МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЁТ

по лабораторной работе №2
по дисциплине «Алгоритмы и структуры данных»
Тема: Иерархические списки

Студентка гр. 7383 _____ Иолшина В.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2018

Содержание

Цель работы.

Цель работы: познакомиться с иерархическими списками и их рекурсивной реализацией на языке программирования C++.

Формулировка задачи: удалить из иерархического списка все вхождения заданного элемента (атома) x;

Реализация задачи.

В данной работе используются структуры two_ptr и s_expr. В первой из них содержится указатель на голову и на хвост списка. Во второй структуре содержится тэг, показывающий, является элемент атомом или структурой two_ptr, и объединение, в котором хранится значение атома или пара (структура two_ptr, содержащая указатели на голову и хвост).

В функции main выводится приглашение выбрать способ ввода входных данных или закончить работу. В случае выбора ввода данных из файла, программа считывает текст из файла text.txt и записывает его в поток ввода, после этого закрывает файл. В случае выбора ввода информации с консоли, функция main считывает строку и записывает ее в этот же поток ввода. Если файл пустой, то main выводит ошибку и начинает выполнение программы заново. Если все было выбрано верно, происходит вызов функции del_atom.

Функция del_atom проверяет, является ли список пустым, в случае, если он пустой, возвращает указатель на NULL. Если элемент является атомом, то функция сравнивает его с искомым элементом, если атом атом таковым не является, функция возвращает указатель на данный атом. Если атом является искомым, функция удаляет его и возвращает указатель на NULL. Далее функция рекурсивно вызывает саму себя для своей головы и хвоста. В случае, если голова очередного элемента указывает на NULL, функция возвращает хвост списка, в противном случае возвращает указатель на данный элемент.

Функция head проверяет, является ли полученный на вход элемент списка пустым, если является, тогда выводит ошибку и начинает выполнение

программы заново, иначе возвращает указатель на голову списка.

Функция isAtom проверяет, является ли полученный на вход элемент списка атомом. Возвращает значение тэга.

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

Функция tail проверяет, является ли полученный на вход элемент списка пустым, если является, тогда выводит ошибку и начинает выполнение программы заново, иначе проверяет, является ли элемент атомом, в случае, 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 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 рекурсивно саму себя для хвоста и для головы.

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.

Процесс тестирования.

Программа собрана в операционной системе Ubuntu 16.04.2 LTS", с использованием компилятора g++ (Ubuntu 5.4.0-6ubuntu1~16.04.5). В других ОС и компиляторах тестирование не проводилось.

Результаты тестирования

Тестовые случаи представлены в Приложении А.

Во время тестирования была обнаружена ошибка: программа падала, если список начинался с искомого элемента. Для этого в функцию del_atom была добавлена проверка, указывает ли голова списка на NULL. В случае, если указывает, возвращается хвост списка.

Выводы.

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.

ИСХОДНЫЙ КОД ПРОГРАММЫ

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

using namespace std;

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, stringstream &xstream);
void read_s_expr (base prev, lisp& y, stringstream &xstream);
```

```
void read_seq (lisp& y, stringstream &xstream);
void write_lisp (const lisp x);
void write_seq (const lisp x);

lisp  del_atom(lisp s, char x)
{
	if (isNull(s)) return NULL;
	if (isAtom(s))
	{
		if(s->node.atom != x)
			return s;
		else
		{
			delete s;
			return NULL;
		}
	}
	s->node.pair.hd = del_atom(head(s),x);
	s->node.pair.tl = del_atom(tail(s),x);
	return ((head(s) == NULL)? tail(s) : s);
}

int main()
{
	stringstream xstream;
	bool b = 1;
	lisp s = NULL;
	char x;
	char str[100];
	int c = 0;
	while(c != 3)
	{
		cout << "Введите 1, если хотите ввести выражение с клавиатуры.\n"
			"Введите 2, если еспользовать выражение из файла test.txt.\n"
			"Введите 3, если хотите закончить работу." << endl;
		cin >> c;
		switch(c)
		{
			case 1:
```

```
        {
            cout<<"Введите искомый элемент X\n";
            cin >> x;
            cout << "Введите выражение: \n";
            cin.get();
            cin.getline(str, 1000);
            xstream << str;
            read_lisp(s, xstream);
            break;
        }
        case 2:
        {
            cout<<"Введите искомый элемент X\n";
            cin >> x;
            ifstream outfile;
            outfile.open("test.txt");
            if (!outfile)
            {
                cout << "Входной файл не открыт!\n";
                b = 0;
                break;
            }
            outfile.read(str, 1000);
            outfile.close();
            xstream << str;
            read_lisp (s, xstream);
            break;
        }
        case 3:
        {
            b=0;
            break;
        }
        default:
        {
            cout<<"Введите верное число\n";
            break;
        }
}
```

```
            if(b)
            {
                cout << "Введен list1: \n";
                write_lisp (s);
                cout<<endl;
                cout << "Произведен поиск элемента: " << x << endl;
                if(s = del_atom(s,x))
                {
                    cout << "Элемент X удален.\n";
                    write_lisp(s);
                    cout << endl;
                }
                else
                    cout << s << "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, stringstream &xstream)
{
	base x;
	do
		xstream >> x;
	while (x==' ');
		if(x)
			read_s_expr (x, y, xstream);
}

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

void read_seq (lisp& y, stringstream &xstream)
{
	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, xstream);
                read_seq (p2, xstream);
                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));
        }
```

ПРИЛОЖЕНИЕ Б.

ТЕСТОВЫЕ СЛУЧАИ

Ввод	Вывод	Верно
1 d (qwer(sdf)sdf)	Введен list1: (q w e r (s d f) s d f) Произведен поиск элемента: d Результат работы программы (q w e r (s f) s f)	Да
1 x (a s (f g (f g) () f j (d) e r) t r)	Введен list1: (a s (f g (f g) () f j (d) e r) t r) Произведен поиск элемента: x Элемент X не найден.	Да
1 g ((f	Error: there is no closing bracket	Да
1 k)dfg	Error: the initial brace is closing	Да
1 h	Введен list1: () Произведен поиск элемента: h Элемент X не найден.	Да
4	Введите верное число	Да
1 d (d d d g (l d n) s f)	Введен list1: (d d d g (l d n) s f) Произведен поиск элемента: d Результат работы программы (g (l n) s f)	Да
1 D (d g g)	Введен list1: (d g g) Произведен поиск элемента: d Результат работы программы (g g)	Да