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: РЕКУРСИВНАЯ ОБРАБОТКА ИЕРАРХИЧЕСКИХ СПИСКОВ

Студент гр. 6381 _____ Павлов А.П.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2018

Цель работы.

Познакомиться с основными функциями создания и обработки иерархического списка.

Постановка задачи.

Задание №7.

Удалить из иерархического списка все вхождения заданного элемента (атома) *x*;

Основные теоретические положения.

Представление иерархического списка

Традиционно иерархические списки представляют или графически, или в виде скобочной записи. На рисунке 1 приведен пример графического изображения иерархического списка. Соответствующая этому изображению сокращенная скобочная запись — это (a (b c) d e).

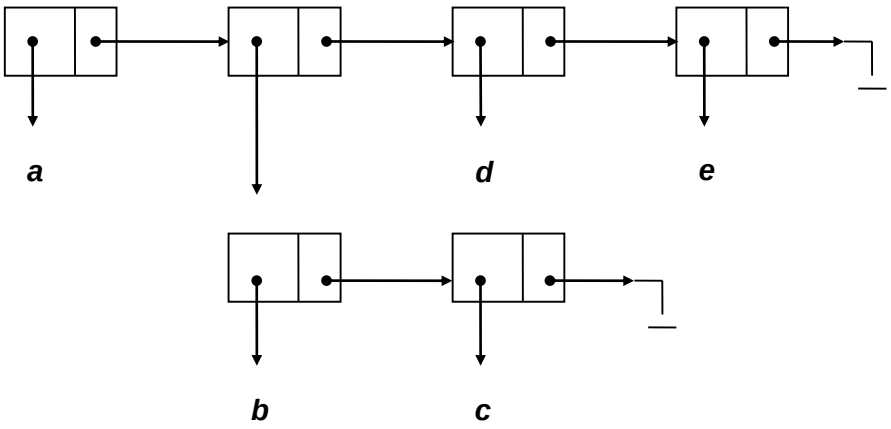

Рис.1 Пример представления иерархического списка в виде двумерного рисунка

Переход от полной скобочной записи, соответствующей определению иерархического списка, к сокращенной производится путем отбрасывания конструкции .Nill и удаления необходимое число раз пары скобок вместе с предшествующей открывающей скобке точкой.

Полная запись	Сокращенная запись
a	*a*
Nil	( )
(*a* . (*b* . (*c* . *Nil*)))	(*a b c*)
(*a* . ((*b* . (*c* . *Nil*)) . (*d* . (*e* . *Nil*))))	(*a* (*b c*) *d e*)

Рис.2 Примеры перехода от полной к сокращенной скобочной записи иерархических списков

Согласно приведенному определению иерархического списка, структура непустого иерархического списка – это элемент размеченного объединения множества атомов и множества пар «голова-хвост».

Рекурсивная структура иерархического списка на языке C++

```
typedef char base; // базовый тип элементов (атомов)
    struct  s_expr;
    struct  two_ptr  {
            s_expr  *hd;
            s_expr  *tl;
    } ;     //end two_ptr;
    struct  s_expr {
            bool tag; // true: atom, false: pair
            union  {
                    base  atom;
                    two_ptr pair;
            } node;                 //end union node
    };                      //end s_expr
    typedef  s_expr  *lisp;
```

Функциональная спецификация и реализация иерархического списка

Функциональная спецификация иерархического списка включает:

- функции — селекторы Head и Tail, выделяющие соответственно «голову» и «хвост» списка
- функции — конструкторы: Cons, создающая точечную пару (новый список из «головы» и «хвоста»), и Make_Atom, создающая атомарное S-выражение.
- предикаты Is Null, проверяющий список на отсутствие в нем элементов, и Atom, проверяющий, является ли список атомом.
- Необходимо включить в список базовых функций и функцию Destroy, позволяющую уничтожить созданный список, т.е. освободить память от ставших ненужными списочных структур.

Спецификация программы.

Программа удаляет все вхождения элемента *X*, введенного с клавиатуры, из иерархического списка. Она состоит из основных функций задания списка и работы с ним, а также из функции, удаляющей нужный элемент ***void deleteAtom(lisp s_prev, lisp s_cur, base element, int flag, int count)***.

Реализация.

Функция ***lisp head (const lisp s)*** получает на в ход указатель типа структуры *lisp*.

Если «голова» списка не атом, то функция *head()* возвращает список, на который указывает голова пары, т.е. подсписок, находящийся на следующем уровне иерархии. Если же «голова» списка — атом, то выводится сообщение об ошибке и функция прекращает работу.

Функция ***bool isAtom (const lisp s)*** получает на в ход указатель типа структуры *lisp*.

Предикат ***isAtom*** возвращает значение tag, которое равно True, если элемент — атом, и значение False, если — «голова-хвост». В случае пустого списка значение предиката False.

Функция ***Cons*** — конструктор, получает на в ход два указателя типа структуры *lisp*. При создании нового S-выражения требуется выделение памяти. Если памяти нет, то p == NULL и это приводит к выводу соответствующего сообщения об ошибке. Если «хвост» — не атом, то для его присоединения к «голове» требуется создать новый узел (элемент), головная ссылка которого будет ссылкой на «голову» этого «хвоста», а хвостовая часть элемента (tag.hd.tl) — ссылкой на его «хвост»

Функция ***void deleteAtom(lisp s_prev, lisp s_cur, base element, int flag, int count)*** получает на в ход 5 переменных:

lisp s_prev, lisp s_cur – указатели на фиктивную и настоящую головы списка.

base element – элемент типа base, который необходимо исключить из иерархического списка.

int flag – переменная необходимая для корректной работы программы (удаления головы списка на следующем уровне).

int count – переменная, необходимая для вывода рекурсии.

Сначала в функции *main()* создаем фиктивную голову и с помощью функци *Cons* объединяем его со считанным списком.

Для головы текущего элемента(s_cur) вызывается функция isAtom(s), если функция истина(голова – атом), то атом сравнивается с искомым элементом(getAtom(head(s_cur)) == element). Если данный атом – искомый элемент, то перекидываются указатели и атом удаляется. Иначе идем по списку дальше: s_prev приравнивается к s_cur, а s_cur – *tail(s_cur).*

В случае если голова указывает на новый список, то рекурсивно вызывается функция *deleteAtom(lisp s_prev, lisp s_cur, base element, int flag, int count)* для списка на следующем уровне.

Тестирование.

Тест№1	Enter list: (a(a)) Entered list: (a (a)) Enter atom that you want to delete a call deleteAtom call deleteAtom llac deleteAtom llac deleteAtom List without deleted atoms (()) destroy list: end!
Тест№2	Enter list: ((kfa(fkrmaa)fka)) Entered list: ((k f a (f k r m a a) f k a)) Enter atom that you want to delete a call deleteAtom call deleteAtom call deleteAtom llac deleteAtom llac deleteAtom llac deleteAtom List without deleted atoms ((k f (f k r m) f k)) destroy list: end!
Тест№3	Enter list: (aaa(kfk)fldaaa) Entered list: (a a a (k f k) f l d a a a) Enter atom that you want to delete a call deleteAtom call deleteAtom llac deleteAtom llac deleteAtom List without deleted atoms ((k f k) f l d) destroy list: end!
Тест№4	Enter list: (a(ab)ab) Entered list: (a (a b) a b) Enter atom that you want to delete a

	call deleteAtom call deleteAtom llac deleteAtom llac deleteAtom List without deleted atoms ((b) b) destroy list: end!
Тест№5	Enter list: ((((fkkakaa)krmaaaaa)ekvmaakm)ekmdmaa) Entered list: ((((f k k a k a a) k r m a a a a a) e k v m a a k m) e k m d m a a) Enter atom that you want to delete a call deleteAtom call deleteAtom call deleteAtom call deleteAtom llac deleteAtom llac deleteAtom llac deleteAtom llac deleteAtom List without deleted atoms ((((f k k k) k r m) e k v m k m) e k m d m) destroy list: end!

Исходный код:

Приложение А: код файлы functions.cpp

```
#include "functions.h"
#include <iostream>
#include <cstdlib>

namespace h_list
{

//.....................................
	lisp head (const lisp s){    // PreCondition: not null (s)
		if (s != NULL)
			if (!isAtom(s))
				return s->node.pair.hd;
			else{
				std::cerr << "Error: Head(atom) \n";
				exit(1);
			}
		else{
			std::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)
{// PreCondition: not null (s)
                if (s != NULL)
                        if (!isAtom(s))
                                return s->node.pair.tl;
                        else{
                                std::cerr << "Error: Tail(atom) \n";
                                exit(1);
                        }
                else{
                        std::cerr << "Error: Tail(nil) \n";
                        exit(1);
                }
        }
//.....................................
        lisp cons (const lisp h, const lisp t){
        // PreCondition: not isAtom (t)
        lisp p;
        if (isAtom(t)){
                std::cerr << "Error: Tail(nil) \n";
                exit(1);
        }
        else{
                p = new s_expr;
                if ( p == NULL){
```

```
                std::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;
        // s = NULL;
    }
```

```
}
//.....................
        base getAtom (const lisp s)
        {
                if (!isAtom(s)){
                        std::cerr << "Error: getAtom(s) for !isAtom(s) \n";
                        exit(1);
                }
                else
                        return (s->node.atom);
        }


//.....................
// enter list from console
        void read_lisp ( lisp& y){
                base x;
                do{
                        std::cin >> x;
                }while (x==' ');
                read_s_expr ( x, y);
        } //end read_lisp
//.....................
        void read_s_expr (base prev, lisp& y)
        { //prev - early read character
                if ( prev == ')' ) {
                        std::cerr << " ! List.Error 1 " << std::endl;
                        exit(1);
                }
                else if ( prev != '(' ) {
```

```
                y = make_atom (prev);
        }
        else{
                read_seq (y);
        }
    } //end read_s_expr
//.....
    void read_seq ( lisp& y)
    {   base x;
        lisp p1, p2;

        if (!(std::cin >> x)){
                std::cerr << " ! List.Error 2 " << std::endl;
                exit(1);
        }
        else {
                while( x==' ' )
                        std::cin >> x;
                if ( x == ')' ){
                        y = NULL;
                }
                else {
                        read_s_expr ( x, p1);
                        read_seq ( p2);
                        y = cons (p1, p2);
                }
        }
    } //end read_seq
//.....
// The procedure for displaying a list with brackets framing it - write_lisp,
```

```
//  and without framing brackets - write_seq
        void write_lisp (const lisp x){
//      cout << "in write_lisp" << endl;
        //empty list is displayed as ()
        if (isNull(x))
                std::cout << " ()";
        else if (isAtom(x))
                std::cout << ' ' << x->node.atom;
        else  { //non-empty list
                        std::cout << " (" ;
                        write_seq(x);
                        std::cout << " )";
                }
        } // end write_lisp
//.....................
void write_seq (const lisp x)
{//displays a sequence of elements of the list without brackets framing it
        if (!isNull(x)) {
                write_lisp(head (x));
                write_seq(tail (x));
        }
}
} // end of namespace h_list
```

Приложение Б: код файла functions.h

```
namespace h_list
{
        typedef char base; // base type of elements

        struct s_expr;
        struct two_ptr
```

```
{
        s_expr *hd;
        s_expr *tl;
} ;     //end two_ptr;

struct s_expr {
        bool tag; // true: atom, false: pair
        union
        {
                base atom;
                two_ptr pair;
        } node;                 //end union node
};                      //end s_expr

typedef s_expr *lisp;

void print_s_expr( lisp s );
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);

}
```

Приложение В: код файла main.cpp

```
#include <iostream>
#include <cstdlib>
#include "functions.h"


using namespace h_list;
void printTabs(int count);
void deleteHead(lisp s_cur, lisp tmpAtom);
void deleteTail(lisp s_cur, lisp tmpAtom);
void throwing_pointers(lisp s_cur, lisp s_prev);
void deleteAtom(lisp s_prev, lisp s_cur, base element, int flag, int count);// delete function

int main(){

        lisp s_prev, s_cur, start;// s_prev - pointer on Atom ; start - pointer on fictitious head for delete the true head of list; s_cur - poinet on true list
        base element;
        std::cout << std::boolalpha;
        std::cout << "Enter list:" << std::endl;
        read_lisp (s_cur);

        std::cout << "Entered list: " << std::endl;
```

```
        write_lisp (s_cur);
        std::cout << std::endl;

        s_prev = make_atom('S');//atom creation
        start = cons(s_prev, s_cur);//merging the list with a fictitious head

        std::cout << "Enter atom that you want to delete" << std::endl;
        std::cin >> element;

        deleteAtom(start, s_cur, element, 0, 0);

        std::cout << "List without deleted atoms" << std::endl;
        write_lisp(tail(start));
        std::cout << std::endl;

        std::cout << "destroy list: " << std::endl;
        destroy (start);

        std::cout << "end! " << std::endl;
        return 0;
}                                       //duplicate
void deleteTail(lisp s_cur, lisp tmpAtom){          //
        tmpAtom = head(s_cur);                      //
        s_cur -> node.pair.hd = NULL;                 //lines
        delete tmpAtom;                             //
}                                       //are
                                        //
void deleteHead(lisp s_cur, lisp tmpAtom){          //separated
        tmpAtom = head(s_cur);                      //
        s_cur -> node.pair.hd = NULL;                 //into
```

```
        delete tmpAtom;                         //
}                                       //separatefunctions

void printTabs(int count){                      //show recursion
        for(int i = 0; i < count; i++)
                std::cout << '\t';
}

void deleteAtom(lisp s_prev, lisp s_cur, base element, int flag, int count){//s_prev
- pointer on previous element of list
        printTabs(count);                                       //s_cur - pointer on
current element of list
        std::cout << "call deleteAtom" << std::endl;
        count++;
        if(s_cur == NULL){
                return;
                printTabs(count-1);
                std::cout << "llac deleteAtom" <<std::endl;
        }
        lisp tmpAtom = NULL; //variable to store an atom
        while(s_cur!= NULL)
        {
                if(isAtom(head(s_cur)))//if head is atom
                {
                        if(getAtom(head(s_cur)) == element)  //check atom
                        {
                                if(isAtom(head(s_prev)) || (flag==1))
                                {
                                        if(tail(s_cur) == NULL)//to delete the tail of list
                                        {
```

```
                    deleteTail(s_cur, tmpAtom);
                    s_prev -> node.pair.tl = NULL;
                    delete s_cur;
                    s_cur = NULL;
                }
                else            //to delete the any correct element
expect tail
                {
                    deleteTail(s_cur, tmpAtom);
                    s_prev -> node.pair.tl = s_cur -> node.pair.tl;
                    tmpAtom = s_cur -> node.pair.tl;
                    delete s_cur;
                    s_cur = tmpAtom;

                }
            }
            else
            {
                flag = 1;
                if(tail(s_cur) == NULL)//to delete the tail of list on
next level
                {
                    deleteHead(s_cur, tmpAtom);
                    s_prev -> node.pair.hd = NULL;
                    delete s_cur;
                    s_cur = NULL;
                }
                else
                {
                    deleteHead(s_cur, tmpAtom);
```

```
                                tmpAtom = tail(s_cur);
                                s_prev -> node.pair.hd = tmpAtom;
                                delete s_cur;
                                s_cur = tmpAtom;
                        }
                }
                else{                   //throwing pointers to move through the
list
                        s_prev = s_cur;
                        s_cur = s_cur -> node.pair.tl;
                }
        }
        else if(!isAtom(head(s_cur))){     //call deleteAtom for next level
                flag = 0;
                deleteAtom(s_cur, s_cur->node.pair.hd, element, flag, count);
                flag = 1;
                s_prev = s_cur;
                s_cur = s_cur->node.pair.tl;
        }
    }
    printTabs(count-1);
    std::cout << "llac deleteAtom" <<std::endl;
}
```

Приложение Г: код файла compile.sh

```
#!/bin/bash
g++ ./Source/functions.cpp ./Source/main.cpp -o Lab
echo 'Test 1:'
cat ./Tests/Test1.txt
echo '\n'
./Lab < ./Tests/Test1.txt
echo '\n'
```

```
echo 'Test 2:'
cat ./Tests/Test2.txt
echo '\n'
./Lab < ./Tests/Test2.txt
echo '\n'
echo 'Test 3:'
cat ./Tests/Test3.txt
echo '\n'
./Lab < ./Tests/Test3.txt
echo '\n'
echo 'Test 4:'
cat ./Tests/Test4.txt
echo '\n'
./Lab < ./Tests/Test4.txt
echo '\n'
echo 'Test 5:'
cat ./Tests/Test5.txt
echo '\n'
./Lab2 < ./Tests/Test5.txt
echo '\n'
```