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(ка) гр. 9382 Иерусалимов Н.

_____

Преподаватель Фирсов М.А.

_____

Санкт-Петербург

2020

Цель работы.

Научиться рекурсивно работать с иерархическими списками.

Задание.

9) подсчитать число атомов в иерархическом списке; сформировать линейный список атомов, соответствующий порядку подсчёта;

Основные теоретические положения.

В практических приложениях возникает необходимость работы с более сложными, чем линейные списки, нелинейными конструкциями. Рассмотрим иерархичесикй список элементов базового типа El или S-выражение.

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) >

Структура иерархического списка:

```
typedef char base; // базовый тип элементов (атомов)
        struct s_expr;
        struct two_ptr {
        s_expr *hd;          //указатеь на голову
        s_expr *tl;//указатель на хвост
} ; //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;
```

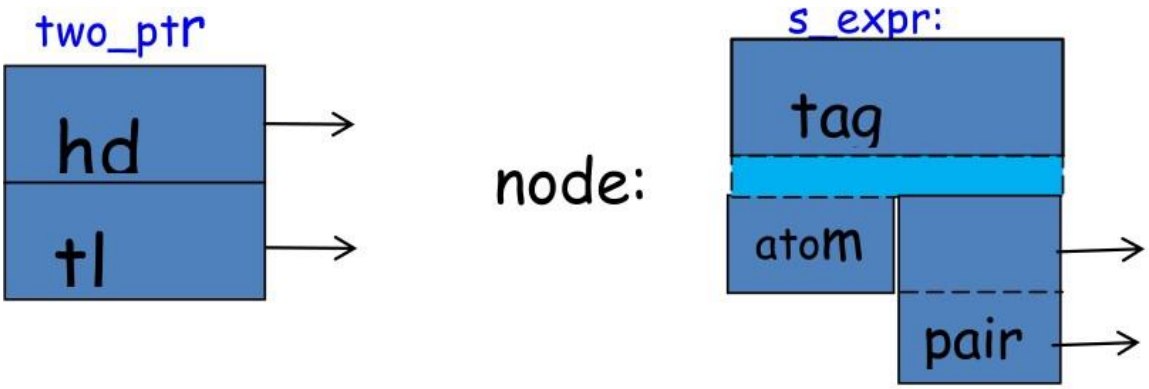

Рис. 2.3. Представление рекурсивной структуры списка

Представление списка (a(b c) d e) в графическом виде:

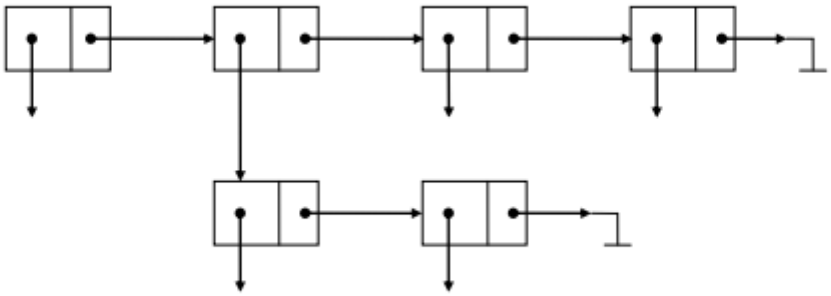

Рис. 2.1. Пример представления иерархического списка в виде двумерного рисунка

Реализованные функции.

Файл l_intrfc.h:

Создаем пространство имен h_list и в нем реализуем структуру иерархического списка, там же храним количество атомов int count; и глубину рекурсии int recDepth. Следующие функции объявлены в этом пространстве имен:

lisp head (const lisp s) — принимает на вход иерархический список s, возвращает иерархический список, создаёт голову списка

lisp tail (const lisp s) - принимает на вход иерархический список s, возвращает иерархический список, создаёт хвост списка

lisp cons (const lisp h, const lisp t) - принимает на вход два константных иерархических списка, возвращает иерархический список, создаёт новый спиок из головы и хвоста

lisp make_atom (const base x) — принимает константу x базового типа, возвращает иерархический список, создаёт атом списка

bool isAtom (const lisp s) - принимает на вход иерархический список s, возвращает логическое значение (true или false), проверяет, атом ли список

bool isNull (const lisp s) - принимает на вход иерархический список s, возвращает логическое значение (true или false), проверяет нулевой ли список

void destroy (lisp s) - принимает на вход иерархический список s, удаляет список

base getAtom (const lisp s) - принимает на вход иерархический список s, возвращает базовый символ, возвращает значение фтома списка

void read_lisp (lisp& y, std::ifstream& temp) — принимает на вход ссылку на иерархичесикй список и ссылку на входной поток, читает список

void read_s_expr (base prev, lisp& y, std::ifstream& temp) — принимает переменную базового типа prev, ссылку на иерархический список y, ссылку на поток ввода, читает s-выражение

void read_seq (lisp& y, std::ifstream& temp) — принимает ссылку на иерархический список y, ссылку на поток ввода, читает последовательность символов

void write_lisp (const lisp x) - принимает на вход константу с иерархическим списком s, выводит список

void write_seq (const lisp x) - принимает на вход константу с иерархическим списком s, выводит последоваетльность символов

int getCount () - дает доступ к счетчику количества атомов.

void print(const char *message) – Принимает сообщение которое надо вывести рекурсии, используется для вывода данных.

void print(const char *message, char z) – принимает сообщение и атом который прочитал парсер, используется для вывода данных.

Файл main.cpp:

Реализуемые функции :

lisp flatten(const lisp s) – Принимает константу со списком s. Выравнивание иерархического списка, т.е. формирование из него линейного списка путем удаления из сокращенной скобочной записи иерархического списка всех внутренних скобок (функция Flatten) . Например, ((a b) c (d (e f (g)) h)) → (a b c d e f g h).

lisp concat (const lisp y, const lisp z); -Принимает два константных списка. Сцепление двух иерархических списков, соединение их в один список (функция Concat). Например, y = (a (b c) d), z = (e f (g) h), Concat (y,z) = (a (b c) d e f (g) h)

void counter(lisp s, lisp *linery) – Принимает список и указатель на список для записи в него линейного списка. Функция запускает парсер для введенной строки, запускает функцию для преобразования списка в линейный и выводит количество атомов.

void printDepth(const char *message) – принимает константный указатель на сообщение которое надо вывести. Выводит глубину рекурсии и переданное сообщение.

Описание алгоритма

Для ввода иерархического списка, запускается процедура read_lisp. Эта процедура использует внутри себя обращение к процедуре read_s_expr, а

она, в свою очередь, обращение к read_seq.
read_lisp объявляется переменная base x в которую будет вписываться символ путем ввода с консоли и с помощью do while() будут вписываться любые символы кроме пробела после чего вызовется функция read_s_expr.
В функции read_s_expr передается символ и список куда записывать атомы. В зависимости от того какой символ от этого и действует, если это закрывающая скобка то выдаст ошибку, если символ не равен открывающей скобке тогда это атом и он записывается в переданный список “y” через функцию make_atom() и прибавляем 1 к count. Если же это открывающая скобка то вызывается функция read_seq.
В функции read_seq объявляется 2 списка p1 и p2, голова и хвост соответственно, если встречается закрывающая скобка то переданный список равен NULL, если же встретили что-то другое то вызывается read_s_expr с аргументами (x, p1) то есть, задаем голову списка пропуская опять через весь алгоритм. После этого вызывается read_seq(p2) где задается хвост. Вызывается функция cons() с аргументами p1 и p2 для создания списка, напоминаю cons() это наш “конструктор”.

Для формирования линейного списка используется lisp flatten(const lisp s) проверяется пустой ли переданный node, если node пуст то мы возвращаем NULL, если node == atom тогда мы вызываем функцию cons(make_atom(getAtom(head(s))),NULL) то есть, создаем новый список с исходным атомом. Если же первый node не атом мы проверям следующий элемент и если он атом тогда вызываем cons(make_atom(getAtom(head(s))), flatten(tail(s))); в первый аргумент конструктора мы передаем новый атом головы, во второй запускаем снова flatten и определяем что у нас там стоит(atom, скобка или что-то другое). Результат выполнение возвращаем. Если и следующий элемент не атом тогда мы заходим в else и вызываем concat() со следующими аргументами 1 – flatten(head(s)) 2 – Flatten(tail(s)) и возвращаем. То есть, когда мы заходим в последний else это значит что была

найдена конструкция на подобие ((b)), тем самым с помощью функции concat и этих аргументов мы идем “глубже” в рекурсию, и выполняем то что было описано выше.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№	Входные данные	Выходные данные
1.	(a)	.We went to "read_s_expr". .Met an opening parenthesis! ..We went to "read_seq"! ...We went to "read_s_expr". ...Counter added met atom 'a' ....We went to "read_seq"! ...Met a closing parenthesis! ..Exited "read_s_expr"! .Exited "read_seq"! Exited "read_seq"! _____ *Forming a linear list* .We went into the "flatten" function! ..entered the "cons" function! ...We went into the "flatten" function! ..Exited "cons" function .Exited "flatten" function _____ There are 1 atoms in the list. (a)
2.	(ab)	.We went to "read_s_expr". .Met an opening parenthesis! ..We went to "read_seq"! ...We went to "read_s_expr". ...Counter added met atom 'a' ....We went to "read_seq"!

		.....We went to "read_s_expr". .....Counter added met atom 'b' .....We went to "read_seq"! .....Met a closing parenthesis! ....Exited "read_s_expr"! ...Exited "read_seq"! ..Exited "read_s_expr"! .Exited "read_seq"! Exited "read_seq"! _____ *Forming a linear list* .We went into the "flatten" function! ..entered the "cons" function! ...We went into the "flatten" function! ....entered the "cons" function! .....We went into the "flatten" function! ....Exited "cons" function ...Exited "flatten" function ..Exited "cons" function .Exited "flatten" function _____ There are 2 atoms in the list. (a b)
3.	((a) f)	There are 2 atoms in the list. (a f)
4.	(a (b) c)	There are 3 atoms in the list. (a b c)
5.	(a (b c))	There are 3 atoms in the list. (a b c)
6.	(a (b (c)))	There are 3 atoms in the list. (a b c)
7.	(FFFF(FFF)FF)	There are 9 atoms in the list.

		(F F F F F F F F F)
8.	(a b (c))	There are 3 atoms in the list. (a b c)
9.	(((DF)DF(DF)CFV))	There are 9 atoms in the list. (D F D F D F C F V)
10.	(Здра(вствуй),Не(бо)в(облаках))	There are 23 atoms in the list. (З д р а в с т в у й , Н е б о в о б л а к а х)

Выводы.

Научились работать с нелинейными списками, реализовали функцию по подсчету атомов в нем. Увидели как переводить нелинейный список в линейный.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: l_intrfc.h

```
namespace h_list
```

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


// базовые функции:
    lisp head (const lisp s);//Возвращет голову узла
    lisp tail (const lisp s);//Возвращет хвост узла
    lisp cons (const lisp h, const lisp t); // Конструктор для связывания списка
    lisp make_atom (const base x);//создает новый элемент
    bool isAtom (const lisp s);//проверяет узел на то атом он или нет
    bool isNull (const lisp s);// проверяет нулевой ли node
    void destroy (lisp s);//очищает память
    base getAtom (const lisp s);//Возвращает атом переданного узла
// функции ввода:
    void read_lisp ( lisp& y); // основная функция для считывания
```

```
    void read_s_expr (base prev, lisp& y);
    void read_seq ( lisp& y);
// функции вывода:
    void write_lisp (const lisp x); // основная функция для вывода на экран
    void write_seq (const lisp x);
    void print(const char *message);//выводит глубину рекурсии
    void print(const char *message, char z);//выводит глубину рекурсии с каким-
то параметром
    lisp copy_lisp (const lisp x);// копирует список и возвращает его
    int getCount ();//дает доступ к количеству атомов.


}
```

Название файла: l_intrfc.cpp

```
#include <iostream>
#include <cstdlib>
#include "l_intrfc.h"
using namespace std;


namespace h_list
{
    int recDepth = 0;
    int count =0;
    int getCount (){return count;}
    void print(const char *message){
        for (int i = 0; i < recDepth; ++i){
            cout<< '.';
        }
```

```
        cout<< message << '\n';
    }
    void print(const char *message, char z){
        for (int i = 0; i < recDepth; ++i){
            cout<< '.';
        }

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

        lisp p;
        if (isAtom(t)) { cerr << "Error: Cons(*, atom)\n"; exit(1);}
        else {
            p = new s_expr;
            if ( p == NULL) {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)) { cerr << "Error: getAtom(s) for !isAtom(s) \n"; exit(1);}
        else return (s->node.atom);
    }
//.....................
// ввод списка с консоли
    void read_lisp ( lisp& y){
        base x;
        do cin >> x; while (x==' ');//Пропускаем все пробелы при вводе
        read_s_expr ( x, y); //отправляем символ (который не пробел) для
обработки и список
    } //end read_lisp
//.....................
    void read_s_expr (base prev, lisp& y)
```

```
    { //prev — ранее прочитанный символ}
        ++recDepth;
        print("We went to \"read_s_expr\".");
        if ( prev == '(' ){print("Met an opening parenthesis!");}

        if ( prev == ')' ) { // Если закрывающая скобка то выводим ошибку
            cerr << " ! List.Error 1 " << endl; exit(1);
        }
        else if ( prev != '(' ){ // Если не закрывающая скобка, то это атом,
добавляем его в список
            ++count;
            print("Counter added met atom ",prev);
            y = make_atom (prev);
        }
        else {
            read_seq(y);
            --recDepth;
            print("Exited \"read_seq\"!");
        }
    } //end read_s_expr
//.....
    void read_seq ( lisp& y)
    {
        ++recDepth;
        base x;
        print("We went to \"read_seq\"!");

        lisp p1, p2; // голова - хвост
        if (!(cin >> x)) {cerr << " ! List.Error 2 " << endl; exit(1);}
        else {
```

```
        while ( x==' ' ) cin >> x;
        if ( x == ')' ) {
           --recDepth;
           print("Met a closing parenthesis!");
           y = NULL;
        }
        else { //Либо '(' либо атом
           read_s_expr ( x, p1); // задаем голову снова пропуская алгоритм и
добавляя новые узлы.
           read_seq ( p2);// задаем хвост

           --recDepth;
           print("Exited \"read_s_expr\"!");
           --recDepth;
           print("Exited \"read_seq\"!");
           y = cons (p1, p2);//объединяем в один список указываем на голову и
хвост
        }
     }
  } //end read_seq


//.....................
// 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)) cout << " ()";
     else if (isAtom(x)) cout << ' ' << x->node.atom;
     else { //непустой список}
```

```
            cout << " (" ;
            write_seq(x);
            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));

        }

    }
//.....................
    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)));
    } //end copy-lisp
} // end of namespace h_list
```

Название файла: main.cpp

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

lisp concat (const lisp y, const lisp z); //соединяет два списка
lisp flatten(const lisp s);//преобразует нелинейный список в линейный
void printDepth(const char *message);//Выводит глубину рекурсии

int depth = 0;//Глубина рекурсии
int main ( )
{ SetConsoleCP(1251); // для вывода кирилицы
   SetConsoleOutputCP(1251); // для вывода кирилицы
   lisp s1, s2;
   cout<<"Enter the list:"<<endl;
   read_lisp (s1);//Считываем список
   cout<<"\n_____\n*Forming a
linear list*\n\n";
   s2 = flatten(s1);//Формируем его в линейный список
   cout<<"_____\n";
   cout<<"There are "<< getCount() <<" atoms in the list." << endl;//выводим
количество атомов
   write_lisp(s2);//Выводим линейный список

   getch();//что бы консоль не закрывалась
   return 0;
}
```

```
void printDepth(const char *message){
    for (int i = 0; i < depth; ++i){
        cout<< '.';
    }
    cout<< message << '\n';
}

//.....................
lisp concat (const lisp y, const lisp z)
{
    ++depth;
    printDepth("We went into the \"concat\" function!");
    if (isNull(y)) return copy_lisp(z);
    else {
        ++depth;
        printDepth("Entered the \"cons\" function!");
        printDepth("Allocated memory for a new node");
        lisp temp = cons (copy_lisp(head (y)), concat (tail (y), z));

        --depth;
        printDepth("Exited \"cons\" function");
        --depth;
        printDepth("Exited \"concat\" function");
        return temp;
    }
} // end concat
//.....................
lisp flatten(const lisp s)
{
    ++depth;
```

```
    printDepth("We went into the \"flatten\" function!");
    if (isNull(s)) return NULL;
    else if(isAtom(s)) return cons(make_atom(getAtom(s)),NULL);
    else if (isAtom(head(s))) {//s ? непустой список
        ++depth;
        printDepth("entered the \"cons\" function!");
        lisp temp =cons( make_atom(getAtom(head(s))),flatten(tail(s)));

        --depth;
        printDepth("Exited \"cons\" function");
        --depth;
        printDepth("Exited \"flatten\" function");

        return temp;
    }
    else {//Not Atom(Head(s))
        lisp temp =concat(flatten(head(s)), flatten(tail(s)));
        --depth;
        printDepth("Exited \"flatten\" function");
        --depth;
        printDepth("Exited \"flatten\" function");
        --depth;
        printDepth("Exited \"concat\" function");
        return temp;
    }

} // end flatten
```