        list >> x;
    }while (x==' ');
    read_s_expr ( x, y, list);
}

void read_s_expr (char prev, lisp& y, istream &list){
    if ( prev == ')' ) {
        cerr << " ! Закрывающая скобка перед открывающей ! " << endl;
        exit(1);
    }
    else if ( prev != '(' )
        y = make_atom (prev);
    else
        read_seq (y, list);
}

void read_seq ( lisp& y, istream &list){
    char x;
    lisp p1, p2;
    if (!(list >> x)) {
        cerr << " ! Не хватает символов ! " << endl;
        exit(1);
    }
    else {
        while ( x==' ' ){
            list >> x;
        }

        if ( x == ')' )
            y = NULL;
        else {
            read_s_expr ( x, p1, list);
            read_seq ( p2, list);
            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 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)));
}

int start( lisp& s1, lisp& s2, istream &astr){
    char ch;
    read_lisp (s1, astr);
    cout << "введен list1: " << endl;
    write_lisp (s1);
    cout << endl;
    if(astr >> ch){
        cout<<"! Лишние символы !"<<endl;
        return 0;
    }
    cout << "Список перевернут: " << endl;
    s2 = rev (s1, NULL);
    write_lisp (s2);
    cout << endl;
    delete(s2);
    delete (s1);
    return 0;
}

int main ( ){
    char forSwitch;
    stringbuf exp;
    string list;
    char ch;
    lisp s1, s2;
    while (1){
        cout << "Нажмите 1 для ввода с клавиатуры, 2 для ввода с файла, 0 для
выхода"<<endl;
        cin>>forSwitch;
        cin.ignore();
        switch(forSwitch){
            case '1':{
                cout << "введите list1:" << endl;
                getline(cin, list);
                istream str(&exp);
                exp.str(list);
                start(s1,s2,str);
                break;
            }
            case '2':{
                ifstream infile("list.txt");
                getline(infile, list);

```

```

        istream str(&exp);
        exp.str(list);
        start(s1,s2,str);
        break;
    }
    case '0':{
        cout<<"Bye!"<<endl;
        return 0;
    }
    default:{
        cout<<"НЕВЕРНЫЙ ВВОД!"<<endl;
        break;
    }
}

}

}

```

ПРИЛОЖЕНИЕ Б.

ТЕСТОВЫЕ СЛУЧАИ

Таблица 1 — Результаты тестов.

Input	Output	True/False
((qwe)rty(w))	((w) y t r (e w q))	True
(q	! Не хватает символов !	True
(Q)	(Q)	True
A	Error: Head (atom)	True
(qwe(ert(tr)))	(((r t) t r e) e w q)	True
(qwer)(! Лишние символы !	True
()	()	True