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 _____ Голубева В.П.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Научиться рекурсивно работать с иерархическими списками.

Задание.

15) Проверить структурную идентичность двух иерархических списков (списки структурно идентичны, если их устройство (скобочная структура и количество элементов в соответствующих (под)списках) одинаково, при этом атомы могут отличаться);

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) >

Струтура иерархического списка:

```
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;
```

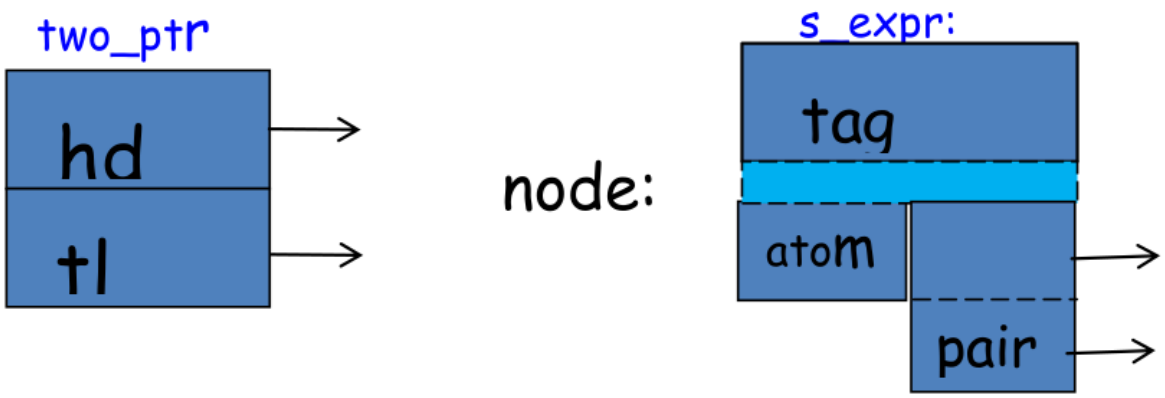
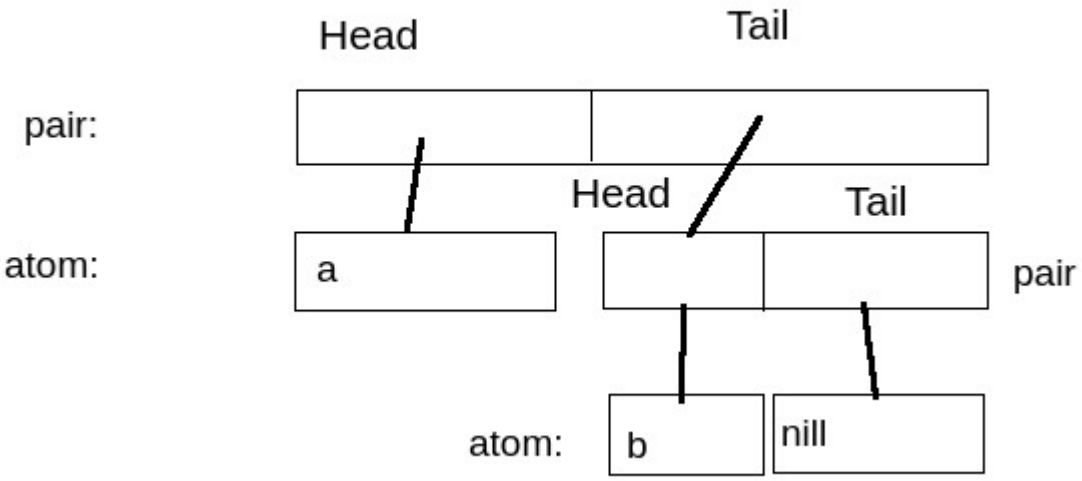


Рис. 2.3. Представление рекурсивной структуры списка

(a (b))



Реализованные функции.

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

void destroy_2(lisp x, lisp y) - принимает на вход два иерархических списка x и y, удаляет два списка

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

void list_match1 (const lisp x, const lisp y) - принимает на вход два константных иерархических списка, сравнивает два списка, рекурсивно вызывает void list_match2 (const lisp , const lisp)

void list_match2 (const lisp x, const lisp y) - принимает на вход два константных иерархических списка, сравнивает два списка, рекурсивно вызывает void list_match1 (const lisp x, const lisp y)

Описание рекурсивной функции.

Функция void list_match1(const lisp , consp list) передаем в неё два списка для сравнения. Проверяем на то, пустые ли они. Если да, то выводим сообщение о совпадении и очищаем память. Потом проверяем, не атом ли

списки. Вызываем функцию void list_match2 (const lisp , const lisp). Если один из списков нулевой, а другой нет, то выводим сообщение о несовпадении и очищаем память. Если какой-то из списков является атомом, то выводим сообщение о несовпадении и очищаем память. Если они оба нулевые, то выводим сообщение о совпадении и очищаем память. Если они не нулевые, то вызываем void list_match1(const lisp , const lisp) для head(x) и head(y), а затем void list_match2 (const lisp , const lisp) для tail(x) и tail(y), тем самым обходим списки.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	(g h p (x(c))	! List.Error 2	Проверка на некорректных данных
2.	(g (h) p))s(xc)	! List.Error 1	Проверка на некорректных данных
3.	() ()	Ввёдён list1: () Введён list2: () Структурная идентичность списков: true	
4.	(s(d(c))d) (x(s(z))c)	Ввёдён list1: (s (d (c)) d) Введён list2: (x (s (z)) c)	

		Структурная идентичность списков: true	
5.	(s((p)d((b)c))d) (x((l)s((q)z))c)	Ввёдён list1: (s ((p) d ((b) c)) d) Введён list2: (x ((l) s ((q) z)) c) Структурная идентичность списков: true	
6.	(g h p) (x(c))	Ввёдён list1: (g h p) Введён list2: (x (c)) Структурная идентичность списков: false	
7.	(a g) ()	Ввёдён list1: (a g) Введён list2: () Структурная идентичность списков: false	
8.	(a g) (d(c))	Ввёдён list1: (a g) Введён list2: (d (c))	

		Структурная идентичность списков: false	
9.	(a) (x c d h j k l)	Ввёдён list1: (a) Введён list2: (x c d h j k l) Структурная идентичность списков: false	
10.	(h k s(x d(a))a s f) (z x q(f u(d))l m c)	Ввёдён list1: (h k s (x d (a)) a s f) Введён list2: (z x q (f u (d)) l m c) Структурная идентичность списков: true	

Выводы.

Я научилась работать с иерахическими списками. Создала программу, которая проверяет структурную идентичность двух иерархических списков.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: realisation.cpp

```
// continue of namespace h_list
#include "declaration.h"


using namespace std;
namespace h_list
{

//.....................................
      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";
                        exit(1);
                  }
            else {
                  cerr << "Error: Head(nil) \n";
                  exit(1);
            }
      }
//.....................................
      bool isAtom (const lisp s){
            if(s == NULL) //если список пустой то false
                  return false;
            else
                  return (s -> tag);
      }
//.....................................
      bool isNull (const lisp s){
            return s==NULL;//возвращает сравнение с null
      }
//.....................................
      lisp tail (const lisp s)
{// PreCondition: not null (s)
            if (s != NULL)
                  if (!isAtom(s))//проверка на то, что это не атом
                        return  s->node.pair.tl;//возвращает  хвост
переданного списка
                  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: Tail(nil) \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;//выделяем память
            s -> tag = true;
            s->node.atom = x;//присваиваем переданное значение
            return s;
        }

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

        void destroy_2(lisp x, lisp y){
            destroy (x);
            destroy (y);
            //совместили  удаление  двух  списков,  чтоб  избежать
излишнего дублирования в коде
        }


    //.........................
        base getAtom (const lisp s)
```

```
        {
            if (!isAtom(s)) { //проверка на то, атом ли это
                cerr << "Error: getAtom(s) for !isAtom(s) \n";
                exit(1);
            }
            else
                return (s->node.atom);//выводит значения атома
        }


    //.........................
    //
        void read_lisp ( lisp& y, ifstream& temp)
        {   base x;
            do
                temp >> x; //читаем последовательность из файла
            while (x==' ');
                read_s_expr ( x, y, temp);
        } //end read_lisp
    //.........................
        void read_s_expr (base prev, lisp& y, ifstream& temp)
        {
            if ( prev == ')' ) {
                cerr << " ! List.Error 1 " << endl;
                exit(1);
            }
            else if ( prev != '(' )
                y = make_atom (prev);//создаём атом
            else
                read_seq (y, temp);
        } //end read_s_expr
    //.........................
        void read_seq ( lisp& y, ifstream& temp)
        {   base x;
            lisp p1, p2;


            if  (!(temp  >>  x))  {//читаем  последовательность  из
файла
                cerr << " ! List.Error 2 " << endl;
                exit(1);
            }
            else {
                while  ( x==' ' )
                    temp >> x;
                if ( x == ')' )
                    y = NULL;
                else{
                    read_s_expr ( x, p1, temp);
                    read_seq ( p2, temp);
                    y = cons (p1, p2);
                }
            }
        } //end read_seq
    //.........................
```

```
          void write_lisp (const lisp x)
          {
          if (isNull(x)) cout << " ()";
          else if (isAtom(x))
               cout << ' ' << x->node.atom;//выводим значение узла
               else  {
                    cout << " (" ;
                    write_seq(x);//извлекаем данные из списка дальше
                    cout << " )";
               }
          } // end write_lisp
     //.........................
          void write_seq (const lisp x)
          {
               if (!isNull(x)) {
                    write_lisp(head (x));
                    write_seq(tail (x));
               }
          }
     //.........................
          void list_match1 (const lisp x, const lisp y){
               /*if (isNull(x)&&isNull(y)){
                    cout << " ()\n";
                    cout<<"Структурная идентичность списков: true\n";
                    destroy_2(x, y);//очищаем память
                    exit(1);
               }
               else */if (isAtom(x)&&isAtom(y)) {


               }

                    else  if (!(isAtom(x)||isAtom(y))){

                         list_match2(x,  y);//вызываем  функцию  для
дальнейшей обработки списка


                    }
                    else {
                         cout<<"Структурная  идентичность  списков:
false\n";
                         destroy_2(x,y);
                         exit(1);

                    }

     }
     //.........................
          void list_match2 (const lisp x, const lisp y){

               if                        ((isNull(x)&&!isNull(y))||((!
isNull(x))&&isNull(y))){//проверка  на  то,  закончилась  ли  проверка
одного списка раньше другого
                    cout<<"Структурная  идентичность  списков:  false\
n";
```

```
                    destroy_2(x,y);
                    exit(1);
                }

                if ((isAtom(x)||isAtom(y))){//проверка на то, что в
каком-то узле атом, а в каком-то список
                    cout<<"Структурная идентичность списков: false\
n";
                    destroy_2(x,y);
                    exit(1);
                }

                if  (isNull(x)&&isNull(y)){//если  на  данном  этапе
списки оба пустые, то они идентичны

                    cout<<"Структурная идентичность списков: true\n";

                    destroy_2(x,y);
                    exit(1);
                }

                if (!isNull(x)&&!isNull(y)) {//рекурсивный вызов, если
обработка не завершена
                    list_match1(head (x), head(y));
                    list_match2(tail (x), tail(y));


                }

          }


     } // end of namespace h_list
```

Название файла: declaration.h

```
     #pragma once
     #include <fstream>
     #include <iostream>
     #include <cstdlib>

     namespace h_list
     {
          typedef char base;

          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;


        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);
        void destroy_2(lisp, lisp);

        base getAtom (const lisp s);

        void read_lisp ( lisp& y, std::ifstream& temp);
        void read_s_expr (base prev, lisp& y, std::ifstream& temp);
        void read_seq ( lisp& y, std::ifstream& temp);


        void write_lisp (const lisp x);
        void write_seq (const lisp x);

        void list_match1 (const lisp x, const lisp y);
        void list_match2 (const lisp x, const lisp y);

} // end of namespace h_list
```

Название файла: main.cpp

```
#include "declaration.h"

using namespace std;
using namespace h_list;

        int main(){
        lisp s1;
        lisp s2;

        ifstream temp("t.txt");

        read_lisp (s1, temp);
        cout << "Ввёдён list1: " << endl;
        write_lisp (s1);
        cout << endl;

        read_lisp (s2, temp);
        cout << "Введён list2: " << endl;
        write_lisp (s2);
```

```
        cout << endl;

        temp.close();

        cout<<"\n";

        if (isAtom(s1)&&isAtom(s2)){
            cout<<"Структурная идентичность списков: true\n";
            destroy_2(s1, s2);
            return 0;
        }

        list_match1(s1, s2);

        destroy_2(s1, s2);

        return 0;
}
```