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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Иерархические списки

Студент гр. 7381 _____ Дорох С.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2018

Задание

14 вариант:

Обратить иерархический список на всех уровнях вложения; например, для исходного списка (***a*** (***b c***) ***d***) результатом обращения будет список (***d*** (***c b***) ***a***).

В работе используется язык программирования C/C++.

Пояснение задания

Задача состоит в том, чтобы пройтись по всем уровням вложенности иерархического списка, и из них записать все элементы в новый иерархический список в обратном порядке.

В качестве примера, наглядно демонстрирующего иерархические списки, на рисунке 1 представлен список, соответствующий сокращенной записи ***(a (b ( ) c) d)***.

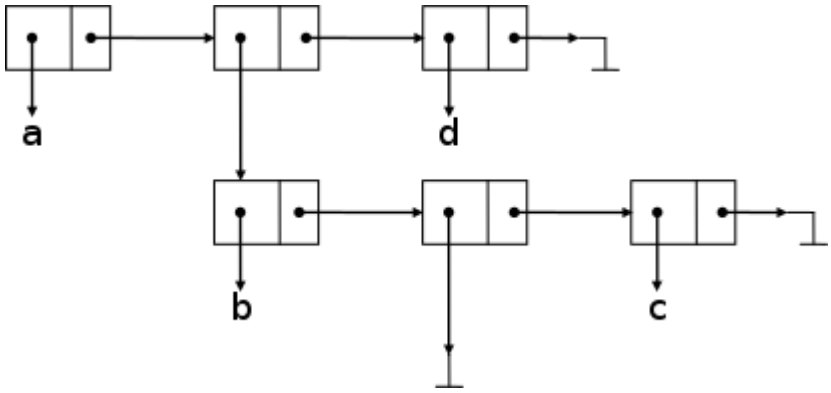

Рисунок 1 – Пример представления иерархического списка ***(a (b ( ) c) d)***.

Описание алгоритма

Создана рекурсивная функция обращения иерархического списка на всех уровнях вложения, принимающая в качестве аргументов начальный список и пустой список, в который будут записываться атомы начального списка в обратном порядке. В данной функции проверяется является ли элемент начального списка атомом, если это так то рекурсивно вызывается это же функция и в пустой список записывается атом начального списка. Если элемент списка не является атомом, то рекурсивно вызывается эта функция и происходит переход на следующий уровень начального списка с созданием подсписка для хранения обращённых элементов данного уровня. После

прохождения подсписка на определённом уровне функция склеивает обращённый подсписок со своим родителем, так выполняется до тех пор, пока весь начальный список не будет обработан.

Когда начальный список будет обработан данная рекурсивная функция вернёт обращённый список и произойдёт выход из функции.

Описание функций

Базовые функции:

`lisp head(const lisp s)` - функция перехода в голову узла списка.

`const lisp s` - список, в котором требуется перейти в голову первого узла.

Возвращаемое значение: если `s` пуст, то выводится сообщение об ошибке и программа завершает работу, если же список не пуст и первый узел указывает на подсписок, то возвращается этот подсписок, в противном случае узел указывает на атом, при этом выводится сообщение об ошибке и программа завершает работу.

lisp tail(const lisp s) - функция перехода в хвост списка.

const lisp s - список, в котором требуется перейти в хвост.

Возвращаемое значение: если `s` пуст, то выводится сообщение об ошибке и программа завершает работу. Если же список не пуст и он не атом , то возвращается этот же список, начиная со второго узла, в противном случае список состоит из одного атома, при этом выводится сообщение об ошибке и программа завершает работу.

lisp cons(const lisp h, const lisp t) - функция создания нового списка из h и t.

const lisp h - список, который будет вставлен в голову нового списка.

const lisp t — список, который будет хвостом нового списка.

Возвращаемое значение: если t атом, то выводится сообщение об ошибке и программа завершает работу. Если t- список, то создается новый список, в хвост которого помещается t, а в голову h.

lisp make_atom(const char x) - функция создания списка, состоящего из одного атома.

const char x- символ, который будет значением атома в списке.

Возвращаемое значение: список, с одним атомом.

bool isAtom(const lisp s) - функция, проверяющая является ли s атомом.

const lisp s - проверяемый список.

Возвращаемое значение: true - атом, false - в противном случае.

bool isNull(const lisp s)- функция, проверяющая являетсяли s пустым списком.

const lisp s - проверяемый список.

Возвращаемое значение: true - список пуст, false - в противном случае.

char getAtom(const lisp s) - функция, возвращающая значения атома s.

const lisp s - список, являющийся атомом.

Возвращаемое значение: зачение поля atom, если список - атом. Если же список не является атомом, то выводится сообщение об ошибке и программа завершает работу.

void destroy(lisp s) - функция удаления списка.

lisp s - список, который требуется удалить.

Возвращаемое значение: функция ничего не возвращает.

Функции ввода:

void read_lisp(lisp &y, std::istream &list) - основная функция создания списка.

Считывает символ с клавиатуры и направляет его в следующую функцию read_s_expr.

lisp &y - ссылка на создаваемый список. Возвращаемое значение: функция ничего не возвращает.

std::istream &list – ссылка на потоковый ввод list

void read_s_expr(char prev, lisp &y, std::istream &list) - функция, определяющая дальнейшее создание списка по аргументу prev. Если prev - символ `)`, то выводится сообщение об ошибке и программа завершает работу, так как список не может начинаться с этого символа. Если prev является символом `(`, то вызывается следующая функция read_seq для создания списка или подсписков.

char prev - символ, считанный в функции read_lisp и который определяет, как дальше будет создаваться список.

lisp &y - ссылка на создаваемый список. Возвращаемое значение: функция ничего не возвращает.

std::istream &list – ссылка на потоковый ввод list

void read_seq(lisp &y std::istream &list) - функция, считывающая и объединяющая основной список и подсписки, а также проверяющая наличие

символов в потоке.

lisp &y - ссылка на создаваемый список.

std::istream &list – ссылка на потоковый ввод list.

Функции для выполнения лабораторной работы:

void print_basic (int deepCount, lisp s) - Функция печати состояния иерархического списка на определённом вызове рекурсии.

lisp s - список, состояние которого необходимо вывести.

int deepCount - счетчик промежуточных значений глубины списка для вывода промежуточных данных.

void print_reverse (int deepCount, lisp s) - Функция печати состояния обращённого иерархического списка на определённом вызове рекурсии.

lisp s - список, состояние которого необходимо вывести.

int deepCount - счетчик промежуточных значений глубины списка для вывода промежуточных данных.

lisp reverse (lisp s, lisp z, int count) – рекурсивная функция выполняющая обращение иерархического списка.

lisp s – список, который нужно обратить.

lisp z -- список, в который будут записываться обращённые атомы.

int count – счётчик промежуточных значений глубины списка.

Тестирование

Для проверки работоспособности программы был создан скрипт для автоматического ввода и вывода тестовых данных:

test1.txt: (a(bc(de)fg));

test2.txt: (qwer(ty));
test3.txt: (esrever);

test4.txt: (abc(de(f)))))

test5.txt:)kjndkjfn);

test6.txt: (khbkhb(shd(bd));

Результаты тестирования сохраняются в файл result.txt.

Ниже представлена таблица тестирования программы с подробным описанием работы алгоритма для первого теста.

Входные данные	Выходные данные
(a(bc(de)fg))	Test 1: (a(bc(de)fg)) Entered list: (a (b c (d e) f g)) call (a (b c (d e) f g)) call ((b c (d e) f g)) call (b c (d e) f g) call (c (d e) f g) call ((d e) f g) call (d e) call (e) call () llac (e d) llac (d) llac () call (f g) call (g) call () llac (g f (e d) c b) llac (f (e d) c b) llac ((e d) c b) llac (c b) llac (b) llac () call () llac ((g f (e d) c b) a) llac (a) llac () Reversed list: ((g f (e d) c b) a)
(qwer(ty))	Test 2: (qwer(ty)) Entered list: (q w e r (t y)) call (q w e r (t y)) call (w e r (t y)) call (e r (t y)) call (r (t y)) call ((t y)) call (t y) call (y) call () llac (y t)

	llac (t) llac () call () llac ((y t) r e w q) llac (r e w q) llac (e w q) llac (w q) llac (q) llac () Reversed list: ((y t) r e w q)
(esrever)	Test 3: (esrever) Entered list: (e s r e v e r) call (e s r e v e r) call (s r e v e r) call (r e v e r) call (e v e r) call (v e r) call (e r) call (r) call () llac (r e v e r s e) llac (e v e r s e) llac (v e r s e) llac (e r s e) llac (r s e) llac (s e) llac (e) llac () Reversed list: (r e v e r s e)
(abc(de(f)))	Test 4: (abc(de(f))) Extra characters!

)kjndkjfn)	Test 5:)kjndkjfn) Closed bracket before opened bracket!
(khbkhb(shd(bd))	Test 6: (khbkhb(shd(bd)) Not enough symbols!

Вывод:

В процессе выполнения лабораторной работы были поли получены знания и навыки по иерархическим спискам, рекурсивным функциям, bash-скриптам и автоматизации тестирования. Работа была написана на C/C++.

ПРИЛОЖЕНИЕ А

КОД MAIN.CPP

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <sstream>
#include "lisp_func.h"

void print_basic (int deepCount, lisp s) {   // Функция печати состояния
иерарх. списка на определённом вызове рекурсии
    for (int i = 0; i < deepCount; i++)
        std::cout << "\t";
    std::cout << "call ";
    write_lisp(s);
    std::cout << std::endl;
}

void print_reverse (int deepCount, lisp s) {    // Функция печати состояния
обратного иерарх. списка на определённом вызове рекурсии
    for (int i = 0; i < deepCount-1; i++)
        std::cout << "\t";
    std::cout << "llac ";
    write_lisp(s);
    std::cout << std::endl;
}

lisp reverse( lisp s, lisp z, int count) {  //Рекурсивная функция для
создания обратного иерарх. списка
    print_basic(count, s);
    lisp k;
    if (isNull(s)){
        print_reverse(++count, z);
        --count;
        return z;
    }
    if(isAtom(head(s))){
        k = reverse(tail(s), cons(head(s), z), ++count);    //Если элемент
атом, то вызывается reverse и выполняется запись атома в новый список
        print_reverse(count, z);
        return k;
    }
    k = reverse(tail(s), cons(reverse(head(s), NULL, count), z), ++count);
//Если не атом, то создаётся подсписок z для для записи в него
    print_reverse(count, z);
//подсписка исходного иерархического списка
    return k;
```

```
}

int main () {
    std::stringbuf buffer;
    std::string list;
    char ch;

    getline(std::cin, list);
    std::istream str(&buffer);
    buffer.str(list);

    if(!list.length()){
        std::cout << "List is empty!" << std::endl;
        return 0;
    }
    lisp s1;
    read_lisp (s1, str);

    if(str >> ch){                                          //Проверка на то,
осталось ли что в str после выполнения функции read_lisp
        std::cout<<"Extra characters!"<< std::endl;
        destroy(s1);
        return 0;
    }

    std::cout << "Entered list: ";
    write_lisp (s1);
    std::cout << std::endl;
    lisp array[20];
    lisp k = reverse(s1, NULL, 0);
    std::cout << "Reversed list:";
    write_lisp (k);
    std::cout << std::endl;
    destroy(k);
    delete(s1);
    return 0;
}
```

ПРИЛОЖЕНИЕ Б
ФАЙЛ LISP_FUNC.CPP

```
#include "lisp_func.h"
#include <iostream>
#include <cstdlib>
#include <fstream>

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) {
    lisp p;
    if (isAtom(t)) {
        std::cerr << "Error: Cons(*, atom)\n";
        exit(1);
    }
    else {
        p = new s_expr;
        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)
        return;
```

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

void read_lisp ( lisp& y, std::istream &list){
        char x;
        do {
            list >> x;
        } while (x == ' ');
        read_s_expr ( x, y, list);
}

void read_s_expr (char prev, lisp& y, std::istream &list){
        if ( prev == ')' ) {
            std::cout << "Closed bracket before opened bracket! " <<
std::endl;
            exit(1);
        }
        else if ( prev != '(' )
            y = make_atom (prev);
        else
            read_seq (y, list);
}



void read_seq ( lisp& y, std::istream &list){
        char x;
        lisp p1, p2;
        if (!(list >> x)) {
            std::cout << "Not enough symbols!" << std::endl;
            exit(1);
        }
        else {
            while ( x == ' ' ){
                list >> x;
            }

            if ( x == ')' )
                y = NULL;
```

```
            else {
                read_s_expr ( x, p1, list);
                read_seq ( p2, list);
                y = cons (p1, p2);
            }
        }
}
//.........................
// Процедура вывода списка с обрамляющими его скобками — write_lisp,
// а без обрамляющих скобок — write_seq
void write_lisp (const lisp x){
    //пустой список выводится как ()
    if (isNull(x))
        std::cout << " ()";
    else if (isAtom(x))
        std::cout << ' ' << x->node.atom;
    else { //непустой список}
        std::cout << " (" ;
        write_seq(x);
        std::cout << " )";
    }
} // end write_lisp
//.........................
void write_seq (const lisp x) {
//выводит последовательность элементов списка без обрамляющих его скобок
    if (!isNull(x)) {
        write_lisp(head (x));
        write_seq(tail (x));
    }
}
```

ПРИЛОЖЕНИЕ В
ФАЙЛ HIERAR_LIST.HPP

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <sstream>

typedef char base; // базовый тип элементов (атомов)
struct s_expr;
struct two_ptr{
    s_expr *hd;
    s_expr *tl;
};

struct s_expr {
    bool tag; // true: atom, false: pair
    union
```

```
    {
        base atom;
        two_ptr pair;
    } node;
};
typedef s_expr *lisp;
// функции
void print_s_expr( lisp s );
// базовые функции:
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);
// функции ввода:
void read_lisp ( lisp& y, std::istream &astr);

void read_s_expr (char prev, lisp& y, std::istream &astr);

void read_seq ( lisp& y, std::istream &astr);
// функции вывода:
void write_lisp (const lisp x); // основная
void write_seq (const lisp x);
lisp copy_lisp (const lisp x);
```