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. 7381 _____ Тарасенко Е. А.
Преподаватель _____ Фирсов М. А.

Санкт-Петербург
2018

Задание.

Вариант 8:

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

Пояснение задания.

Задача состоит в том, чтобы пройти по всем уровням вложенности иерархического списка, попутно заменяя все вхождения заданного элемента (атома) x на заданный элемент (атом) y.

В качестве примера структуры списка на рисунке 1 представлен иерархический список, соответствующий записи **(a (b ( ) c) d)**.

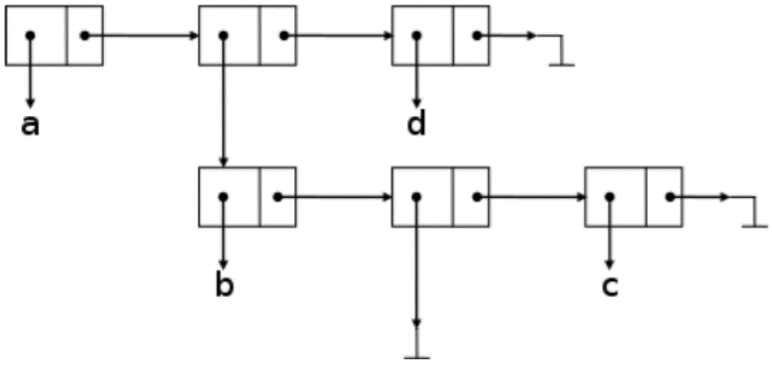

Рис. 1 – Пример представления иерархического списка (a (b ( ) c) d)

Описание алгоритма.

Программа посимвольно принимает скобочную запись иерархического списка, пропуская пробелы, реагируя на скобки увеличением или уменьшением уровня вложенности каждого текущего элемента и занося в список (при помощи функции **push**) каждый символ, принятый за так называемый атом. После этого на вход программе подаются **x** и **y** – символ, который следует заменить, и символ, на который следует заменить первый, соответственно. Совершается проход по списку и поиск необходимых вхождений

символа **x** с последующей заменой его на **y**. В конце происходит вывод итогового списка (с расстановкой скобок) и очистка выделенной памяти. В алгоритме присутствует отслеживание некорректных данных с выводом соответствующих ошибок. Для удобства восприятия предусмотрен «цветной» вывод.

Функции и структуры данных.

int main() – головная функция программы, в которой происходит инициализация переменных, выделение и освобождение памяти, вызовы функций, необходимых для построения и работы с иерархическим списком.

void push(Atom* list, int level, char value, int i) – функция добавления элемента в список. Она отвечает за заполнение элемента (атома) необходимыми данными и привязку атома к определенному «хвосту» иерархического списка.

void create_atom(Atom* atom, char value, int level) – функция, отвечающая за создание и начальную инициализацию атома, без учета его привязки к остальному списку.

void change_atom(Atom* list, char x, char y) – производит рекурсивный проход по списку с целью обнаружения всех вхождений элемента **x** и последующей замены их на элементы **y**.

void print_list(Atom* list, int is_son) – печать итогового списка, с соответствующей расстановкой скобок. Также использует рекурсию.

typedef struct Atom – структура, элемент списка. Содержит поля со значением элемента, уровнем вложенности и ссылками на элемент более глубокого уровня (если такой есть) и своего (также если таковой присутствует). В противном случае ссылки содержат **NULL**.

Тестирование.

Таблица 1 - Примеры условий и ответов

Исходное выражение:	**Результат:**
( )	**The list is empty!**
(a (b c) d) **a x**	**Result: (x(bc)d)**
(a (b c) h)))	**Counts of characters '(' and ')' is not equal!**
a b c	**The symbol '(' was expected!**
(a (b c) d (a) e) **a y**	**Result: (y(bc)d(y)e)**
	The list is empty!
(a b c(d e f(g)) h(i) j(k l) m n o(p)) **i 1**	**(abc(def(g))h(1)j(kl)mno(p))**

Вывод.

В ходе данной лабораторной работы были приобретены необходимые навыки по построению иерархических списков и работе с ними. На языке Си была составлена программа по замене всех вхождений заданного элемента списка на другой.

Приложение 1. Код программы.

```
#include <stdio.h>
#include <stdlib.h>
#define YES 1
#define NO 0

typedef struct Atom {
	char value;
	int level;
	struct Atom* son;
	struct Atom* brother;
} Atom;

void create_atom(Atom* atom, char value, int level){
	atom->value = value;
	atom->level = level;
	atom->son = NULL;
	atom->brother = NULL;
}

void print_list(Atom* list, int is_son){
	if(is_son == YES) printf("\x1b[32m(\x1b[0m");
	if(list->value)
		printf("\x1b[32m%c\x1b[0m", list->value);
	// recursive printing;
	if(list->son)
		print_list(list->son, YES);
	if(list->brother)
		print_list(list->brother, NO);

	if(is_son == YES) printf("\x1b[32m)\x1b[0m");
}

void change_atom(Atom* list, char x, char y){
	// change the atom;
	if(list->value == x) list->value = y;
	// recursion;
	if(list->son)
		change_atom(list->son, x, y);
	if(list->brother)
		change_atom(list->brother, x, y);
}

void push(Atom* list, int level, char value, int i){
	Atom* tmp = list;
	create_atom(&list[i - 1], value, level);
	Atom* element = &list[i - 1];

	while(((tmp->son) && (tmp->level != level)) || (tmp->brother)){
		if(tmp->brother){
```

```
                tmp = tmp->brother;
                continue;
            }
            if((tmp->son) && (tmp->level != level)){
                tmp = tmp->son;
                continue;
            }
        }
        if(i != 1){
            if((tmp->level) != (element->level)){
                tmp->son = element;
                printf("\x1b[33m'%c' is a son of '%c';\x1b[0m\n", value,
tmp->value);
            }
            else{
                tmp->brother = element;
                printf("\x1b[33m'%c' is a brother of '%c';\x1b[0m\n",
value, tmp->value);
            }
        }
        else
            printf("\x1b[33m'%c' is a head of the list;\x1b[0m\n", value);
}

int main(){
        //initialization;
        printf("Please, enter list members. Ex.: (a(bc)d).\n");
        char c, x, y;
        int level = 0, i = 1;
        Atom* list = (Atom*)malloc(i * sizeof(Atom));

        // create a head of the list;
        create_atom(&list[0], 0, 1);

        // entering the data;
        while((c = getchar()) != '\n'){
            if(c == ' ')
                continue;
            if(c == '('){
                        level++;
                        continue;
            }
            if(c == ')'){
                        level--;
                        continue;
            }
            list = (Atom*)realloc(list, i * sizeof(Atom));
            if(level > 0){
                push(list, level, c, i);
                i++;
            }
```

```
            else{
                  fprintf(stderr, "\x1b[31mThe symbol '(' was
expected!\x1b[0m\n");
                  free(list);
                  return 0;
            }
      }

      // list errors;
      if(level != 0){
            fprintf(stderr, "\x1b[31mCounts of characters '(' and ')' is
not equal!\x1b[0m\n");
            free(list);
            return 0;
      }
      if(list[0].value == 0){
            fprintf(stderr, "\x1b[31mThe list is empty!\x1b[0m\n");
            free(list);
            return 0;
      }

      // a problem solution;
      printf("Enter an element, which we should change: ");
      scanf("%c", &x);
      getchar();
      printf("Enter a new element: ");
      scanf("%c", &y);
      if(list->value)
            change_atom(list, x, y);
      printf("\x1b[32mResult: \x1b[0m");
      print_list(list, YES);
      printf("\n\x1b[32mFinish!\x1b[0m\n");

      //free memory;
      free(list);
      return 0;
}
```