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: Рекурсия

Студент гр. 8304 _____ Мухин А. М.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург

2019

Цель работы.

Ознакомиться с основными понятиями и приёмами рекурсивного программирования, получить навыки программирования рекурсивных процедур и функций на языке программирования C++.

Задание.

Вычислить глубину (число уровней вложения) иерархического списка как максимальное число одновременно открытых левых скобок в сокращённой скобочной записи списка; принять, что глубина пустого списка и глубина атомарного *S*-выражения равны нулю.

13й вариант.

Выполнение работы.

Входящий список будет поступать из аргумента командной строки и передаваться в поток, для дальнейшей его обработки. Далее мы объявляем экземпляр нашего списка, вызываем функцию чтения списка (инициализации) выводим значение потока на экран для удобства пользователя и вычисляем максимальную глубину нашего списка. И затем разрушаем его.

Структура списка состоит из булевого элемента, который является истиной, если наш элемент – атомарный, объединения, включающего в себя значение атома, если булев элемент – это истина и переменной типа two_ptr, которая отвечает за указатели на текущий элемент (голову) и последующий (хвост).

Функция bool isAtom(lisp const) принимает указатель на S-выражение и возвращает правду, если элемент является атомарным.

Функция lisp head(const lisp) возвращает ссылку на текущий элемент, если он не является атомарным.

Функция lisp tail(const lisp) возвращает ссылку на следующий элемент, если он не является атомарным.

Функция void destroy(lisp) рекурсивно удаляет все элементы списка.

Функция lisp cons(lisp const, lisp const) создаёт указатель на новое S-выражение и возвращает его.

Функция lisp make_atom(char const) создаёт указатель на атомарный элемент и заполняет его значением.

Функции void read_lisp(lisp&, std::stringstream&), void read_s_expr(char, lisp&, std::stringstream&), void read_seq(lisp&, std::stringstream) выполняют считывание того самого аргумента командной строки и преобразуют его в список.

Функция int dip(lisp const) выполняет рекурсивный обход списка и вычисляет максимальную глубину. Если это атомарный элемент, мы вычисляем глубину только хвоста, так как головы у атомарного элемента нет. Если же это не атомарный элемент, мы запускаем повторный обход, передавая в качестве параметра уже голову, и на выходе прибавляем к этому значению 1, так как это новый список. Тоже самое делаем с хвостом, но без прибавления единицы, так как мы ещё не знаем, что за элемент идёт дальше. В конце мы возвращаем максимальное значение и в конце концов получаем его для всего списка.

std::stringstream str – переменная, необходимая для того, чтобы работать со строкой как с потоком, позже мы перегрузим в неё значения аргумента командной строки.

isp lst – собственно и есть наш список.

Разработанный программный код см. в приложении А.

Тестирование.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	(a)	dip: 1
2.	(a(a(a)((a)a))(a(a)(a)))	dip: 4
3.	(a(b(cd)d)(d(f(gg)f(ds))))	dip: 4

4.	(a(b(cd())d)(d(f(g()g)f(ds))))	dip: 4
5.	(a(bc(de)d(er)q(er(er)ty)er(df)))	Dip: 4
6.	(a(a(a)((a)a))(a(a)(a))	! List.Error 2
7.	(a(bfs)f	! List.Error 2

Выводы.

Ознакомились с основными понятиями и приёмами рекурсивного программирования, получили навыки программирования рекурсивных процедур и функций на языке программирования C++ и создания списка. Научились работать с потоками ввода и вывода и направлять строку в поток с помощью std::stringstream.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab2.cpp

```
#include <fstream>
#include "iostream"
#include "lab2.h"

static                                                  std::ifstream
in("/home/mukhamux/CLionProjects/algoritms/lab2/tests.txt");
int main(int argc, char* argv[]) {
    std::stringstream str;
    str << argv[1];
    lisp lst = new s_expr;
    read_lisp(lst, str);
    std::cout << str.str();
    std::cout << " dip: " << dip(lst) << std::endl;
    destroy(lst);
    return 0;
}
```

Название файла: lab2.h

```
#ifndef LAB2_H
#define LAB2_H
```

```
#include <algorithm>
#include <sstream>
struct s_expr;

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

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

typedef s_expr* lisp;
void read_lisp(lisp& y, std::stringstream& s);
void read_s_expr(char prev, lisp& y, std::stringstream& s);
void read_seq(lisp& y, std::stringstream& s);


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

lisp head(const lisp s) {
    if (s != nullptr)
        if (!isAtom(s))
            return s->node.pair.hd;
        else {
            std::cerr << "Error: Head(atom) \n";
            exit(1);
        }
    else {
        std::cerr << "Error: Head(nil) \n";
        exit(1);
    }
}

lisp tail (const lisp s) {
    if (s != nullptr)
        if (!isAtom(s))
```

```
            return s->node.pair.tl;
        else {
            std::cerr << "Error: Tail(atom) \n";
            exit(1);
        }
    else {
        std::cerr << "Error: Tail(nil) \n";
        exit(1);
    }
}

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

lisp cons(lisp const h, lisp const t) {
    lisp p;
    if (isAtom(t)) {
        std::cerr << "Error: cons(*, atom) \n";
        exit(1);
    }
    else {
        p = new s_expr;
        p->tag = false;
        p->node.pair.hd = h;
        p->node.pair.tl = t;
        return p;
    }
}

lisp make_atom (char const x) {
    lisp s;
    s = new s_expr;
    s->tag = true;
    s->node.atom = x;
    return s;
}

void read_seq (lisp& y, std::stringstream& s) {
    char x;
```

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

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

void read_lisp(lisp& y, std::stringstream& s) {
    char x;
    do s >> x;
    while (x == ' ');
    read_s_expr (x, y, s);
}

int dip(lisp const x) {
    int head = 0, tail = 0;
    if (x != nullptr) {
        if (x->tag) {
            if (x->node.pair.tl != nullptr) {
                tail = dip(x->node.pair.tl);
            }
        }
        else {
```

```
                head = (x->node.pair.hd != nullptr) ? dip(x-
>node.pair.hd) + 1 : 0;
                tail = (x->node.pair.tl != nullptr) ? dip(x-
>node.pair.tl) : 0;
        }
        return std::max(head, tail);
    }
    return 0;
}

#endif
```