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. 9383 _____ Корсунов А.А.

Преподаватель _____ Попова Е.В.

Санкт-Петербург
2020

Цель работы.

Познакомиться и научиться работать с иерархическими списками и написать программу на языке программирования C++.

Основные теоретические положения.

Иерархический список представляет собой либо элемент базового типа EI (атомарным S-выражением или атомом), либо другой список, который так же может содержать либо атом, либо список, другими словами иерархический список содержит либо атом, либо список иерархических списков. Иерархические списки представляют либо графически, используя для изображения структуры списка двумерный рисунок, либо в виде одномерной скобочной записи.

Задание.

Вариант 6:

проверить иерархический список на наличие в нем заданного элемента (атома) x;

Ход работы.

После анализа поставленного задания была разработана программа с использованием ООП и рекурсии.

Классы, используемые в программе:

1) lisp – класс создаваемого иерархического списка. Данный класс содержит открытые свойства: tag (типа bool) и node (объединение из atom (типа base, где base – псевдоним char) и pair(псевдоним типа two_ptr, где two_ptr – класс, описание которого содержится во втором пункте);
2) two_ptr – класс, который служит вспомогательным типом для элементов иерархического списка. Данный класс содержит открытые свойства: hd

(типа lisp*) - указатель на голову списка, tl (типа lisp*) - указатель на хвост списка.

Функции программы:

1) main (файл main.cpp):

В данной функции создается требуемый иерархический список, предварительно освобождая память под него, считывается сам список с помощью функции read_lisp, после чего считывается искомый атом и объявляется переменная check типа bool, которой присвается значение false, данная переменная используется в рекурсивной функции для проверки списка на вхождение искомого элемента, затем вызывается функция find_func, которая и является вышеупомянутой рекурсивной функцией, после его вызова переменная check может измениться с false на true (если искомый элемент найден) и выводится соответствующее сообщение на экран, после чего с помощью функции destroy подчищается выделенная под иерархический список память. (в данной функции закомментирована 27-ая строчка как write_lisp(&m); - она пригодится, если появится необходимость вывести веденный список на экран с помощью функции write_lisp).

Последующие функции объявлены в файле List.h, а описаны в List.cpp

2) head:

Данная функция возвращает голову иерархического списка (предусмотрен вывод ошибки при неправильном переданном аргументе в функцию).

3) tail:

Данная функция возвращает хвост иерархического списка (предусмотрен вывод ошибки при неправильном аргументе).

4) make_atom:

В данной функции создается объект типа lisp (предусмотрена проверка на правильность выделения памяти под этот объект), после чего в tag записывается true (говорит о том, что список — атом), а в node устанавливается значения аргумента функции, в конце функция возвращает атом.

5) destroy:

Данная функция рекурсивно освобождает выделенную под список память.

6) read_lisp, read_s_exptr, read_seq:

Данные функции служат для рекурсивного считывания иерархического списка.

7) write_lisp, write_seq:

Данные функции служат для рекурсивного вывода списка на экран.

8) isNull:

В данной функции происходит проверка переданного указателя на нулевой указатель nullptr.

9) isAtom:

В данной функции происходит проверка списка на атом.

10) cons:

В данную функцию в качестве аргументов передаются хвост и голова списка (учтены проверки на ошибки с соответствующем выводом на экран). Сама же функция создает новый иерархический список (также учтена проверка на правильность выделенной памяти), после чего происходит возвращение нового списка, предварительно сохранив в нем значения головы и хвсота.

11) fine_func:

Данная функция рекурсивной решает поставленную в лабораторной работе задачу поиска заданного атома. Для этого аргументами функции служат сам список lisp**x, искомый атом base y, переменная bool* check, которая установится в значение true при наличия вхождения искомого атома. Вначале с помощью функции isNull проверяется список на нулевой указатель, затем проверяют голову на атом: а) если атом, то происходит сравнения атома головы с искомым атомом y и если они равны, то функция завершается, предварительно установив check в true, иначе проверяется хвост на нулевой указатель, если является, то происходит выход из функции, в другом случае вызывается эта же функция, где уже передается хвост, б) если же не атом, то происходит проверка головы на нулевой указатель, где при true функция завершается, в другом случае вызывается эта же функция, где передается

следующая голова, после чего идет проверка хвоста на нулевой указатель, где при true происходит выход из функции, в другом же случае вызывается эта же функция, где передается хвост.

Пример работы программы.

№ п/п	Входные данные	Выходные данные	Комментарии
1	))(a b c d)	! List.Error 1	
2	(a(sd(fhd(hmf)asd(das)t))) m	Элемент найден	
3	(a(sd(fhd(hmf)asd(das)t))) p	Элемент не найден	

Иллюстрация работы программы.

*IDE – Code::Blocks 20.03

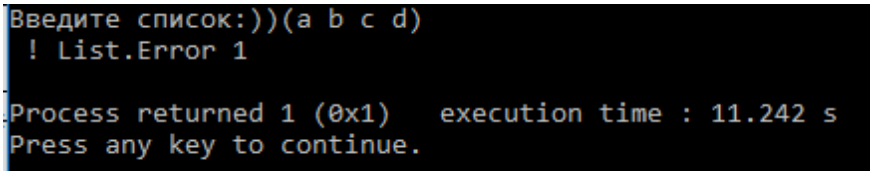

Рисунок 1 - Пример работы программы с входными данными №1

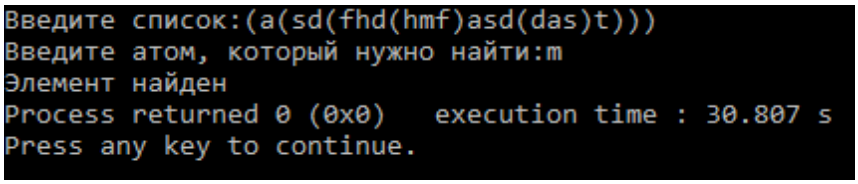

Рисунок 2 - Пример работы программы с входными данными №2

```
Введите список:(a(sd(fhd(hmf)asd(das)t)))
Введите атом, который нужно найти:p
Элемент не найден
Process returned 0 (0x0)   execution time : 23.226 s
Press any key to continue.
```

Рисунок 3 - Пример работы программы с входными данными №3

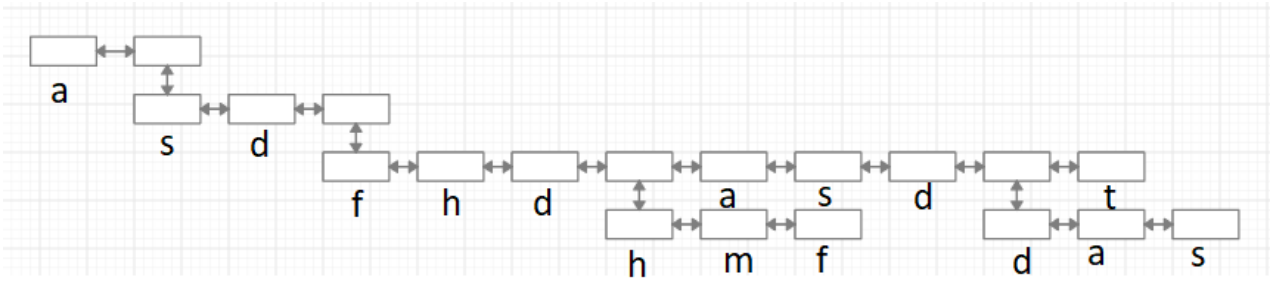

Рисунок 4 — Графическое представление иерархического списка, который поступает на вход в примерах №2 и №3

Выводы.

Произошло ознакомление с иерархическими списками и была написана программа на языке программирования C++.

ПРИЛОЖЕНИЕ А

Файл main.cpp

```
#include <iostream>
#include <locale.h>
#include "List.h"

using namespace std;

int main()
{
   setlocale(0,"");

   lisp *m = new lisp();
   cout<<"Введите список:";
   read_lisp(&m);
   cout<<"Введите атом, который нужно найти:";
   base x;
   cin>>x;
   bool check = false; // переменная, которая проверяет вхождение
   find_func(&m, x, &check);
   if (check)
   {
      cout << "Элемент найден";
   }
   else
   {
      cout << "Элемент не найден";
   }
```

```
    //write_lisp(&m);
    destroy(&m);
    return 0;
}
```

Файл List.h

```
#pragma once
#include <iostream>

typedef char base; // базовый тип элементов (атомов)

class lisp;
class two_ptr
{
public:
lisp *hd;
lisp *tl;
};      //end two_ptr;

class lisp
{
public:
bool tag; // true: atom, false: pair
union
{
base atom;
two_ptr pair;
} node;                 //end union node
```

```
};                          //end s_expr

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

// функции ввода:
void read_lisp(lisp** y);                   // основная
void read_s_expr(base prev, lisp** y);
void read_seq(lisp** y);

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

// функция поиска
void find_func(lisp** x, base y, bool* check);
```

ФайлList.cpp

```
#include "List.h"

using namespace std;
```

```
lisp **head(lisp** x)
{// PreCondition: not null (s)
   if ((*x) != nullptr) if (!isAtom(*x)) return &((*x)->node.pair.hd);
                        else { cerr << "Error: Head(atom) \n"; exit(1); }
                else { cerr << "Error: Head(nil) \n";
                        exit(1);
   }
}

bool isAtom(lisp* x)
{
   if(x == nullptr) return false;
        return x->tag;
}

bool isNull(lisp* x)
{
   return x==nullptr;
}

lisp** tail(lisp** x)
{// PreCondition: not null (s)
                if ((*x) != nullptr) if (!isAtom(*x))     return &((*x)->node.pair.tl);
                        else { cerr << "Error: Tail(atom) \n"; exit(1); }
                else { cerr << "Error: Tail(nil) \n";
                        exit(1);
   }
}
```

```
lisp* cons(lisp* h, lisp* t)
        // PreCondition: not isAtom (t)
        {lisp* p;
        if (isAtom(t)) { cerr << "Error: Cons(*, atom)\n"; exit(1);}
   p = new lisp;
   if (p == nullptr) {cerr << "Memory not enough\n"; exit(1); }
      p->tag = false;
      p->node.pair.hd = h;
      p->node.pair.tl = t;
      return p;
   }

lisp* make_atom(base x)
{
   lisp *s;
   s = new lisp;
   s->tag = true;
   s->node.atom = x;
   return s;
}

void destroy(lisp** s)
{
   if ((*s) != nullptr) {
                if (!isAtom(*s)) {
                        destroy (head(s));
                        destroy (tail(s));
                }
                delete *s;
                // s = nullptr;
```

```
        }
}

// ввод списка с консоли
        void read_lisp(lisp** y)
        {       base x;
                do cin >> x; while (x==' ');
                read_s_expr(x,y);
        } //end read_lisp

        void read_s_expr(base prev, lisp** y)
        { //prev - ранее прочитанный символ}
                        if ( prev == ')' ) {cerr << " ! List.Error 1 " << endl; exit(1); }
                else if ( prev != '(' ) *y = make_atom(prev);
                         else read_seq(y);
        } //end read_s_expr

        void read_seq(lisp** y)
        {       base x;
                lisp *p1, *p2;
                if (!(cin >> x)) {cerr << " ! List.Error 2 " << endl; exit(1);}
                else {
                        while  ( x==' ' ) cin >> x;
                        if ( x == ')' ) *y = nullptr;
                        else {
                                read_s_expr(x,&p1);
                                read_seq(&p2);
                                *y = cons(p1, p2);
                        }
                }
```

```
        } //end read_seq

// Процедура вывода списка с обрамляющими его скобками - write_lisp,
//  а без обрамляющих скобок - write_seq
        void write_lisp(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(lisp** x)
{//выводит последовательность элементов списка без обрамляющих его
скобок
        if (!isNull(*x)) {
                write_lisp(head (x));
                write_seq(tail (x));
        }
}

void find_func(lisp** x, base y, bool* check)
{
   if (isNull(*x)) return;
   if (isAtom(*(head(x))))
   {
      if((*head(x))->node.atom==y)
```

```
        {
                *check = true;
                return;
        }
        else
{
   if (isNull(*tail(x))) return;
   find_func(tail(x), y, check);
}
   }
   else
   {
      if (isNull(*head(x))) return;
      find_func(head(x), y, check);
      if (isNull(*tail(x))) return;
      find_func(tail(x), y, check);
   }
}
```