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. 9381 _____ Камакин Д.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Познакомиться с основными функциями создания и обработки иерархического списка. Изучить синтаксис языка программирования C++.

Задание.

Вариант №6.

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

Основные теоретические положения.

Согласно рекурсивному определению, иерархический список — такой список, элементами которого могут являться иерархические списки.

Традиционно иерархические списки представляют или графически, или в виде скобочной записи. На рис.1 приведен пример графического изображения иерархического списка. Соответствующая этому изображению сокращенная скобочная запись – это (a (b c) d e).

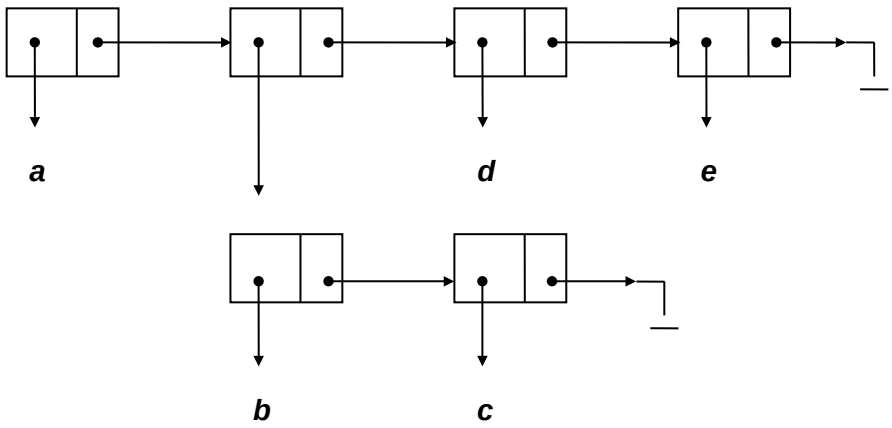


Рисунок 1 - Пример представления иерархического списка в виде двумерного рисунка

Согласно приведенному определению иерархического списка, структура непустого иерархического списка — это элемент размеченного объединения множества атомов и множества пар «голова-хвост».

Описание алгоритма.

В функции main запускаем функцию void execProgram(), в которой расположен основной while-цикл для обработки команд пользователя. Рекурсивно считываем иерархический список с клавиатуры/файла, после чего проверяем, есть ли введённый пользователем атом x в списке. Для этого были объявлены и определены следующие функции:

- *void readList(ListP &list, std::istream &stream)* - Пропускает пробелы и вызывает следующую функцию. На вход ListP &list — ссылка на список и *std::istream &stream* — поток ввода. Ничего не возвращает.
- *void readExp(char prev, ListP &list, std::istream& stream)* - Создает атомы и вызывает следующую функцию с номер для дальнейшей обработки строки. На вход подается *char prev* - последний считанный символ, *ListP& list* - ссылка на список и *std::istream& stream* - поток ввода. Ничего не возвращает.
- *void readRecursion(ListP &list, std::istream& stream)* - Рекурсивная функция, которая обрабатывает строку и создает и скрепляет узлы между собой. На вход подается *ListP &list* - *ссылка* на список и *std::istream& stream* — поток ввода. Ничего не возвращает.
- *ListP makeAtom(char symbol)* - Возвращает структуру с атомом. На вход подается *char symbol* - имя атома.
- *ListP addNode(ListP head, ListP tail)* - Присоединяет узел к списку. На вход подается *ListP head* - указатель на голову и *ListP tail* - хвост узла. Возвращает указатель на присоединенный узел.
- *void output(ListP list)* - Выводит на экран список атомов и узлов в виде скобок. На вход подается *ListP list* - указатель на список. Ничего не возвращает.
- *void outputRecursion(ListP list)* - Выводит на экран список атомов и узлов в виде скобок. Сама функция выводит непосредственно хвост узла. На вход подается *ListP list* - указатель на список. Ничего не возвращает.

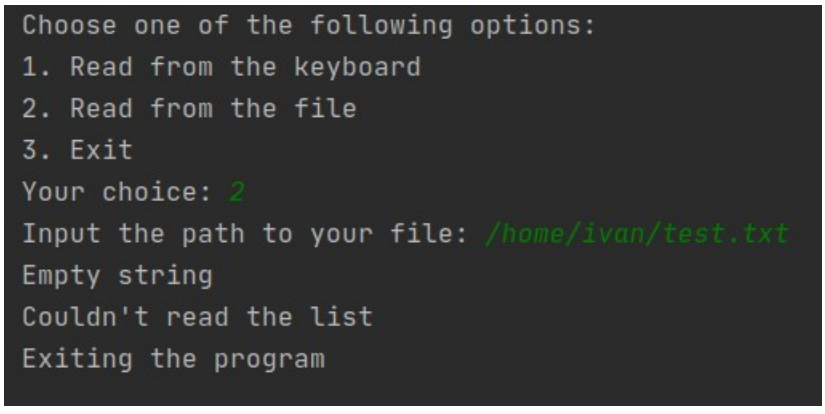
- *bool isAtom(ListP list)* - Проверяет является ли этот элемент списка атомом или узлом. На вход подается *ListP list* - указатель на структуру. Возвращает bool (1 — атом, иначе 0).
- *ListP getTail(ListP list)* - Возвращает указатель на tail списка. На вход подается *ListP list* - указатель на элемент списка.
- *ListP getHead(ListP list)* - Возвращает указатель на head списка. На вход подается *ListP list* - указатель на элемент списка.
- *bool isNull(ListP list)* - Проверяет является ли список пустым. На вход подается *ListP list* - указатель на элемент списка. Возвращает 1 – если пуст, 0 – если не пуст.
- *bool findAtom(ListP list, char atom, bool isIn)* - Возвращает 1, если элемент найден, иначе 0. На вход подается *ListP list* - указатель на начало иерархического списка, *char atom* - искомый атом x, *bool isIn* - начальное значение результата поиска.

Вывод.

В ходе выполнения лабораторной работы был изучен такой вид данных, как иерархический список. Была реализована программа, которая считывает, обрабатывает и выводит иерархический список.

Тестовые задания.

Входные данные	Исходные данные
(a (b)) b	Choose one of the following options: 1. Read from the keyboard 2. Read from the file 3. Exit Your choice: 1 Input your list: (a(b)) Your list: (a(b)) Input the Atom: b CHECK_HEAD_TO_ATOM_X function is starting The last symbol is: a Check if it's the atom b That's not the atom. Continue searching CHECK_HEAD_TO_ATOM_X function is ending CHECK_TAIL_TO_ATOM_X function is starting CHECK_HEAD_TO_ATOM_X function is starting CHECK_HEAD_TO_ATOM_X function is starting The last symbol is: b Check if it's the atom b That's the atom. Ending functions. CHECK_HEAD_TO_ATOM_X function is ending CHECK_TAIL_TO_ATOM_X function is starting CHECK_TAIL_TO_ATOM_X function is ending CHECK_HEAD_TO_ATOM_X function is ending CHECK_TAIL_TO_ATOM_X function is starting CHECK_TAIL_TO_ATOM_X function is ending CHECK_TAIL_TO_ATOM_X function is ending The Atom is in the list

(x y z) z	The Atom is in the list
(1 2 3 (4)) 3	The Atom is in the list
(d f g (f g h)) g	The Atom is in the list
Проверка на пустой файл	

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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

#define RED "\033[31m"
#define GREEN "\033[32m"
#define RESET "\033[0m"

using namespace std;

int getAction(); // Output the information and returns input
void execProgram(); // Main while
bool findAtom(ListP list, char atom, bool isIn); // Recursion
searching for the atom

bool findAtom(ListP list, char atom, bool isIn) {
    if (isNull(list) || isIn)
        return isIn;

    static int space = 0; // Static counter for the depth of the
recursion

    if (isAtom(list)) {
        printSpace(space);
        cout << "The last symbol is: " << getAtom(list) << '\n' <<
                "Check if it's the atom " << atom << '\n';

        if (getAtom(list) == atom) {
            cout << "That's the atom. Ending functions." << '\n';
            return true;
        }

        cout << "That's not the atom. Continue searching" << '\n';
    } else {
        cout << "CHECK_HEAD_TO_ATOM_X function is starting" << '\n';
        space++;
        printSpace(space);
        isIn = findAtom(getHead(list), atom, isIn);
        space--;
        printSpace(space);
        cout << "CHECK_HEAD_TO_ATOM_X function is ending" << '\n';

        cout << "CHECK_TAIL_TO_ATOM_X function is starting" << '\n';
        space++;
        printSpace(space);
        isIn = findAtom(getTail(list), atom, isIn);
        space--;
        printSpace(space);
        cout << "CHECK_TAIL_TO_ATOM_X function is ending" << '\n';
    }

    return isIn;
```

```
}

int getAction() {
    int action = 0;

    cout << "Choose one of the following options: " << '\n' <<
         "1. Read from the keyboard" << '\n' <<
         "2. Read from the file" << '\n' <<
         "3. Exit" << '\n' <<
         "Your choice: ";
    cin >> action;

    return action;
}

void execProgram() {
    int action;
    char atom;
    ifstream file;
    ListP list = nullptr;
    string fileName;

    while ((action = getAction()) != 3) {
        switch (action) {
            case 1:
                cout << "Input your list: ";
                readList(list, std::cin);
                break;
            case 2:
                cout << "Input the path to your file: ";
                cin >> fileName;
                file.open(fileName);

                if (!file.is_open()) {
                    cout << "Wrong file" << '\n';
                    continue;
                }

                readList(list, file);
                file.close();
                break;
            default:
                cout << "Exiting the program" << '\n';
                freeMemory(list);
                return;
        }

        cout << "Your list: ";
        output(list);
        cout << '\n' << "Input the Atom: ";
        cin >> atom;

        cout << (findAtom(list, atom, false) ? GREEN "The Atom is in
the list" :
                 RED "The Atom is not in the list") <<
             RESET << '\n' << '\n';
        freeMemory(list);
```

```
    }

    cout << "Exiting the program" << '\n';
}

int main() {
    execProgram();
    return 0;
}
```

Название файла: structures.h

```
#ifndef LISTS_STRUCTURES_H
#define LISTS_STRUCTURES_H

struct List;

typedef List* ListP;

struct Pair {
    ListP head;
    ListP tail;
};

struct List {
    bool atom;

    union {
        char atom;
        Pair pair;
    } Node;
};

#endif
```

Название файла: list.h

```
#ifndef LISTS_LIST_H
#define LISTS_LIST_H

#include <iostream>
#include "structures.h"

#define SPACE ' '

ListP getHead(ListP list); // Get a head of the list
ListP getTail(ListP list); // Get a tail of the list
bool isAtom(ListP list); // Check if an element is an atom
char getAtom(ListP list); // Get an atom of the element
bool isNull(ListP list); // Check if an element is null

ListP addNode(ListP head, ListP tail); // Add a node to the list
ListP makeAtom(char symbol); // Make an atom

void readList(ListP &list, std::istream &stream); // Read list
void readExp(char prev, ListP &list, std::istream& stream); //
Recursion (builds atoms)
void readRecursion(ListP &list, std::istream& stream); // Recursion
reading
```

```
void output(ListP list); // Output recursion
void outputRecursion(ListP list); // Output recursion

void printSpace(int amount); // Print spaces (or any character) for
recursion
void freeMemory(ListP list); // Free memory of the list

#endif
```

Название файла: list.cpp

```
#include "list.h"

ListP getHead(ListP list) { // Returns the head of the element
    if (!list || isAtom(list))
        return nullptr;
    return list->Node.pair.head;
}

bool isAtom(ListP list) { // Check if an element is an atom
    if (!list)
        return false;
    return (list->atom);
}

char getAtom(ListP list) { // Returns atom
    if (isAtom(list))
        return list->Node.atom;
    else return '\0';
}

bool isNull(ListP list) { // Check if an element is null
    return list == nullptr;
}

ListP getTail(ListP list) { // Returns the tail of the element
    if (!list || isAtom(list))
        return nullptr;
    return list->Node.pair.tail;
}

ListP addNode(ListP head, ListP tail) { // Builds a node to the list
from a head and a tail
    if (isAtom(tail))
        return nullptr;

    auto list = new List;
    list->atom = false;
    list->Node.pair.head = head;
    list->Node.pair.tail = tail;
    return list;
}

ListP makeAtom(char symbol) { // Builds an atom from the character
    auto list = new List;
    list->atom = true;
    list->Node.atom = symbol;
```

```
    return list;
}

void readList(ListP &list, std::istream& stream) { // Start of the
recursion
    char symbol;

    do {
        stream >> symbol;
    } while (symbol == ' ');

    readExp(symbol, list, stream);
}

void readExp(char prev, ListP& list, std::istream& stream) { //
Recursion reading (builds atoms)
    if (prev != '(')
        list = makeAtom(prev);
    else
        readRecursion(list, stream);
}

void readRecursion(ListP& list, std::istream& stream) { // Recursion
reading (builds the list)
    char symbol;
    ListP p1, p2; // head and tail

    stream >> symbol;

    if (symbol == ')')
        list = nullptr;
    else {
        readExp(symbol, p1, stream);
        readRecursion(p2, stream);
        list = addNode(p1, p2);
    }
}

void output(ListP list) { // Recursion output
    if (isNull(list)) // Empty list is ()
        std::cout << "()";

    else if (isAtom(list))
        std::cout << list->Node.atom;
    else {
        std::cout << '(';
        outputRecursion(list);
        std::cout << ')';
    }
}

void outputRecursion(ListP list) { // Recursion output
    if (isNull(list))
        return;
    output(getHead(list));
    outputRecursion(getTail(list));
}
```

```
void freeMemory(ListP list) {
    if (!list)
        return;

    if (!isAtom(list)) {
        freeMemory(getHead(list));
        freeMemory(getTail(list));
    }

    delete list;
}

void printSpace(int amount) {
    for (int i = 0; i < amount; i++)
        std::cout << SPACE;
}
```