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: Иерархические списки

Студент гр. 7383 _____ Лосев М. Л.

Преподаватель _____ Размочаева Н. В.

Санкт-Петербург

2018

Содержание

1. ЦЕЛЬ РАБОТЫ

Цель работы: получение навыков работы с иерархическими списками на языке программирования C++.

Формулировка задачи: проверить идентичность двух иерархических списков.

2. РЕАЛИЗАЦИЯ ЗАДАЧИ

2.1 ИСПОЛЬЗУЕМЫЕ СТРУКТУРЫ ДАННЫХ

two_ptr – структура, которая содержит два поля: указатель на «голову» и указатель на «хвост».

s_expr – структура элемента иерархического списка. Содержит поля: *tag* – тэг: если структура является «атомом», значение тэга равно *true*, иначе – *false*, и *node* – объединение, которое содержит два поля: *atom* – поле типа *base* (хранит символ, если элемент списка является атомом), *и pair* – поле типа *two_ptr* (хранит указатель на «хвост» элемента и указатель на «голову» его вложенного списка, если элемент является атомом).

lisp – тип, определенный как указатель на объект типа *two_ptr.*

base – тип, определенный *как char.*

2.2 ИСПОЛЬЗУЕМЫЕ ФУНКЦИИ

lisp head (const lisp s) – функция, которая принимает указатель на указатель на элемент иерархического списка и возвращает указатель на «голову» вложенного списка этого элемента.

lisp tail (const lisp s) – функция, которая принимает указатель на указатель на элемент иерархического списка и возвращает указатель на его «хвост» (следующий за ним элемент).

lisp cons (const lisp h, const lisp t) – функция, которая принимает два указателя на указатель на элементы иерархического списка, создает третий объект этого типа, для которого первый элемент является «головой» его вложенного списка, а второй его «хвостом», и возвращает указатель на созданный объект.

lisp make_atom (const base x) – функция, которая принимает значение переменной типа *base,* создает атом (элемент иерархического списка, у которого нет «головы» и «хвоста», но определено поле *atom* – оно равно *x*), и возвращает указатель на созданный элемент.

bool isAtom (const lisp s) – функция, которая принимает указатель на элемент иерархического списка и возвращает *true,* если этот объект является «атомом» или *false* во всех остальных случаях.

bool isNull (const lisp s) – функция, которая принимает указатель на элемент иерархического списка, и возвращает true, если этот указатель «нулевой» или false во всех остальных случаях.

void destroy (lisp s) – функция, которая принимает указатель на элемент иерархического списка, и освобождает память, занятую всеми элементами иерархического списка, «головой» которого он является.

base getAtom (const lisp s) – функция, которая принимает указатель на элемент иерархического списка, и возвращает значение его поля *atom* (если он является «атомом»).

bool isEqual (const lisp f, const lisp s) – функция, которая принимает два указателя на элементы иерархического списка, и возвращает true, если два иерархических списка, «головами» которых являются объекты, указатели на которые были приняты, эквивалентны, или false во всех остальных случаях.

template <typename stream>

void read_lisp (stream s, lisp& y) – шаблонная функция, которая получает указатель на поток *s* (поток может быть разным: например, ifstream или istream) и указатель на элемент иерархического списка, считывает из потока *s* иерархический список и сохраняет его (то есть его голову) в объект, указатель на который который был получен.

template <typename stream>

void read_s_expr (stream s, base prev, lisp& y) – шаблонная функция, которая получает указатель на поток *s* (поток может быть разным: например, ifstream или istream), переменную типа *base* (предыдущий считанный символ) и элемент иерархического списка, считывает из потока *s* один элемент иерархического списка и сохраняет его (то есть его голову) в объект, указатель на который был получен.

template <typename stream>

void read_seq (stream s, lisp& y) – шаблонная функция, которая получает указатель на поток *s* (поток может быть разным: например, ifstream или istream), объект типа, указателем на который является lisp, то есть

s_expr, считывает из потока *s* последовательность элементов иерархического списка и сохраняет первый из них (то есть его голову) в объект, который был получен, второй – как хвост первого и так далее.

void write_lisp (const lisp x) – функция, которая получает указатель на элемент иерархического списка, выводит в поток *cout* иерархический список, головой которого является принятый объект.

void write_seq (const lisp x) – функция, которая получает указатель на указатель на элемент иерархического списка, выводит в поток *cout* последовательность элементов иерархического списка, головой которого является принятый объект.

3. ТЕСТИРОВАНИЕ

3.1 ПРОЦЕСС ТЕСТИРОВАНИЯ

Программа собрана в операционной системе Ubuntu 18.04 компилятором gcc. В других ОС тестирование не проводилось.

3.1 Результаты тестирования

Тестовые случаи представлены в приложении Б. По результатам тестирования было показано, что задача выполнена.

4 . ВЫВОД

В ходе работы была написана программа на языке C++, решающая поставленную задачу с помощью рекурсии. Был получен опыт использования рекурсии, работы с иерархическими списками.

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

файл l_intrfc.h:

```
// интерфейс АТД "Иерархический Список"
#include <iostream>
#include <cstdlib>

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 *lisp;

	// функции
	bool isEqual (const lisp f, const 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:
// ввод списка с консоли
template <typename stream>
void read_lisp ( stream *s, lisp& y)
{     base x;
      do (*s).get(x); while (x==' ');
      read_s_expr (s, x, y);
} //end read_lisp
//.........................
template <typename stream>
void read_s_expr (stream s, base prev, lisp& y)
{ //prev - ранее прочитанный символ}
      if ( prev == ')' ) { return; }
      else if ( prev != '(' ) y = make_atom (prev);
             else read_seq (s, y);
} //end read_s_expr
//.........................
template <typename stream>
void read_seq (stream *s, lisp& y)
{     base x;
      lisp p1, p2;

      if (!(*s).get(x)) { return;}
      else {
            while  ( x==' ' ) (*s).get(x);
            if ( x == ')' ) y = NULL;
            else {
                  read_s_expr (s, x, p1);
                  read_seq (s, p2);
                  y = cons (p1, p2);
            }
      }
} //end read_seq

// функции вывода:
void write_lisp (const lisp x);            // основная
void write_seq (const lisp x);
```

```
	lisp copy_lisp (const lisp x);

} // end of namespace h_list
```

файл l_intrfc.cpp:

```
// continue of namespace h_list
#include "l_intrfc.h"
#include <iostream>
#include <cstdlib>

using namespace std;
namespace h_list
{

	bool isEqual (const lisp f, const lisp s)
	{
		if (isNull(f) && isNull(s))
			return true;
		if (isNull(f) != isNull(s))
			return false;



		if (isAtom(f) && isAtom(s))
			return (getAtom(f) == getAtom(s));
		else
			if (!isAtom(f) && !isAtom(s))
				return  (  isEqual(head(s),  head(f))  &&  isEqual(tail(s),
tail(f)) );
			else return false; // one is atom and the other is not
	}

	//.................................
	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"; return NULL; }
      else { cerr << "Error: Tail(nil) \n";
            return NULL;
      }
}
//.....................................
lisp cons (const lisp h, const lisp t)
// PreCondition: not isAtom (t)
{lisp p;
if (isAtom(t)) { cerr << "Error: Tail(nil) \n"; return NULL;}
else {
      p = new s_expr;
      if ( p == NULL)    {cerr << "Memory not enough\n"; return NULL; }
      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"; return
0;}
        else return (s->node.atom);
    }



    //.........................

    //.........................
    // 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
```

файл l_mod1.cpp:

```
/*  использование модуля с АТД "Иерархический Список" .
Интерфейс модуля в заголовочном файле "l_intrfc.h"
и его реализация (в отдельном файле l_impl.cpp) образуют
пространство имен namespace h_list
*/

#include <iostream>
#include <fstream>
#include <cstdlib>
#include "l_intrfc.h"



using namespace std;
using namespace h_list;



void UserInterface ()
{
    int key;
    lisp s1, s2;
```

```
    while (true) {
        cout << "Выбери действие:" << endl;
        cout << "0. Завершить выполнение;" << endl;
        cout << "1. Загрузить списки из пользовательского файла;" << endl;
        cout << "2. Загрузить списки из файла по умолчанию;" << endl;
        cout << "3. Ввести списки с клавиатуры;" << endl;
        cin >> key;
        switch (key) {
            case 0:
                return;
                break;
            case 1:
                  {
                        string filename;
                        cout << "input file name: ";
                        cin.ignore(256, '\n');// игнорируем оставшиеся в cin после
ввода '1' символы
                        getline(cin, filename);

                        ifstream infile (filename);

                        read_lisp (&infile, s1);
                        cout << "введен list1: " << endl;
                        write_lisp (s1);
                        cout << endl;
                        infile.ignore(256, '\n'); // игнорируем оставшиеся в первой
строке файла символы
                        read_lisp (&infile, s2);
                        cout << "введен list2: " << endl;
                        write_lisp (s2);
                        cout << endl;

                        infile.close();

                        if (isEqual(s1, s2))
                              cout << "list1 = list2";
                        else
                              cout << "list1 != list2";
                        cout << endl;
            }
```

```
            break;
            case 2:
            {
            ifstream infile ("input");

                        read_lisp (&infile, s1);
                        cout << "введен list1: " << endl;
                        write_lisp (s1);
                        cout << endl;
                        infile.ignore(256, '\n'); // игнорируем оставшиеся в первой
строке файла символы
                        read_lisp (&infile, s2);
                        cout << "введен list2: " << endl;
                        write_lisp (s2);
                        cout << endl;

                        infile.close();

                        if (isEqual(s1, s2))
                              cout << "list1 = list2";
                        else
                              cout << "list1 != list2";
                        cout << endl;
            }
            break;
            case 3:
            {
                        cout << "введите list1:" << endl;
                        cin.ignore(256, '\n'); // игнорируем оставшиеся в cin после
ввода '3' символы
                        read_lisp (&cin, s1);
                        cout << "введен list1: " << endl;
                        write_lisp (s1);

                        cout << endl;
                        cout << "введите list2:" << endl;
                        cin.ignore(256, '\n'); // игнорируем оставшиеся в cin после
ввода '3' символы
                        read_lisp (&cin, s2);
                        cout << "введен list2: " << endl;
```

```
                        write_lisp (s2);
                        cout << endl;

                        if (isEqual(s1, s2))
                              cout << "list1 = list2";
                        else
                              cout << "list1 != list2";
                        cout << endl;
            }
            break;

            default : cout << "! - ...";
            break;
        };
    }
}



int main ( )
{
      UserInterface();

      cout << "end! " << endl;
      return 0;
}
```

файл MakeFile:

```
all: l_mod1.o l_impl.o
      g++ l_mod1.o l_impl.o -o l_mod1
l-mod1.o: l_mod1.cpp l_intrfc.h
      g++ -c l_mod1.cpp
l_impl.o:l_impl.cpp
      g++ -c l_impl.cpp
clean:
      rm *.o l_mod1 result.png
```

ПРИЛОЖЕНИЕ Б
ТЕСТОВЫЕ СЛУЧАИ

Входное выражение	Вывод программы	Корректность выполнения
(a) (a)	list1 = list2	да
(a) (b)	list1 != list2	да
(asdvc(sad)(sadsa(sadgft))) (asdvc(sad)(sadsa(sadgft)))	list1 = list2	да
(asdvc(sad)(sadsa(sadgft))) (asdvc(sss)(sadsa(sadgft)))	list1 != list2	да
(a b c (d e (d))) (abc(de(d)))	list1 = list2	да
a a	list1 = list2	да