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: Иерархический список

Студент гр. 9303 _____ Махаличев Н.А.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с понятием «иерархический список», его приёмах и методах. Реализовать программу, решающую поставленную задачу на языке C++.

Задание.

Вариант 13.

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; например, глубина списка (a (b ( ) c) d) равна двум.

Основные теоретические положения.

Иерархический список, являющийся динамической структурой, предназначен для обработки сложных, нелинейных конструкций. Иерархический список элементов базового типа El или S-выражения определяется соответствующим типом данных S_expr (El) рекурсивно:

< S_expr (El) > ::= < Atomic (El) > | < L_list (S_expr (El)) >,

< Atomic (E) > ::= < El >.

Иерархический список, согласно определению, представляет собой или элемент базового типа El, называемый в этом случае атомом (атомарным S-выражением), или линейный список из S-выражений. Приведенное определение задает структуру непустого иерархического списка как элемента размеченного объединения [1] множества атомов и множества пар «голова»–«хвост» и порождает различные формы представления в зависимости от принятой формы представления линейного списка. Традиционно иерархические списки представляют или графически, используя для изображения структуры списка двухмерный рисунок, или в виде одномерной скобочной записи.

Зададим функциональную спецификацию иерархического списка, определив с помощью системы правил (аксиом) A1–A7, справедливых для всех

t типа El, всех u типа L_list(S_expr (El)), всех v типа Non_null_list (S_expr (El)), всех w типа S_expr (El), константу Nil, обозначающую пустой список, четыре ранее уже встречавшиеся базовые функции Null, Head, Tail, Cons, а также предикат Atom, проверяющий S-выражение на атомарность:

0) Nil: → Null_list;

1) Null: L_list (S_expr (El)) → Boolean;

2) Head: Non_null_list (S_expr (El)) → S_expr (El);

3) Tail: Non_null_list (S_expr (El)) → L_list (S_expr (El));

4) Cons: S_expr (El) ⊗ L_list (S_expr(El)) → Non_null_list (S_expr(El));

5) Atom: S_expr (El) → Boolean;

A1) Null (Nil) = true;

A2) Null (Cons (w, u)) = false;

A3) Head (Cons (w, u)) = w;

A4) Tail (Cons (w, u)) = u;

A5) Cons (Head(v), Tail(v)) = v.

A6) Atom (t) = true;

A7) Atom (u) = false.

Выполнение работы.

Создан класс Workspace, в котором выбирается метод ввода данных (с помощью консоли, или из файла). В зависимости от выбора пользователя вызывается метод void Console() или void File(string inputFile), в которых происходит построчное считывание данных из консоли и из файла соответственно.

После считывания очередной строки, с помощью метода string TransformLine(string line) для корректной работы программы в строке удаляются пробелы и табуляции, а также строка переворачивается. Данное изменение строки необходимо, чтобы работать с методами класса string, а именно с методом pop_back(), который удаляет последний символ строки.

Определен класс Node, в котором, при вызове финкции void read_lisp(lisp& y, string *line), строка представляется в виде запрограммированного иерархического списка. Функция void write_lisp(x, out_line) представляет хранящийся в памяти иерархический список в виде строки.

Поставленная задача решается рекурсивным обходом иерархического списка с помощью функций int depth(lisp x, int d, bool &isNotEmpty) и int depth_search(lisp x, int deep). При вызове depth_search() узнается глубина каждого подсписка иерархического списка и после проверки выводится максимальное значение полученной глубины.

Для проверки работоспособности программы создан класс Test. Принцип его работы следующий:

1) В директории ./Tests/ToCheck находятся файлы, в которых записаны строки, которые должна обработать программа, а в директории ./Tests/Answers находятся файлы с написанными в них ожидаемыми результатами работы программы. Стоит заметить, что название файла, содержащего строки для проверки, должно совпадать с названием файла, в котором записаны ожидаемые результаты работы программы.
2) С помощью цикла while каждый файл директории ./Tests/ToCheck открывается в программе, из него считываются данные и выполняется программа.
3) Выходные данные программы для каждого файла построчно сравниваются с данными файла с таким же названием в директории ./Tests/Answers. Если строки являются одинаковыми, программа выводит в консоль надпись «Correct answer» и строку, которая была проверена, иначе выводит «Not correct answer» и сравниваемые строки.

Код программы см. в приложении А. Тестирование программы см. в приложении Б.

Выводы.

В процессе выполнения лабораторной работы были изучено понятие динамических структур, в том числе иерархического списка, его приёмы и особенности.

Была разработана программа, определяющая глубину введённого иерархического списка путем рекурсивного обхода данного списка. Программа была протестирована с помощью написанного класса Test.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp

```
#include "../Headers/Workspace.h"
#include "../Headers/Test.h"

int main(int argc, char **argv){
    string arg;
    if (argc == 2){
        arg = argv[1];
    }
    if (arg == "test"){
        Test::Run();
    } else {
        Workspace a;
        a.ChooseInput();
    }
    return 0;
}
```

Файл Workspace.h

```
#ifndef WORKSPACE_H
#define WORKSPACE_H

#define RED "\033[1;31m"
#define ORANGE "\033[1;33m"
#define GREEN "\033[1;32m"
#define RESET "\033[0m"

#include <algorithm>
#include <iostream>
#include <string>
#include <fstream>
#include "Node.h"

using namespace std;

class Workspace{
public:
    Workspace();
    void ChooseInput();
    void Console();
    void File(string inputFile);
    static string ClearSymbols(string line);
    string TransformLine(string line);
    string line;
    string inputFile;
```

```
    ifstream input;
    ofstream output;
    bool isBrackets;
    lisp x;
};

#endif
```

Файл Workspace.cpp

```
#include "../Headers/Workspace.h"

Workspace::Workspace(){}

string Workspace::TransformLine(string line){
    reverse(line.begin(), line.end());
    line = ClearSymbols(line);
    return line;
}

void Workspace::ChooseInput(){
    Workspace method;
    cout << "Please select a data entry method" << endl;
    cout << "Press \"1\" if the console" << endl;
    cout << "Press \"2\" if the file" << endl;
    string choice;
    getline(cin, choice);
    switch (choice[0]){
        case '1':
            method.Console();
            break;
        case '2':
            cout << "Please enter filename: ";
            getline(cin, inputFile);
            method.File(inputFile);
            break;
        default:
            cout << "The entered digit does not match the input options"
<< endl;
            break;
    }
}

void Workspace::Console(){
    string out_line;
    cout << "Enter the line: ";
    getline(cin, line);
    string copy = TransformLine(line);
    read_lisp(x, &copy);
    write_lisp(x, out_line);
```

```
    int res = depth_search(x);
    destroy(x);
    cout << "List is:" << out_line << " - " << res;
    cout << endl;
}

void Workspace::File(string inputFile){
    input.open(inputFile);
    output.open("output.txt");
    if (input.is_open() && output.is_open()){
        while (getline(input, line)){
            string out_line;
            string copy = TransformLine(line);
            read_lisp(x, &copy);
            write_lisp(x, out_line);
            int res = depth_search(x);
            destroy(x);
            output << out_line << " - " << res << endl;
            cout << "List is:" << out_line << " - " << res;
            cout << endl;
        }
        cout << "The program has been completed. The result is written to
the file \"output.txt\"" << endl;
    } else {
        cout << "Can't open file" << endl;
    }
    input.close();
    output.close();
}

string Workspace::ClearSymbols(string line){
    for (int i = 0; i < line.length(); i++){
        if ((line[i] == ' ') || (line[i] == '\t')) {
            line.erase(i, 1);
            i--;
        }
    }
    return line;
}
```

Файл Node.h

```
#ifndef NODE_H
#define NODE_H

#include <string>
#include <iostream>

using namespace std;
```

```
typedef char Base;

class  s_expr;

class two_ptr{
public:
     s_expr *hd;
     s_expr *tl;
};

class s_expr{
public:
     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);
Base getAtom(const lisp s);
void read_lisp(lisp& y, string *line = NULL);
void read_s_expr(Base prev, lisp& y, string *line);
void read_seq(lisp& y, string *line);
void write_lisp(const lisp x, string &out);
void write_seq(const lisp x,  string &out);
int depth(lisp x, int d, bool &isNotEmpty);
int depth_search(lisp x, int deep = 0);

#endif
```

Файл Node.cpp

```
#include "../Headers/Node.h"

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){
    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;
    }
}

Base getAtom(const lisp s){
    if (!isAtom(s)){
        cerr << "Error: getAtom(s) for !isAtom(s) \n";
        exit(1);
    } else {
        return (s->node.atom);
    }
}

void read_lisp(lisp& y, string *line){
    Base x;
    if (line){
        x = (*line)[line->length()-1];
        line->pop_back();
    }
    read_s_expr(x, y, line);
}

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

```
void read_seq(lisp& y, string *line){
    Base x;
    lisp p1, p2;
    if (!line){
        cerr << " ! List.Error 2 " << endl;
        exit(1);
    } else {
        x = (*line)[line->length()-1];
        line->pop_back();
    }
    if (x == ')'){
        y = NULL;
    } else {
        read_s_expr ( x, p1, line);
        read_seq ( p2, line);
        y = cons (p1, p2);
    }
}

void write_lisp(const lisp x, string &out){
    if (isNull(x)){
        out += " ()";
    } else {
        if (isAtom(x)){
            out += ' ';
            out += x->node.atom;
        } else{
            out += " (";
            write_seq(x, out);
            out += " )";
        }
    }
}

void write_seq(const lisp x, string &out){
    if (!isNull(x)){
        write_lisp(head(x), out);
        write_seq(tail(x), out);
    }
}

int depth(lisp x, int d, bool &isNotEmpty){
    if (!isNull(x)){
        if (isAtom(x)){
            isNotEmpty = true;
            return d;
        } else {
            depth(head(x), d+1, isNotEmpty);
        }
    }
```

```
    return 0;
}

int depth_search(lisp x, int deep){
    string current;
    write_lisp(x, current);
    cout << "Checking - " << current << endl;
    bool isNotEmpty = false;
    int res = deep;
    if (!isNull(x)){
        if (isAtom(x)){
            return res;
        } else {
            int current = depth(head(x), 0, isNotEmpty);
            int current1 = depth_search(head(x), deep+1);
            int current2 = depth_search(tail(x), deep);
            if (isNotEmpty){
                res += current;
                if (current1 > res){
                    res = current1;
                }
                if (current2 > res){
                    res = current2;
                }
            }
        }
    }
    return res;
}
```

Файл Test.h

```
#ifndef TESH_H
#define TESH_H

#include <iostream>
#include <dirent.h>
#include <string>
#include "Workspace.h"

using namespace std;

class Test{
public:
    Test();
    static void Run();
    void File(char *name);
    void Compare(string inputFile);
    string lineOutput;
    string lineAnswers;
```

```
    string inputFile;
    string directory;
    ifstream inputFromOutput;
    ifstream inputFromAnswers;
    Workspace workspace;
};

#endif
```

Файл Test.cpp

```
#include "../Headers/Test.h"

Test::Test(){}

void Test::Run(){
    Test test;
    DIR *myDir = opendir("./Tests/ToCheck");
    if(myDir == NULL) {
        perror("Unable to open directory ./Tests/ToCheck");
        return;
    }
    struct dirent *entry;
    while ((entry = readdir(myDir)) != nullptr){
        test.File(entry->d_name);
    }
    closedir(myDir);
}

void Test::File(char *name){
    inputFile = name;
    if ((inputFile != ".") && (inputFile != "..")){
        directory = "./Tests/ToCheck/";
        cout << ORANGE "Comparing file " RESET << inputFile << endl;
        workspace.File(directory + inputFile);
        Test::Compare(inputFile);
    }
}

void Test::Compare(string inputFile){
    inputFromOutput.open("output.txt");
    inputFromAnswers.open("./Tests/Answers/" + inputFile);
    if (inputFromOutput.is_open() && inputFromAnswers.is_open()){
        while (getline(inputFromOutput, lineOutput) &&
getline(inputFromAnswers, lineAnswers)){
            if (lineOutput == lineAnswers){
                cout << GREEN "Correct answer - " RESET << lineAnswers <<
endl;
            } else {
```

```
                cout << RED "Not correct answer - " RESET << lineAnswers
<< RED "\n - Program response - " RESET << lineOutput << endl;
            }
        }
    }
    inputFromOutput.close();
    inputFromAnswers.close();
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица Б.1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1	(a (b (c (d (e f) g) h) r) q)	(a (b (c (d (e f) g) h) r) q) - 5	Получен ожидаемый вывод.
2	()	() - 0	Получен ожидаемый вывод
3	a	a - 0	Получен ожидаемый вывод
4	(a (b () c) d)	(a (b () c) d) - 2	Получен ожидаемый вывод
5	(a (f (g z) g))	(a (f (g z) g)) - 3	Получен ожидаемый вывод
6	((l k) (a () e))	((l k) (a () e)) - 2	Получен ожидаемый вывод