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

ОТЧЕТ

по практической работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Иерархические списки

Студент гр. 7382 _____ Находько А.Ю.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2018

Цель работы.

Ознакомиться с основными понятиями и приёмами рекурсивной обработки иерархических списков, получить навыки решения задач обработки иерархических списков.

Задание.

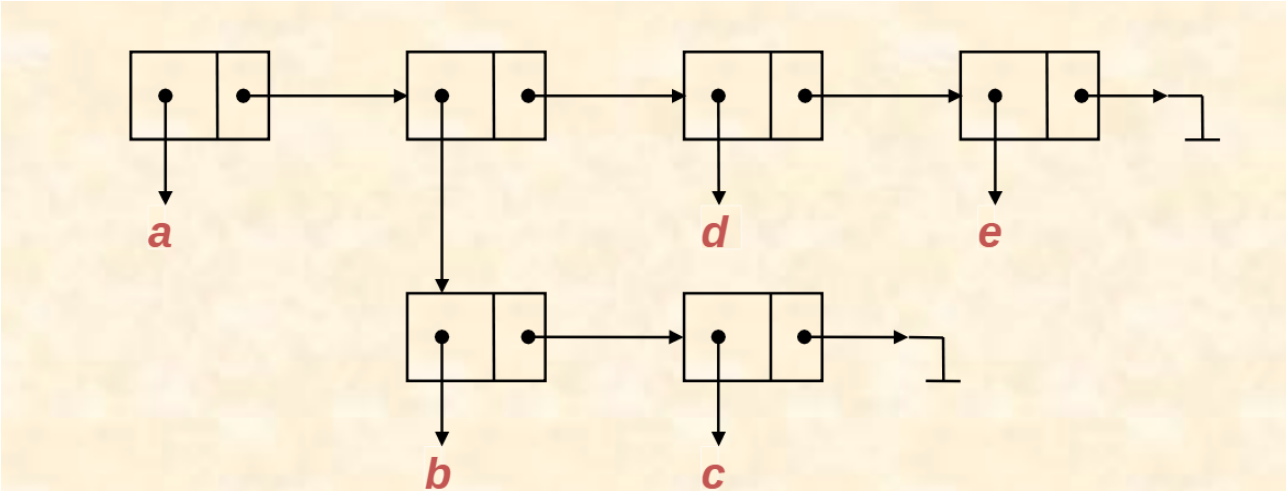
Вариант 8. Заменить в иерархическом списке все вхождения заданного элемента (атома) x на заданный элемент (атом) y;

Пояснение задачи.

Требуется, чтобы программа исходя из выбранных пользователем пунктов работы, корректно осуществляла эти действия. Производила замену в иерархическом списке всех вхождений заданного элемента на выбранный пользователем элемент.

Основные теоретические положения.

Иерархические списки могут быть представлены графически или в виде скобочной записи.



На данном рисунке изображён список: (a(b)(c)de). Переход от полной к сокращённой записи производится путём отбрасывания внутренних скобок.

Выполнение работы.

1) Заведём переменные под заменяемый элемент, его заменяющий, список, переменную действия и переменную flag, которая определяет продолжение выполнения программы.
2) Создадим меню с помощью оператора switch, исходя из значения которое поступает в него – выбирается действие для выполнения. Из возможных действий: можем загрузить текстовый файл для замены, либо замена в заданном нами вручную списке.
3) Замена в загружаемом файле осуществляется предварительным выбором элементов, пока наш список не закончится будем выполнять считывание списка функцией read_lisp, его запись в первоначальном виде функцией write_lisp, передачу списка в функцию Swap для осуществления замены требуемых элементов. После успешного выполнения задачи – удалим писок с помощью функции destroy.
4) Если мы хотим ввести список для замены сами, выбираем пункт 2 в меню “ручная работа с файлом”. На данном этапе можем осуществить следующие действия: считать новый иерархический список используя функцию read_lisp, удалить список с помощью функции destroy, осуществить требуемую в задании замену элементов с помощью написанной рекурсивной функции Swap, вывести список с помощью функции write_lisp (при условии что он не пустой), либо выйти из программы.
5) После каждого проделанного действия программа будет спрашивать у пользователя – хочет ли он продолжить работу с программой, либо хочет выйти.

Описание функций.

Swap – рекурсивная функция, которая в качестве аргументов принимает исходный список, его копию для вывода промежуточных результатов, изменяемый и подставляемый элемент. Функция изначально устанавливает, не пуст ли наш список. Перебор списка осуществляется с помощью функций list, tale. Если текущим элементом оказывается изменяемый, то заменяем его на

подставляемый. Также при изменении исходного списка выводим промежуточные результаты.

act_to_perform – функция, которая определяет действие которое мы хотим совершить с введённым пользователем списком. Функция устанавливает корректность выбранного действия, если данное действие не предусмотрено, то просим выбрать существующее действие.

isAtom – на вход принимается иерархический список, функция проверяет поступивший список на атомарность. Если s=NULL, то возвращается false.

write_lisp – функция вывода списка.

head – функция, которая возвращает указатель на голову списка, если он пуст, то возвращает сообщение об этом.

tail – функция, которая возвращает указатель на хвост списка, если он пуст, то возвращает сообщение об этом.

read_lisp – подаём на вход данные с клавиатуры, на выходе получим иерархический список.

destroy – на вход принимает иерархический список, производит рекурсивное освобождение памяти, очищая то голову, то хвост, в итоге этого метода удалятся все элементы списка. Возвращаемое значение отсутствует.

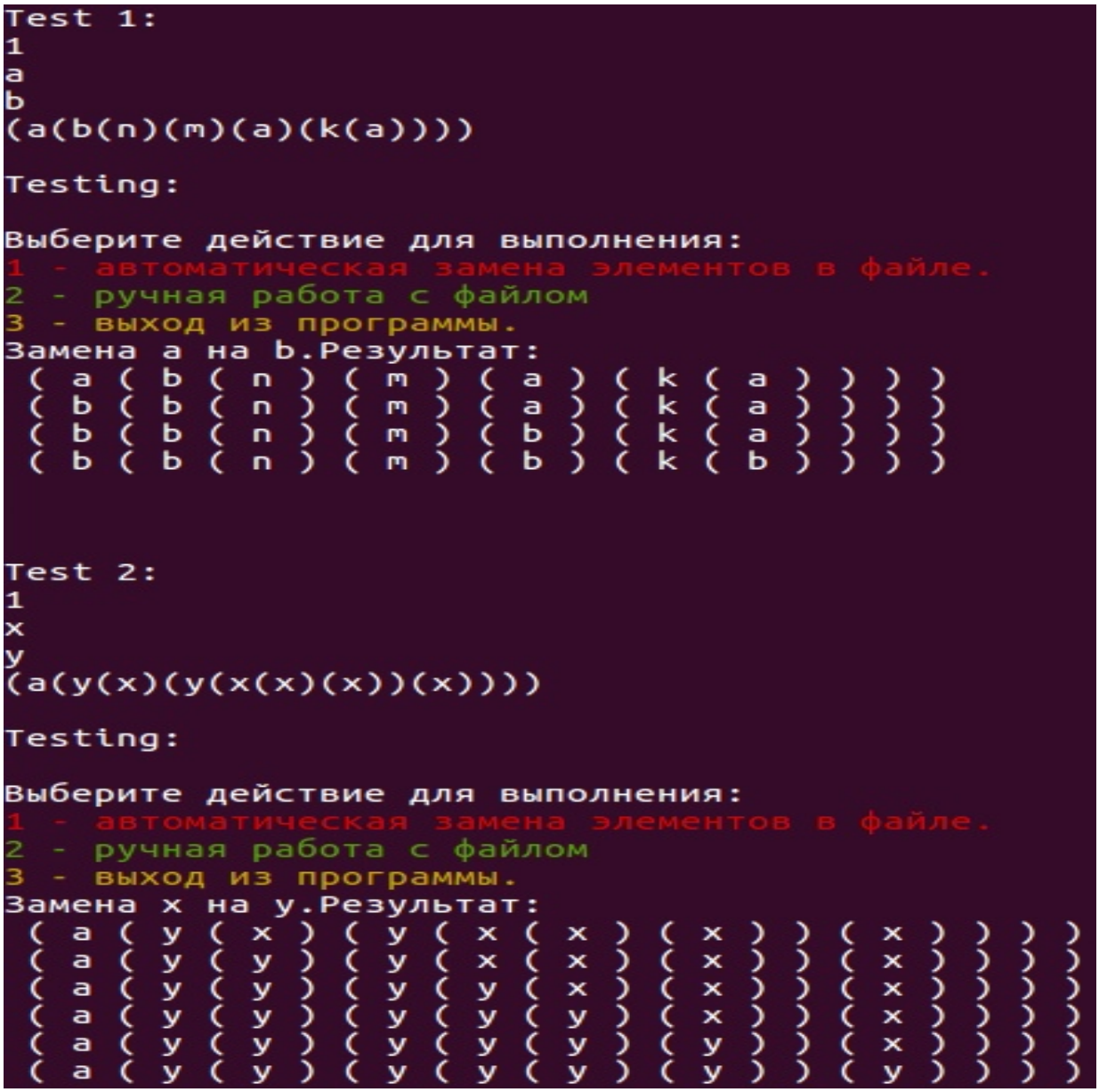
Тестирование.

Таблица тестирования программы.

№ теста	**Входные данные:**	**Результат:**
1	(a(b(n)(m)(a)(k(a))))	(b(b(n)(m)(b)(k(b))))
2	(a(y(x)(y(x(x)(x))(x))))	(a(y(y)(y(y(y)(y)(y))))
3	(b(b(b(b))))	(b(b(b(b))))
4	(a(m(a)))	(b(m(b)))
5	(l(o(d(f(f)))))	(l(o(d(k(k)))))
6	(f(f(a))(f))	(g(g(a))(g))

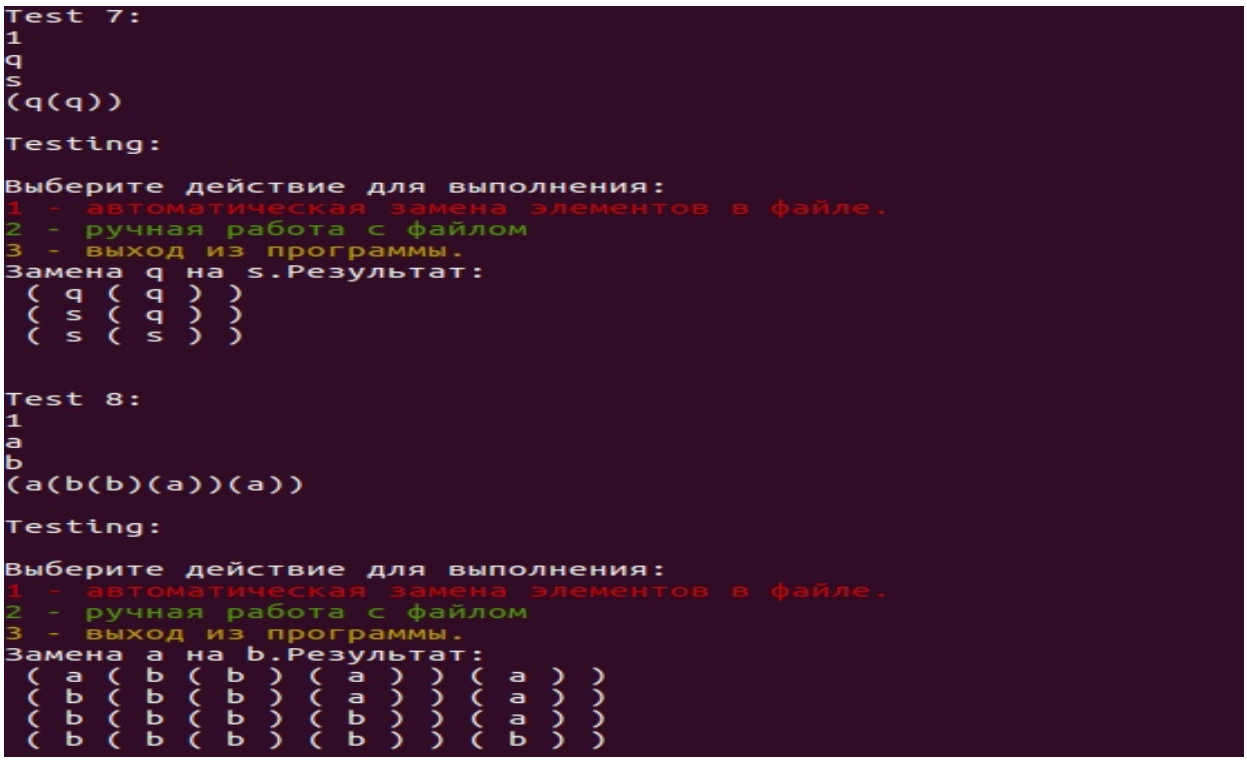
7	(q(q))	(s(s))
8	(a(b)(b)(a))(a))	(b(b)(b)(b))(b))

Иллюстрация работы тестирования:



```
Test 3:
1
a
b
(b(b(b(b))))

Testing:

Выберите действие для выполнения:
1 - автоматическая замена элементов в файле.
2 - ручная работа с файлом
3 - выход из программы.
Замена a на b.Результат:
 ( b ( b ( b ( b ) ) ) )
```

```
Test 4:
1
a
b
(a(m(a)))

Testing:

Выберите действие для выполнения:
1 - автоматическая замена элементов в файле.
2 - ручная работа с файлом
3 - выход из программы.
Замена a на b.Результат:
 ( a ( m ( a ) ) )
 ( b ( m ( a ) ) )
 ( b ( m ( b ) ) )


Test 5:
1
f
k
(l(o(d(f(f)))))

Testing:

Выберите действие для выполнения:
1 - автоматическая замена элементов в файле.
2 - ручная работа с файлом
3 - выход из программы.
Замена f на k.Результат:
 ( l ( o ( d ( f ( f ) ) ) ) )
 ( l ( o ( d ( k ( f ) ) ) ) )
 ( l ( o ( d ( k ( k ) ) ) ) )


Test 6:
1
f
g
(f(f(a))(f))

Testing:

Выберите действие для выполнения:
1 - автоматическая замена элементов в файле.
2 - ручная работа с файлом
3 - выход из программы.
Замена f на g.Результат:
 ( f ( f ( a ) ) ( f ) )
 ( g ( f ( a ) ) ( f ) )
 ( g ( g ( a ) ) ( f ) )
 ( g ( g ( a ) ) ( g ) )
```



Описание алгоритма

На примере теста №6.

1) В меню работы с программой выбираем пункт “ручная работа с файлом”. Тем самым переходим в опратор switch со значением 2.

2) Запускается второе меню, с помощью которого, выбрав действие со значением 1 введём список для осуществления преобразования. Считывание списка осуществляется с помощью предоставленной к л\р функцией read_lisp. Введём список: **(f (f (a) (f)))** .

3) Выбрав действие №3, выберем элемент для замены “f” и элемент на который будем его заменять, в данном случае элемент “g”.

4) Замена элементов в списке осуществляется с помощью написанной рекурсивной функцией Swap. В качестве аргументов функция Swap примет исходный список, копию списка для вывода промежуточных результатов, а также заменяемые элементы “f” и “g”.

5) Сперва проверим список на пустоту, если он пустой, то происходит выход из функции. В нашем случае список не пуст, поэтому двигаемся дальше по функции. Если текущий элемент оказывается “f”, то проводим его замену на элемент “g”. Также, если происходит замена, то выводим промежуточный

результат на консоль. Обход по списку осуществляется рекурсивно, посредством вызовов исходных функций head и tail, значения которых передаются в функцию Swap вместе с заменяемыми элементами и копией списка. Когда весь список будет просмотрен и все искомые элементы заменены, происходит выход из функции.

6) Можем убедиться в успешном результате выполнения задачи выведя список на экран с помощью исходной функции write_lisp. После выполнения программы можем удалить найденный список выбрав в меню соответствующее действие. Удаление производится вызовом исходной функции destroy.

Выводы.

В результате выполнения лабораторной работы ознакомился с основными понятиями и приёмами рекурсивной обработки иерархических списков. Получил навыки решения задач обработки иерархических списков. Написал рекурсивную функцию замены одного элемента списка на другой, создал удобное меню для диалога с пользователем. Ознакомился с представленными базовыми функциями обработки иерархических списков.

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

```
#include <stdio.h>
#include <iostream>
#include <cstdlib>
#include "l_intrfc.h"
#include <clocale>
#define COL "\033[0;31m"
#define COL1 "\033[0;32m"
#define COL2 "\033[0;33m"
#define COL3 "\033[0;34m"
#define COL4 "\033[0;35m"
#define NONE "\033[0m"

using namespace std;
using namespace h_list;

//Рекурсивная функция замены в исходном списке задаваемый элемент "a" на "b".
//В качестве аргументов принимает: список, задаваемый элемент "a", "b" и копию списка для вывода промежуточный результатов.
void Swap(lisp list, const base &a, const base &b, lisp s)
{
if (isNull(list)) //Проверяем список на пустоту
return;
else if (isAtom(list))
{
if(list->node.atom == a) //Входим в этот блок, если текущий элемент надо заменить
{
list->node.atom = b; //Заменяемый элемент
write_lisp(s); //Вывод промежуточных результатов
cout « endl;
}
}
else //Перебор элементов
```

```
{
Swap(head(list), a, b, s);
Swap(tail(list), a, b, s);
}
return;
}

int act_to_perform()
{
while (true)
{
int x; //Показатель действия
printf("Выберите действие для выполнения:\n%s1 - ввести список;%s\n%s2 - удалить список;
%s\n%s3 - осущетвить замену элементов;%s\n%s4 - вывести список на экран%s\n%s5 -выйти
из программы.%s\n", COL, NONE, COL1, NONE, COL2, NONE, COL3, NONE, COL4,
NONE);
cin » x;
if ((x<1)|| (x>5)) //Проверка показателя на корректность
{
printf("Некорректно введённые данные, попробуйте ещё раз!\n");
continue;
}
return x;
}
}

int main ( )
{
int x;
char a, b; //Изменяемые элементы
lisp s;
int act, flag;
printf("Выберите действие для выполнения:\n%s1 - автоматическая замена элементов в файле.
%s\n%s2 - ручная работа с файлом%s\n%s3 - выход из программы.%s\n", COL, NONE, COL1,
NONE, COL2, NONE);
```

```
cin » act;
switch (act)
{
case 1:
cin » a;
cin » b;
cout « "Замена " « a « " на " « b «".Результат: " « endl;
while (s!=NULL)
{
read_lisp (s);
write_lisp(s);
cout « endl;
Swap(s, a, b, s);
cout « endl;
destroy (s); //Удаления списка
s=NULL;
}
break;
case 2:
while (true)
{
x=act_to_perform();
switch (x)
{
case 1:
cout « "Введите список: ";
read_lisp(s);
break;
case 2:
destroy(s);
s = NULL;
cout « "Список удален успешно!" « endl;
break;
case 3:
cout « "Выберите элемент для замены: ";
```

```
cin » a;
cout « "Выберите подставляемый элемент: ";
cin » b;
Swap(s, a, b, s);
break;
case 4:
if (isNull(s))
cout « "Список пуст!" « endl;
else
{
cout « "Список: ";
write_lisp (s);
cout « endl;
}
break;
case 5:
return 0;
}
cout « "Хотите продолжить работу с программой? 1-да; 0-нет.";
cin » flag;
cout « endl;
if (flag == 0) return 0;
}
break;
}
}
```