МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных »

Тема: Рекурсивная обработка иерархических списков

Студент гр. 9384		Давыдов Д.С.
Преподаватель		Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Разобраться в представлении и реализации таких структур, как линейный и иерархический список, способах их организации и рекурсивной обработки.

Задание.

Вариант 4.

Подсчитать число атомов в иерархическом списке; сформировать линейный список атомов, соответствующий порядку подсчета;

Теоретические сведения.

В данной лабораторной работе будет использована конструкция иерархического списка элементов базового типа El или S-выражений, где S выражание определяется рекурсивно с помощью определения линейного списка 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_list> ::= 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 непустого иерархического списка - это элемент размеченного объединения множества атомов и множества пар «голова-хвост».

Анализ задачи.

Составление линейного списка s1 из атомов нелинейного списка s2 осуществляется с помощью функции `toLinearConvert`.

Пользователь передает адрес первого элемента списка s. Функция проверяет является ли элемент нулевым указателем `isNull(s)`, если да — то

возвращаем нулевой указатель `return nullptr` , если нет то проверяем указывает ли «голова» на атом `!isAtom(head(s))`, если указывает то объединяем линейные списки `concat`, на которые указывают голова и хвост, путем рекурсии нашей функции (первым аргументом идет рекурсия с адресом линейного списка «головы» `toLinearConvert(H_list::head(s))`, т.е мы идем вглубь списка). Если элемент не проходит обе проверки, т.е его голова указывает на атом, то проверяем является ли указатель хвоста пустым `!isNull(tail(s))`: 1 – нет: вызываем конструктор линейного списка `L_list::cons` от головы, равной символу атома и хвоста, равного адресу линейного списка хвоста текущего элемента. Чтобы узнать адрес вызываем рекурсию нашей функции от указателя хвоста текущего элемента. 2 – да: вызываем конструктор линейного списка `L_list::cons` от головы, равной символу атома и хвоста, равного нулевому указателю.

Путем обхода всего нелинейного списка в конце мы получим линейный список с порядком равным считыванию атомов нелинейного списка.

Разработанный код смотреть в приложении А.

Выполнение работы.

Был разработан примитивный API, благодаря которому пользователь может выбрать, как ввести нелинейный список — из файла `readLispFile` или консоли `readLisp`, а также нужно ли выводить созданный линейный список из атомов нелинейного списка в файл.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 - результаты тестирования.

№ п/п	Входные данные	Выходные данные	Комментарии
1	//Введены из консоли с выводом в файл (a () j (o) b(p)())	//Представлены в файле test1.txt 1 : a(97) 2 : j(106)	Программа успешно считывает нелинейный список с пустыми линейными списками.

		3 : o(111) 4 : b(98) 5 : p(112)	

2	//Введены из файла без вывода в файл
	//Представлены в файле input2.txt
	(a j(k(p i) n) b)
		a (97) : j (106) :
		k (107) : p (112) :
		i (105) : n (110) :
		b (98) : nil
			Программа успешно находит все атомы в сложном нелинейном списке.

3	//Введены из консоли без вывода в файл (n 666 ! (h k) p)	n (110) : 6 (54) : 6 (54) : 6 (54) : ! (33) : h (104) : k (107) : p (112) : nil	Программа может считывать не только буквы, но и символы с цифрами.
4	//Введены из файла с выводом в файл (() a b (()()))	Представлены в файле test4.txt a (97) : b (98) : nil	Программа корректно считывает список, где некоторые адреса пар голова-хвост могут быть пустыми указателями одновременно и стоять как в начале так и в конце любого линейного списка.

Вывод

В ходе выполнения лабораторной работы была создана рабочая программа с необходимым API интерфейсом, которая может считывать из файла или командой строки необходимые символы, создавая тем самым из них нелинейный список, а затем нахождения в нем всех атомов и составления из них линейного списка. На практике были закреплены навыки по написанию и использованию рекурсивных функций, представлении и рекурсивной обработки иерархических и линейных списков

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММ.

Название файла: main.cpp

```
#include "hierarchical_list.h"
#include "linear_list.h"

using namespace std;
using namespace H_list;
using namespace L_list;

L_list::List toLinearConvert(const Lisp s);

int main (){

    H_list::Lisp s1;
    L_list::List s2;

    cout<<"Choose way to enter hierarchical List:\n1 - read from console\
n2 - read from file"<<endl;
    char c1;
    cin>>c1;
    switch(c1){
        case '1': readLisp(s1); break;
        case '2': readLispFile(s1); break;
        default: cout<<"You haven't choose anything"<<endl; return 1;
    }

    cout<<"Your hierarchical List:"<<endl;
    printLisp(s1);
    cout<<'\n';

    s2 = toLinearConvert(s1);

    cout<<"Number of elements/atoms"<<endl;
    cout << countElements(s2)<<endl;
```

```
    cout<<"Your lineal list of hierarchical atoms:"<<endl;
    s2 = toLinearConvert(s1);
    L_list::printList(s2);

     cout<<"Do you want to write it in file?\n1 - yes\nanything else -
no"<<endl;
    char c2;
    cin>>c2;
    if(c2 == '1') L_list::writeInFile(s2);
    else cout<<"List was not written to file"<<endl;

    H_list::destroy(s1);
    L_list::destroy(s2);

    return 0;
}

L_list::List toLinearConvert(const Lisp s) {
    if (isNull(s)){
        return nullptr;
    }

    else if (!isAtom(head(s))) {
                return  L_list::concat(toLinearConvert(H_list::head(s)),
toLinearConvert(H_list::tail(s)));
    }

    if (!isNull(tail(s))) {
                          return   L_list::cons(getAtom(H_list::head(s)),
toLinearConvert(H_list::tail(s)));
    }
    else {
        return L_list::cons(getAtom(H_list::head(s)), nullptr);
    }
}
```

Название файла: hierarchical_list.cpp

```
#include "hierarchical_list.h"

using namespace std;

namespace H_list {

    Lisp head(const Lisp s) {// PreCondition: not null (s)
        if (s != nullptr)
            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 == nullptr) return false;
        else return (s->tag);
    }

    bool isNull(const Lisp s) {
        return s == nullptr;
    }

    Lisp tail(const Lisp s) {
        // PreCondition: not null (s)
        if (s != nullptr)
            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 == nullptr) {
            cerr << "Memory not enough\n";
            exit(1);
        } else {
            p->tag = false;
            p->node.pair.hd = h;
            p->node.pair.tl = t;
            return p;
        }
    }
}

Lisp makeAtom(const Base x) {
    Lisp s;
    s = new S_expr;
    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 = 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 readLisp(Lisp &y) {
    cout << "Enter your List" << endl;
    Base x;
    do cin >> x; while (x == ' ');
    readS_expr(x, y);
}

void readS_expr(Base prev, Lisp &y) {
    //prev - ранее прочитанный символ
    if (prev == ')') {
        cerr << " ! List.Error 1 " << endl;
        exit(1);
    } else if (prev != '(') y = makeAtom(prev);
    else readSeq(y);
}

void readSeq(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 {
                readS_expr(x, p1);
                readSeq(p2);
                y = cons(p1, p2);
            }
        }
    }

    // процедура вывода списка с обрамляющими его скобками - printLisp,
    //  а без обрамляющих скобок - printSeq
    void printLisp(const Lisp x) {
        //пустой список выводитс¤ как ()
        if (isNull(x)) cout << " ()";
        else if (isAtom(x)) cout << ' ' << x->node.atom;
        else { //непустой список
            cout << " (";
            printSeq(x);
            cout << " )";
        }
    }

    void printSeq(const Lisp x) {
        //выводит последовательность элементов списка без обрамляющих его
скобок
        if (!isNull(x)) {
            printLisp(head(x));
            printSeq(tail(x));
        }
    }

    Lisp copyLisp(const Lisp x) {
        if (isNull(x)) return nullptr;
        else if (isAtom(x)) return makeAtom(x->node.atom);
        else return cons(copyLisp(head(x)), copyLisp(tail(x)));
    }

    Lisp concat(const Lisp y, const Lisp z) {
        if (isNull(y)) return copyLisp(z);
        else return cons(copyLisp(head(y)), concat(tail(y), z));
```

```
}

void readLispFile(Lisp &y) {
    std::string filePathIn;
    std::cout << "Enter input file:\n";
    std::cin >> filePathIn;
    std::ifstream in;
    in.open(filePathIn);
    if (!in.is_open()) {
        cout << "There is no such file" << endl;
        exit(0);
    }
    Base x;
    do in >> x; while (x == ' ');
    readS_exprFile(x, y, in);
}

void readS_exprFile(Base prev, Lisp &y, ifstream &infile) {
    //prev - ранее прочитанный символ
    if (prev == ')') {
        cerr << " ! List.Error 1 " << endl;
        exit(1);
    } else if (prev != '(') y = makeAtom(prev);
    else readSeqFile(y, infile);
}

void readSeqFile(Lisp &y, ifstream &infile) {
    Base x;
    Lisp p1, p2;

    if (!(infile >> x)) {
        cerr << " ! List.Error 2 " << endl;
        exit(2);
    } else {
        while (x == ' ') infile >> x;
        if (x == ')') y = nullptr;
        else {
            readS_exprFile(x, p1, infile);
            readSeqFile(p2, infile);
```

```
                y = cons(p1, p2);
            }
        }
    }
}
```

Название файла: linear_list.cpp

```
#include "linear_list.h"

using namespace std;

namespace L_list{

    // рекурсивный вывод
    void printList(List s){
        if (s != nullptr) {
            cout << *s->hd << " (" << int(*s->hd) << ") : ";
            printList(s->tl);
        } else { // s = nil
            cout << "nil\n";
        }
    }

    Base head(List s) {// PreCondition: not null (s)
        if (s != nullptr) return *s->hd;
        else {
            cerr << "Error: head(nil) \n";
            exit(1);
        }
    }

    List tail(List s) {// PreCondition: not null (s)
        if (s != nullptr) return s->tl;
        else {
            cerr << "Error: tail(nil) \n";
            exit(1);
        }
```

```
}

List cons(Base x, List s) {
    List p;
    p = new node;
    if (p != nullptr) {
        p->hd = new char;
        *p->hd = x;
        p->tl = s;
        return p;
    } else {
        cerr << "Memory not enough\n";
        exit(1);
    }
}

void destroy(List s) {
    if (s != nullptr) {
        destroy(tail(s));
        delete s->hd;
        delete s;
        // s = NULL;
    };
}

List concat(List s1, List s2) {
    if (s1 == nullptr) return s2;
    else return cons(head(s1), concat(tail(s1), s2));
}

void writeInFile(List s){
    std::string filePathOut;
    std::cout << "Enter output file:\n";
    std::cin >> filePathOut;
    std::ofstream out;
    out.open(filePathOut);
    List p = s;
    int i = 0;
    while (p != nullptr) {
```

```
            i++;
            out << i << " : " << *p->hd << "(" << int (*p->hd) << ") "<<
endl;
            p = p->tl;
        }
    }

    int countElements(List s){
        int count = 0;
        List p = s;
        while (p != nullptr){
            count++;
            p = p->tl;
        }
        return count;
    }
}
```

Название файла: hierarchical_list.h

Только структура:

```
namespace H_list{
 typedef char Base;   // базовый тип элементов (атомов)

    struct S_expr;
    struct  two_ptr
    {
        S_expr *hd;
        S_expr *tl;
    } ;

    struct S_expr {
        bool tag; // true: atom, false: pair
        union
        {
            Base atom;
            two_ptr pair;
        } node;
    };
```

```
}
```

Название файла: linear_list.h

Только структура:

```
namespace L_list{

    typedef char Base;
    struct node {
        Base *hd;
        node *tl;
        // constructor
        node ()
        {hd = nullptr; tl = nullptr;
        }
    };
    typedef node *List;
}
```