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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «АиСД»

Тема: Рекурсивная обработка иерархических списков

Студент гр. 9303 _____ Емельянов С.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

Цель работы.

Познакомиться с одной из часто используемых на практике нелинейных конструкций, способами её организации и рекурсивной обработки.

Задание.

Вариант 6

Проверить иерархический список на наличие в нём заданного элемента (атома) x;

Основные теоретические положения.

В практических приложениях возникает необходимость работы с более сложными, чем линейные списки, нелинейными конструкциями. Рассмотрим одну из них, называемую иерархическим списком элементов базового типа El или S-выражением.

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) >

Иерархические списки состоят из элементов различных уровней, при этом элементы нижних уровней подчинены элементам верхних уровней. Существует два вида иерархии списков: иерархия групп и элементов и иерархия элементов. Вид устанавливается конфигурацией.В списке с иерархией групп и элементов содержатся два вида элементов –группы и

собственно элементы. Группа обозначает узел, в который входят другие (подчиненные) группы и элементы, а элемент является конкретным объектом.

Выполнение работы.

Для решения поставленной задачи была разработана программа в среде Visual Studio Code на языке C++.

`int main()` – в данной функции происходит вызов функции `read_lisp(lisp& y)` для считывания списка, введённого пользователем с консоли, затем сразу же вызывается функция для вывода этого списка в консоль `write_lisp(const lisp x)`. После функция запрашивает у пользователя ввести элемент для поиска в списке, происходит вызов функции `start_search_item(const lisp x, base& item, bool& flag)`, после поиска выводится в консоль информация о поиске.

Для вывода иерархического списка в консоль были написаны следующие функции: `void write_lisp(const lisp x)`, `void write_seq(const lisp x)`.

Для считывания списка из консоли используются следующие функции: `void read_lisp(lisp& y)`, `void read_s_expr(base prev, lisp& y)`, `void read_seq(lisp& y)`.

За удаление списка после завершения программы отвечает функция `void destroy(lisp s)`.

За проверку элемента,что он не ссылается на NULL отвечает функция `bool isNull(const lisp s)`, а за проверку элемента, что он является атомом отвечает функция `bool isAtom(const lisp s)`.

За заполнение элемента списка атомом отвечает функция `lisp make_atom(const base x)`.

Функции `lisp head(const lisp s)`, `lisp tail(const lisp s)` возвращают указатели на голову и хвост списка.

Функция `lisp cons(const lisp h, const lisp t)` формирует связь между головой и хвостом списка за счёт структур `two_ptr` и `s_expr`.

Функции отвечающие за поиск элемента в списке `bool start_search_item(const lisp x, base& item, bool& flag)`, `void search_item (const lisp x, base& item, bool& flag)`.

Код программы приведён в приложение А, результаты тестирования в приложении Б.

Выводы.

Было изучено понятие иерархические списки и принципы работы с ним. Были приобретены навыки по работе с иерархическими списками с помощью рекурсивных функций.

Была реализована программа, включающая в себя рекурсивную функции для считывания, вывода и обработки иерархического списка. Также было проведено тестирование программы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdlib>
using namespace std;

typedef char base;

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

struct s_expr {
    bool tag;
    union
    {
        base atom;
        two_ptr pair;
    } node;
};

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);
```

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

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


lisp head(const lisp 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) 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"; exit(1); }
        else {
            cerr << "Error: Tail(nil) \n";
            exit(1);
        }
    }

    lisp cons(const lisp h, const lisp 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;
        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;

    };
}

void read_lisp(lisp& y)
{
    base x;
    cin >> x;
    if (x == '(')
        read_s_expr(x, y);
    else{
        cerr << " ! List.Error " << endl; exit(1);
    }
}

void read_s_expr(base prev, lisp& y)
{
     if (prev == ')') { cerr << " ! List.Error 1 " << endl;
exit(1); }
    else if (prev != '(') y = make_atom(prev);
    else read_seq(y);
}
void read_seq(lisp& y)
{
```

```
        base x;
        lisp p1, p2;

         if (!(cin >> x)) { cerr << " ! List.Error 2 " << endl;
exit(1); }
        else {
            if (x == ')') y = NULL;
            else {
                read_s_expr(x, p1);
                read_seq(p2);
                y = cons(p1, p2);
            }
        }
    }



    void write_lisp(const lisp x)
    {
        if (isNull(x)) cout << " ()";
        else if (isAtom(x)) cout << ' ' << x->node.atom;
        else
        {
            cout << " (";
            write_seq(x);
            cout << " )";
        }
    }

    void write_seq(const lisp x)
    {
        if (!isNull(x)) {
            write_lisp(head(x));
            write_seq(tail(x));
```

```
        }
    }

    void search_item (const lisp x, base& item, bool& flag){
        if(isAtom(x)){
            if(x->node.atom == item) flag = true;
        }
        else{
            if(!isNull(x)){
                search_item(head(x),item,flag);
                search_item(tail(x),item,flag);
            }
        }
    }

    bool start_search_item(const lisp x, base& item, bool& flag)
{
        search_item(x,item,flag);
        return flag;
    }

    int main()
    {
        lisp s1;
        bool flag = false;
        base item;
        cout << "введите первый список:" << endl;
        read_lisp(s1);
        cout << "введен первый список: " << endl;
        write_lisp(s1);
        cout<<endl;
        cin.clear();  cin.clear(); cin.ignore(32767, '\n');
        cout << "введите элемент:" << endl;
        cin>>item;
```

```
    if(start_search_item(s1,item,flag)){
        cout<<"Элемент(атом) есть в списке!"<<endl;
    }
    else{
         cout <<"Элемента(атома) нет в списке!"<< endl;
    }
    destroy(s1);
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	введите первый список: (acv) введите элемент: c	Элемент(атом) есть в списке!	Программа работает исправно.
2.	введите первый список: () введите элемент: f	Элемента(атома) нет в списке!	Программа работает исправно.
3.	введите первый список: (cxczc)(vxvd) введите элемент: g	Элемента(атома) нет в списке!	Программа работает исправно, но 2 список программа отбросила,так как возможно подать для обработки только 1 список.
4.	введите первый список: (((fxv)(afaf)(afv))vx(fsf)) введите элемент: j	Элемента(атома) нет в списке!	Программа работает исправно.
5.	введите первый список: afasfagx)	! List.Error	Программа работает исправно.