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

ОТЧЕТ
по практической работе №2
по дисциплине «Алгоритмы и структуры данных»
Тема: Рекурсивная обработка иерархических списков

Студентка гр. 7382 _____ Еременко А.А

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2018

Цель работы.

Познакомиться с одной из часто используемых на практике нелинейных конструкций, способами её организации и рекурсивной обработки. Получить навыки решения задач обработки иерархических списков, как с использованием базовых функций их рекурсивной обработки, так и без использования рекурсии.

Задание.

Сформировать линейный список атомов исходного иерархического списка таким образом, что скобочная запись полученного линейного списка будет совпадать с записью исходного иерархического списка после устранения всех внутренних скобок.

Основные теоретические положения.

Определим соответствующий тип данных S_expr (El) рекурсивно, используя определение линейного списка (типа L_list):

< S_expr (El) > ::= < Atomic (El) > | < L_list (S_expr (El)) >,

< Atomic (E) > ::= < El >.

< L_list(El) > ::= < Null_list > | < Non_null_list(El) >

< Null_lis t> ::= Nil

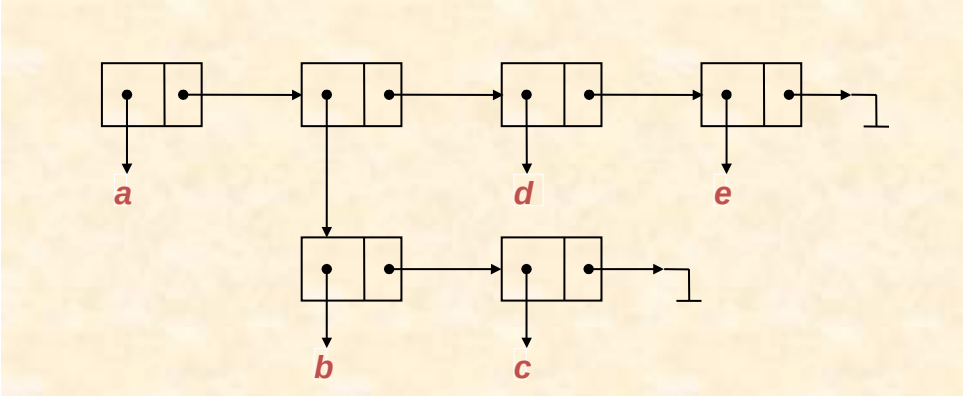
< Non_null_list(El) > ::= < Pair(El) >

< Pair(El) > ::= (< Head_l(El) > . < Tail_l(El) >)

< Head_l(El) > ::= < El >

< Tail_l(El) > ::= < L_list(El) >

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



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.

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

Выполнение работы.

Программа предназначена для формирования линейного списка атомов исходного иерархического списка путем устранения всех внутренних скобок в его сокращенной скобочной записи.

Описание функций.

Разберем основные функции из кода, указанного в приложении А.

1. bool isAtom(const lisp s) – на вход подается иерархический список s. Функция, проверяет атомарен ли поступивших на вход список. Если s==NULL то возвращается значение false, иначе возвращаемым значением служит s->test.
2. bool isNull(const lisp s) – на вход подается иерархический список s. Функция, проверяет поступивших список на пустоту. Возвращаемым значением является s == NULL.

3. void destroy(lisp s) – на вход подается иерархический список s. Функция производит освобождение памяти. Возвращаемое значение отсутствует.
4. char getAtom(const lisp s) – на вход подается иерархический список s. Функция, позволяет получить значение атомарного выражения. Если происходит ошибка, то выводится сообщение " Error - it's not an atom \n" и происходит выход по exit(1). Иначе возвращаемым значением служит s->node.atom.
5. void read_lisp(lisp& y) – на вход подаются введенные с консоли данные. Функция считывает иерархический список. Для нее есть две вспомогательные функции void read_s_expr(char prev, lisp& y) и void read_seq(lisp& y). Первой на вход подается предыдущий элемент и данные, второй только данные. На выходе получается иерархический список.
6. void write_lisp(const lisp x) – функция вывода списка с внешними скобками.
7. void write_seq(const lisp x) – функция вывода последовательности элементов списка без скобок.

Тестирование.

Для начала разберем примеры правильно введенных данных.

Введенные данные	Результат
(a(b)(c))	Input list: a(b)(c) Linear list = (a(b)(c)) ((b)(c)) ((c)) (c) (b) (a b c) Freeing memory: End!
(a(n(p(q)(w))(i(e)(r)))(m(s(t)(y))(d(u)(l))))	Input list: a(n(p(q)(w))(i(e)(r)))(m(s(t)(y))(d(u)(l))) Linear list = (a(n(p(q)(w))(i(e)(r)))(m(s(t)(y))(d(u)(l)))) ((n(p(q)(w))(i(e)(r)))(m(s(t)(y))(d(u)(l)))) ((m(s(t)(y))(d(u)(l)))) (m(s(t)(y))(d(u)(l)))

	((s(t)(y))(d(u)(l))) ((d(u)(l))) (d(u)(l)) ((u)(l)) ((l)) (l) (u) (s(t)(y)) ((t)(y)) ((y)) (y) (t) (n(p(q)(w))(i(e)(r))) ((p(q)(w))(i(e)(r))) ((i(e)(r))) (i(e)(r)) ((e)(r)) ((r)) (r) (e) (p(q)(w)) ((q)(w)) ((w)) (w) (q) (a n p q w i e r m s t y d u l) Freeing memory: End!
c	Input list: Atom

Неверный ввод данных.

Данные	Результат
(asa)(i)	Input list: a s a Linear list = (a s a) (s a)

	(a) (a s a) Freeing memory: End!

В данном примере вторая внешняя скобка не рассматривается, так как она не объединена с первой.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <cstdlib>
#include <windows.h>
#include <conio.h>
using namespace std;
```

```
namespace h_list {                                  //создание структуры иерархического
списка
        struct s_expr;
        struct  two_ptr {
                s_expr *head;                               //указатель на начало списка
                s_expr *tail;                               //указатель на конец списка
        };

        struct s_expr {
                bool test;                                  // если true: atom, иначе false: pair
                struct {
                        char 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 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);
        void read_s_expr(char prev, lisp& y);
        void read_seq(lisp& y);
        void write_lisp(const lisp x);
        void write_seq(const lisp x);
        lisp copy_lisp(const lisp x);
}

using namespace h_list;
lisp concat(const lisp y, const lisp z);
lisp l_list(const lisp s);

int main() {
        lisp s1, s2;
        cout << "Enter the list:" << endl;
        read_lisp(s1);
        cout << "Input list: " << endl;
        write_seq(s1);
        cout << endl;

        cout << "Linear list = " << endl;
        s2 = l_list(s1);
        write_lisp(s2);
        cout << endl;
        cout << "Freeing memory: " << endl;
        destroy(s2);
        cout << "End!" << endl;
        _getch();
        return 0;
}


lisp concat(const lisp y, const lisp z) {
        if (isNull(y)) return copy_lisp(z);
        else return cons(copy_lisp(head(y)), concat(tail(y), z));
```

```
}

lisp l_list(const lisp s) {
	if (isNull(s)) return NULL;
	else if (isAtom(s)) {

		cout << s->node.atom << endl;
		return cons(make_atom(getAtom(s)), NULL);
	}
	else //s - not empty list
		if (isAtom(head(s))) {
			cout << "(";
			write_seq(s);
			cout << ")" << endl;
			cout << endl;
			return cons(make_atom(getAtom(head(s))), l_list(tail(s)));
		}
		else {
			cout << "(";
			write_seq(s);
			cout << ")" << endl;
			cout << endl;
			return concat(l_list(head(s)), l_list(tail(s)));
		}
}

namespace h_list {
	lisp head(const lisp s) {
		if (s != NULL) if (!isAtom(s)) return s->node.pair.head;
		else { cout << "Atom\n"; _getch(); exit(1); }
		else { cout << "The list is empty\n"; _getch(); exit(1); }
	}
	bool isAtom(const lisp s) {                                    //функция проверяет
атомарен ли спискок
		if (s == NULL) return false;
		else return (s->test);
	}
	bool isNull(const lisp s) {                                  //функция проверяет
список на пустоту
		return s == NULL;
	}

	lisp tail(const lisp s) {
		if (s != NULL) if (!isAtom(s)) return s->node.pair.tail;
		else { cout << "The list is empty\n"; _getch(); exit(1); }
	}

	lisp cons(const lisp h, const lisp t) {
		lisp p;
		if (isAtom(t)) { cout << "Atom\n"; _getch(); exit(1); }
		else {
			p = new s_expr;
			if (p == NULL) {
				cerr << "Not enough memory\n";
				_getch();
				exit(1);
			}
			else {
				p->test = 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->test = true;
            s->node.atom = x;
            return s;
        }

        void destroy(lisp s) {                          //функция освобождения памяти
            if (s != NULL) {
                if (!isAtom(s)) {
                    destroy(head(s));
                    destroy(tail(s));
                }
                delete s;
            };
        }

        char getAtom(const lisp s) {                    //функция дает
возможность получить значение атомарного выражения
            if (!isAtom(s)) { cout << " Error - it's not an atom \n"; _getch();  exit(1); }
            else return (s->node.atom);
        }

        void read_lisp(lisp& y) {                       //функция считывания
списка
            char x;
            do cin >> x; while (x == ' ');
            read_s_expr(x, y);
        }

        void read_s_expr(char prev, lisp& y) {
        //вспомогательная процедура для read_lisp
            if (prev == ')') { cout << " Incorrect using a closing bracket\n" << endl;
_getch(); exit(1); }
            else if (prev != '(') y = make_atom(prev);
            else read_seq(y);
        }

        void read_seq(lisp& y) {                        //вспомогательная
процедура для read_lisp
            char x;
            lisp p1, p2;
            if (!(cin >> x)) { cout << "Error\n" << endl; _getch(); exit(1); }
            else {
                while (x == ' ') cin >> x;
                if (x == ')') y = NULL;
                else {
                    read_s_expr(x, p1);
                    read_seq(p2);
                    y = cons(p1, p2);
                }
            }
        }

        void write_lisp(const lisp x) {                 //функция вывода
списка с внешними скобками
            if (isNull(x)) cout << "()";
            else if (isAtom(x)) cout << ' ' << x->node.atom;
```

```
                else {
                        cout << "(";
                        write_seq(x);
                        cout << ")";
                }
        }

        void write_seq(const lisp x) {                                  //функция вывода
последовательности элементов списка без скобок
                if (!isNull(x)) {
                        write_lisp(head(x));
                        write_seq(tail(x));
                }
        }

        lisp copy_lisp(const lisp x) {
                if (isNull(x)) return NULL;
                else if (isAtom(x)) return make_atom(x->node.atom);
                else return cons(copy_lisp(head(x)), copy_lisp(tail(x)));
        }
        }
```