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 иерархических списков

Студент гр. 9382 _____ Кузьмин Д. И.

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 теоретические положения.

Иерархическим списком называется структура данных, элементы которой могут находиться как на разных уровнях иерархии, так и на одном. Ссылка на следующий элемент назвыается «хвост», а ссылка на элемент, лежащий на более никзмо уровне иерархии «голова». «Голова» может являться атомом, при этом «хвост» — нет. Структура непустого иерархического списка − это элемент размеченного объединения множества атомов и множества пар «голова-хвост».

Задание.

Вариант 12.

проверить идентичность двух иерархических списков.

Описание алгоритма

1) На первом шаге алгоритма сравниваются 2 элемента, и проверяется их тип(самое первое сравнение будет для самих списков). Если они атомы, то далее сравниваются их значения, и в случае, если они совпадают проверка оказывается успешной, в противном случае — нет. Также проверка является успешной, если оба элемента — пусты. Не успешной, если элементы имеют разный тип. Например, один — атом, другой — не атом. Или один — пустой, другой — атом.

2) Если же два элемента являются парами «голова-хвост», то далее проводятся те же самые операции (см п.1 и п.2), но уже для «головы» и «хвоста» элементов.

3) Алгортим завершается, в случае, если во время его выполнения какие-то два сравниваемых элемента оказались не равными. Или в том, случае, если все «проверки» были успешны.

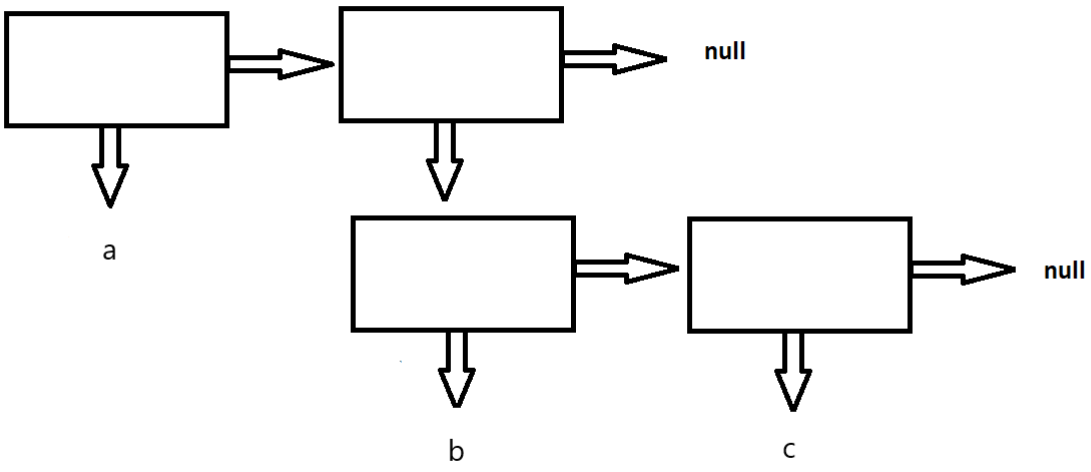
Описание функций и структур данных

1) Структура, определяющая элемент списка lisp. Состоит из:

bool tag — показывает, является ли элемент атомом; base atom – атомарное значение элемента, если он атом; s_expr* hd — указатель на «голову»; s_expr* tl – указатель на хвост.

2) Функции по работе со списком: lisp getHead(const lisp s) - возвращает голову списка; lisp getTail(const lisp s)- возвращает хвост списка; lisp makePair(const lisp h, const lisp t) - добавляет элемент(не атом) в список; lisp makeAtom(const base x) - добавляет атом в список - bool isAtom(const lisp s) проверка является ли элемент атомом; bool isNull(const lisp s) – проверка является ли список пустым; void destroy(lisp s)-удаляет список; base getAtom(const lisp s) – получает значение атома; void readLisp(lisp& y, FILE* f = nullptr) - считывание всего списка; void readSExpr(base prev, lisp& y, FILE* f = nullptr) — считывание отдельно взятого элемента; void readSeqExpr(lisp& y, FILE* f = nullptr); считывание подсписка; bool lispCmp(lisp l1, lisp l2) – реализует алгоритм проверки идентичности списков (см п. «Описание алгоритма»)

3) Графическое представление списка (a (b c))



Исходный код см. в приложении А.

Тестирование.

Результаты тестирование представлены в табл. 1.

Таблица 1 — результаты тестирования.

№ п/п	Входные данные	Выходные данные	Комментарий
0	a a	Списки идентичны	Простейший случай. Оба списка атомы, и их значения равны.
1	(a b c) (a b c)	Списки идентичны	Оба списка состоят из трех равных элементов.
2	() ()	Списки идентичны	Оба списка пусты, поэтому идентичны.
3	(a b (c d)) (a b c (d))	Списки не идентичны	Элемент b в первом списке имеет хвост (c d), а во втором тот же элемент имеет хвост (c (d)).
4	((((a)))) (((a)))	Списки не идентичны	Один и тот же элемент, но находится на разных уровнях иерархии.
5	(a a a) (a (a (a))))	Списки не идентичны	Три элемента, но в первом случае — на одном уровне, во втором — на разных.
6	(a b c) (a b c)	Списки идентичны	Пробелы в изначальном вводе не влияют на список.
7	(c b a) (a b c)	Списки не идентичны.	Равные элементы, но их порядок противоложный

Выводы.

Был изучен принцип устройства иерархических списков. Получены навыки разработки программы, работающей с иерархическими списками.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <cstdio>

using namespace std;

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

struct s_expr {
     bool tag;
     base atom;
     s_expr* hd;
     s_expr* tl;
};

typedef s_expr* lisp;

lisp getHead(const lisp s); // возвращается голову списка
lisp getTail(const lisp s);// возвращается хвост списка
lisp makePair(const lisp h, const lisp t);//добавляет элемент(не
атом) в список
lisp makeAtom(const base x);//добавляет атом в список
bool isAtom(const lisp s);//проверяет, является ли элемент атомом
bool isNull(const lisp s);//проверяется, является ли элемент
пустым
void destroy(lisp s);//уничтожает список
base getAtom(const lisp s);//возвращает значение атома элемента
// функции ввода:
void readLisp(lisp& y, FILE* f = nullptr); // считывание всего
списка
void readSExpr(base prev, lisp& y, FILE* f = nullptr);//отдельно
взятый элемент
void readSeq(lisp& y, FILE* f = nullptr);//для элементов-пар
"голова - хвост"
// функции вывода:
void printLisp(const lisp x); // для элемента
void printSeqExpr(const lisp x); // для элемента (пары "голова -
хвост")


lisp getHead(const lisp s)
{
     if (s != nullptr)
          if (!isAtom(s)) return s->hd;

          else {

               return nullptr;
          }
}
```

```
bool isAtom(const lisp s)
{
     if (s == NULL) return false;
     else return (s->tag);
}

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

lisp getTail(const lisp s)
{
     if (s != nullptr)
          if (!isAtom(s)) return s->tl;

          else {

               return nullptr;
          }
}

lisp makePair(const lisp h, const lisp t)

{
     lisp p;
     if (isAtom(t)) {
          cout << "Хвост не может являться атомом";
     }
     else {

          p = new s_expr;
          p->tag = false;
          p->hd = h;
          p->tl = t;
          return p;

     }
}

lisp makeAtom(const base x)
{
     lisp s;
     s = new s_expr;
     s->tag = true;
     s->atom = x;
     return s;
}

void destroy(lisp s) // удаление списка и освобождение памяти
{
     if (s != nullptr) {
          if (!isAtom(s)) {
```

```
                destroy(getHead(s));
                destroy(getTail(s));
           }
           delete s;

     };
}

base getAtom(const lisp s) // получение значения атома элемента
{
     if (!isAtom(s))
          cout << "не атом\n";
     else return (s->atom);
}

void readLisp(lisp& y, FILE* f) // считывание списка из файла либо
консоли
{
     base x = fgetc(f);


     while (x == ' ' || x == '\n') {

          x = (char)getc(f);
     }
     readSExpr(x, y, f);
}


void readSExpr(base prev, lisp& y, FILE* f) // считывание
следующего элемента и проверка является ли он атомом или нет
{
     if (prev == ')') cout << "Первый символ не может быть \")\"";
     else if (prev != '(') {
          y = makeAtom(prev);
     }
     else readSeq(y, f);
}

void readSeq(lisp& y, FILE* f) // считывание отдельного элемента
списка(не атома)
{
     base x = getc(f);
     lisp p1, p2;
     if (x == '\n') cout << " the end";
     else {
          while (x == ' ') x = fgetc(f);
          if (x == ')') y = nullptr;
          else {
               readSExpr(x, p1, f); //рекурсивный вызов для
следующего элемента
               readSeq(p2, f);
               y = makePair(p1, p2);
```

```
          }
     }
}

void printLisp(const lisp x) // вывод элемента
{
     if (isNull(x)) cout << "()";
     else if (isAtom(x)) cout << ' ' << x->atom;
     else {
          cout << " (";
          printSeqExpr(x);
          cout << " )";
     }
}

void printSeqExpr(const lisp x) //вывод элемента без скобок
{
     if (!isNull(x)) {
          printLisp(getHead(x));
          printSeqExpr(getTail(x));
     }
}

bool lispCmp(lisp l1, lisp l2) {


     if (l1 == nullptr && l2 == nullptr) { //проверка, являются ли
оба списка пустыми
          return true;
     }
     else if (isAtom(l1) && isAtom(l2)) { // проверка, являются ли
оба элемента атомами, если да, проверяется их значения
          cout << "Проверка значений атомов элементов " <<
getAtom(l1) << " и " << getAtom(l2) << "\n";
          return (getAtom(l1) == getAtom(l2));

     }
     else if (isAtom(l1) != isAtom(l2)) {
          cout << "Один элемент является атомом, другой - нет\n";
          return false;
     }
     else if (!isAtom(l1) && !isAtom(l2)) { //в случае, если
элементы не являются атомами, осуществляется проверка головы и
хвоста
          cout << "Элементы ";
          printLisp(l1);
          cout << " и";
          printLisp(l2);
          cout << " не являются атомами. Проверка головы и хвоста
элементов.\n";

          if (lispCmp(getHead(l1), getHead(l2))) {
               cout << "Проверка хвостов ";
```

```
                printLisp(getTail(l1));
                cout << " и ";
                printLisp(getTail(l2));
                cout << endl;
                if (lispCmp(getTail(l1), getTail(l2)))
                     return true;
           }
     }

}
int main() {

     setlocale(LC_ALL, "Russian");
     lisp l1, l2;

     char* c = new char[300];
     char* a = new char[20];
     while (strcmp(a, "1\n") && strcmp(a, "2\n")) {
          cout << "Проверка идентичности двух иерархических
списков. Введите 1 для ввода списков с консоли или 2 для ввода с
файла\n";

          fgets(a, 20, stdin);

     }
     if (strcmp(a, "1\n") == 0) {
          cout << "Считывание списка: " << endl;
          readLisp(l1, stdin);
          cout << "Считывание списка: " << endl;
          readLisp(l2, stdin);
     }
     else {
          FILE* f1 = fopen("Tests\\file.txt", "r+");
          FILE* f2 = fopen("Tests\\file.txt", "r+");
          cout << "Содержимое файла:\n";
          while (fgets(c, 200, f2))  cout << c;
          readLisp(l1, f1);
          readLisp(l2, f1);
          cout << "\n\n";
          fclose(f1);
          fclose(f2);
     }
     cout << "Список 1 - ";
     printLisp(l1);
     cout << endl;
     cout << "Список 2 - ";
     printLisp(l2);
     cout << endl;



     if (lispCmp(l1, l2)) {
```

```
            cout << "Списки идентичны\n";
        }
        else {

            cout << "Списки не идентичны\n";
        }

        destroy(l1);
        destroy(l2);

        system("pause");
        return 0;
}
```