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 СПИСКОВ

Студент гр. 9382 _____ Кодуков А.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

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.

Основные теоритические положения:

Иерархический список:

< 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) >

Задание:

10) подсчитать число различных атомов в иерархическом списке;

сформировать из них линейный список;

Выполнение работы:

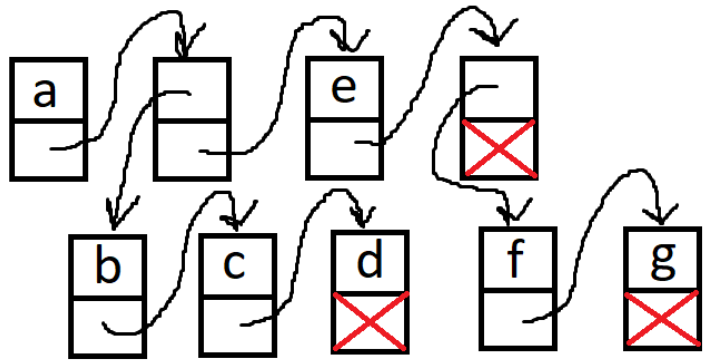
Иерархический список:

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

- tag – является ли элемент атомом
- pair (head, tail) – указатели на голову и хвост списка

Пример:

(a (b c d) e (f g))



Реализованные функции:

Выравнивание списка

Сигнатура:

```
hlist flatten2(hlist s, hlist t)
```

Аргументы:

- s – иерархический список
- t – накопитель

Возвращаемое значение:

hlist – результат выравнивания

Алгоритм:

- s пуст => вернуть t
- s явл. атомом => вернуть иерархический список с головой s и хвостом t
- s не явл. атомом => рекурсивный вызов выравнивания от головы s и результата выравнивания хвоста s и t

Подсчет числа различных атомов

Сигнатура:

```
std::vector<base> unique(hlist s)
```

Аргументы:

- s – иерархический список

Возвращаемое значение:

std::vector<base> - вектор уникальных элементов

Алгоритм:

- выравнивание списка
- запись списка в массив
- сортировка массива

- удаление подряд идущих повторяющихся элементов
- в массиве осталось по одной копии каждого атома

Тестирование:

№	Входные данные	Результат
1	(a(bcd)e(fg))	7 unique elements: a b c d e f g
2	(aabb)	2 unique elements: a b
3	(a(a(a(a))))	1 unique elements: a
4	(abcdefa)	6 unique elements: a b c d e f
5	(aa(bb)cc(dd(ee)))	5 unique elements: a b c d e
6	(a(roza(upala(na(lapu(azora))))))	8 unique elements: a l n o p r u z
7	a	1 unique elements: a
8	()	0 unique elements:
9	)	Wrong input

Вывод:

В результате выполнения работы были изучены принципы рекурсивной обработки списков а также структура иерархических списков. Были реализованы функции выравнивания иерархического списка и поиска различных атомов.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

hlist.h

```
#ifndef __HLIST_H_
#define __HLIST_H_

#include <vector>
#include <iostream>

// интерфейс АТД "Иерархический Список"
namespace h_list {
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* hlist;

// базовые функции:
hlist head(hlist s);
hlist tail(hlist s);
hlist cons(hlist h, hlist t);
hlist make_atom(const base &x);
bool isAtom(hlist s);
bool isNull(hlist s);
void destroy(hlist s);

hlist flatten2(hlist s, hlist t);
std::vector<base> unique(hlist s);

base getAtom(hlist s);

// функции ввода:
void read_hlist(hlist &y, bool m);  // основная
void read_s_expr(base prev, hlist &y, std::istream &str);
void read_seq(hlist &y, std::istream &str);

// функции вывода:
void write_hlist(hlist x);  // основная
void write_seq(hlist x);

hlist copy_hlist(hlist x);

}  // end of namespace h_list

#endif
```

hlist.cpp

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

#include <cstdlib>
#include <fstream>
#include <algorithm>

using namespace std;
namespace h_list {

//.....................................
hlist head(hlist 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(hlist s) {
  if (s == NULL)
    return false;
  else
    return (s->tag);
}
//.....................................
bool isNull(hlist s) { return s == NULL; }
//.....................................
hlist tail(hlist 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);
  }
}
//.....................................
hlist cons(hlist h, hlist t)
// PreCondition: not isAtom (t)
{
  hlist 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;
    }
  }
}
//.........................
hlist make_atom(const base &x) {
  hlist s;
  s = new s_expr;
  s->tag = true;
  s->node.atom = x;
  return s;
}

//.........................
void destroy(hlist s) {
  if (s != NULL) {
    if (!isAtom(s)) {
      destroy(head(s));
      destroy(tail(s));
    }
    delete s;
    // s = NULL;
  };
}
//.........................
base getAtom(hlist 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_hlist(hlist &y, bool m) {
  base x = ' ';
  if (m) {
    std::ifstream fs("input.txt");
    if (fs.is_open()) {
      do {
        fs >> x;
      } while (x == ' ');
      read_s_expr(x, y, fs);
      fs.close();
    } else
    std::cout << "\nImpossible to open file\n";
  } else {
    cout << "введите список:\n";
    do {
      cin >> x;
    } while (x == ' ');
    read_s_expr(x, y, cin);
  }
}  // end read_hlist
//.........................
void read_s_expr(base prev, hlist &y, std::istream &str) {  // prev - ранее
прочитанный символ}
  if (prev == ')') {
    cerr << " ! List.Error 1 " << endl;
```

```
    exit(1);
  } else if (prev != '(')
    y = make_atom(prev);
  else
    read_seq(y, str);
}  // end read_s_expr
//.........................
void read_seq(hlist &y, std::istream &str) {
  base x;
  hlist p1, p2;

  if (!(str >> x)) {
    cerr << " Wrong input " << endl;
    exit(1);
  } else {
    while (x == ' ') str >> x;
    if (x == ')')
      y = NULL;
    else {
      read_s_expr(x, p1, str);
      read_seq(p2, str);
      y = cons(p1, p2);
    }
  }
}  // end read_seq
//.........................
// Процедура вывода списка с обрамляющими его скобками - write_hlist,
//  а без обрамляющих скобок - write_seq
void write_hlist(hlist 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_hlist
//.........................
void write_seq(hlist x) {  //выводит последовательность элементов списка
                                  //без обрамляющих его скобок
  if (!isNull(x)) {
    write_hlist(head(x));
    write_seq(tail(x));
  }
}
//.........................
hlist copy_hlist(hlist x) {
  if (isNull(x))
    return NULL;
  else if (isAtom(x))
    return make_atom(x->node.atom);
  else
    return cons(copy_hlist(head(x)), copy_hlist(tail(x)));
}  // end copy-hlist


// Optimal list straightening function (t - result)
hlist  flatten2(hlist s, hlist t) {
  // Quit recursion
  if (isNull(s)) return t;
  // Atom -> head and tail are linear
```

```
  if (isAtom(s)) return cons(s, t);
  // Not atom -> split head and tail
  return flatten2(head(s), flatten2(tail(s), t));
}  // end flatten2

// Finiding unique atoms
std::vector<base> unique(hlist s) {
  hlist st = nullptr, lin;
  std::vector<base> v;

  if (!isNull(s)) {
    // straightening list and conversion to vector
    lin = flatten2(s, st);
    while (!isNull(lin)) {
      v.push_back(getAtom(head(lin)));
      lin = tail(lin);
    }
    // sort vector
    qsort(v.data(), v.size(), sizeof(base),
      [](const void *a, const void *b) {
        return (*(base *)a > *(base *)b) - (*(base *)a < *(base *)b);
      });
    std::cout << "Результат сортировки:\n ";
    for (auto &a: v)
      std::cout << a << " ";
    std::cout << "\n";
    // erasing repeating elements
    for (int i = 1; i < v.size(); i++)
      if (v[i] == v[i - 1]) {
         v.erase(v.cbegin() + i);
         i--;
        }
    }
  return v;
}

}  // end of namespace h_list
```

main.cpp

```
/* Kodukov A. 9382 v.10
 *
 * подсчитать число различных атомов в иерархическом списке;
 * сформировать из них линейный список;
 */

#include <windows.h>

#include <cstdlib>
#include <iostream>
#include <conio.h>

#include "hlist.h"

using namespace std;
using namespace h_list;

int main() {
  bool m;

  SetConsoleCP(1251);        // для вывода кирилицы
  SetConsoleOutputCP(1251);  // для вывода кирилицы
  while (1) {
```

```
    char mode;

    system("cls");
    std::cout << "Choose mode:\n1 - console input\n2 - file input\n";
    mode = _getch();
    if (mode == '1') {
      m = false;
      break;
    } else if (mode == '2') {
      m = true;
      break;
    } else {
      std::cout << "Wrong option";
      _getch();
    }
  }
  while (1) {
    system("cls");
    hlist s1, s2, s3 = nullptr;
    cout << boolalpha;
    read_hlist(s1, m);
    cout << "введен список: \n";
    write_hlist(s1);
    cout << endl;

    cout << "flatten2 списка =\n";
    s2 = flatten2(s1, s3);
    write_hlist(s2);
    cout << endl;

    std::vector<base> v = unique(s1);
    cout << v.size() << " unique elements:\n ";
    for (auto &a : v)
      cout << a << " ";
    cout << "\n";

    destroy(s2);
    destroy(s3);
    cout << "Press any key...";
    char key = _getch();
    if (key == 27)
      break;
  }

  return 0;
}
```