

**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

Чухарев И.А.

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 № 15.

проверить структурную идентичность двух иерархических списков (списки структурно идентичны, если их устройство (скобочная структура и количество элементов в соответствующих (под)списках) одинаково, при этом атомы могут отличаться);

### **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).

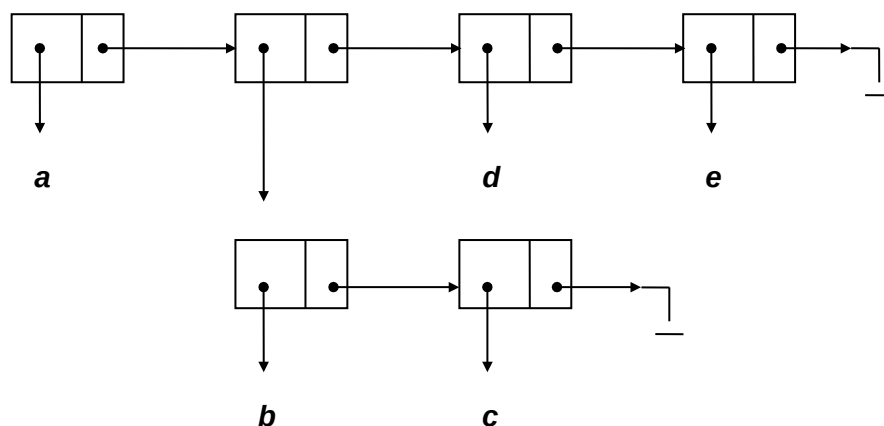


Рисунок 1 - Пример представления иерархического списка в виде двумерного рисунка

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

### **Описание алгоритма и функций.**

В функции `main` запускаем функцию `void execProgram()`, в которой расположен основной `while`-цикл для обработки команд пользователя.

Для структурного сравнения двух списков реализован следующий алгоритм: в рекурсивной функции используем следующие проверки:

1. Убеждаемся в том, что один из списков не является пустым (при существовании второго).
2. Один из списков не является атомом при обратной ситуации у второго.

Если проверки пройдены, тогда можно переходить к следующему звену, либо возвращать `true`, если списки одновременно пусты (означает, что проверка окончена).

Были объявлены и определены следующие функции:

- `void readList(ListP &list, std::istream &stream)` - Пропускает пробелы и вызывает следующую функцию. На вход подаётся `ListP &list` — ссылка на список и `std::istream &stream` — ссылка поток ввода.
- `void readExp(char prev, ListP &list, std::istream& stream)` - Создает атомы и вызывает следующую функцию для дальнейшей обработки строки. На вход подается `char prev` - последний

считанный символ, *ListP & list* - ссылка на список и *std::istream & stream* - поток ввода.

- *void readRecursion(ListP &list, std::istream & stream)* - Рекурсивная функция, которая обрабатывает строку, создает и скрепляет узлы между собой. На вход подается *ListP &list* - ссылка на список и *std::istream & stream* — поток ввода.
- *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.
- *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.
- *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 isMatch(ListP left, ListP right)* - Возвращает true, если списки структурно идентичны, иначе false. На вход подается два списка ListP.

### **Вывод.**

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

## Тестирование.

| Входные данные   | Исходные данные                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (abc)<br>(a(b)c) | <pre> The first list: (abc) The second list: (a(b)c)  Start comparing The First list: abc) and the second list: a(b)c) Keep comparing in recursion The First list: bc) and the second list: (b)c) Keep comparing in recursion The First list: c) and the second list: b)c) Keep comparing in recursion The First list: ) and the second list: )c) Returning The First list: ) and the second list: )c) Keep comparing in recursion The First list:  and the second list: c) Returning The First list:  and the second list: c) Returning The First list:  and the second list: c) One is atom and the other is not. Return The lists don't match </pre> |
| )(abc)           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

Choose one of the following options:

1. Read from the keyboard

2. Read from the file

3. Exit

Your choice: 1

Input the first list: )(abc)

Error during reading, please try again

(abc)

(cbd)

```
The First list: (abc)
The second list: (cbd)

Start comparing
The First list: abc) and the second list: cbd)
Keep comparing in recursion
The First list: bc) and the second list: bd)
Keep comparing in recursion
The First list: c) and the second list: d)
Keep comparing in recursion
The First list: ) and the second list: )
Returning
The First list: ) and the second list: )
Keep comparing in recursion
The First list:  and the second list:
Returning
The First list:  and the second list:
Returning
The First list:  and the second list:
Keep comparing in recursion
The First list:  and the second list:
Returning
The First list:  and the second list:
Returning
The First list:  and the second list:
Keep comparing in recursion
The First list:  and the second list:
Returning
The First list:  and the second list:
Returning
The lists match
```



(c(b(d))e)

(a(b)e)

The first list: (c(b(d))e)

The second list: (a(b)e)

Start comparing

The First list: c(b(d))e) and the second list: a(b)

Keep comparing in recursion

The First list: (b(d))e) and the second list: (b)e)

Keep comparing in recursion

The First list: b(d))e) and the second list: b)e)

Keep comparing in recursion

The First list: (d))e) and the second list: )e)

Returning

The First list: (d))e) and the second list: )e)

Keep comparing in recursion

The First list: d))e) and the second list: e)

Returning

The First list: d))e) and the second list: e)

Returning

The First list: d))e) and the second list: e)

Keep comparing in recursion

The First list: ))e) and the second list: )

One node is null and the other not

The lists don't match

(c(b(d))e)  
(a(e(q))s)

```
The first list: (c(b(d))e)
The second list: (a(e(q))s)

Start comparing
The First list: c(b(d))e) and the second list: a(e(q))s)
Keep comparing in recursion
The First list: (b(d))e) and the second list: (e(q))s)
Keep comparing in recursion
The First list: b(d))e) and the second list: e(q))s)
Keep comparing in recursion
The First list: (d))e) and the second list: (q))s)
Returning
The First list: (d))e) and the second list: (q))s)
Keep comparing in recursion
The First list: d))e) and the second list: q))s)
Returning
The First list: d))e) and the second list: q))s)
Returning
The First list: d))e) and the second list: q))s)
Keep comparing in recursion
The First list: ))e) and the second list: ))s)
Keep comparing in recursion
The First list: )e) and the second list: )s)
Returning
The First list: )e) and the second list: )s)
Keep comparing in recursion
The First list: e) and the second list: s)
Returning
The First list: e) and the second list: s)
```

|  |  |
|--|--|
|  |  |
|--|--|

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

**Название файла: lb2.cpp**

```
#include <iostream>
#include <fstream>

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

struct List;

typedef List* ListP;

struct Pair {
    ListP head;
    ListP tail;
};

struct List {
    bool atom;

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

bool readExp(char prev, ListP& list, std::istream& stream); //
Recursion reading (builds atoms)
void output(ListP list, std::string &string); // Recursion output

static auto space = 0;

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

    return list->atom;
}

ListP getHead(ListP list) { // Returns the head of the element
    if (!list || isAtom(list))
        return nullptr;

    return list->Node.pair.head;
}

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; // allocate new node
    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 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);
    }
}

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

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

    do {
        stream >> symbol;

```

```

        } while (symbol == ' ');

        return readExp(symbol, list, stream);
    }

void outputRecursion(ListP list, std::string &string) { // Recursion
output
    if (isNull(list))
        return;

    output(getHead(list), string);
    outputRecursion(getTail(list), string);
}

void output(ListP list, std::string &string) { // Recursion output
    if (isNull(list)) // Empty list is ()
        string += "()";

    else if (isAtom(list))
        string += list->Node.atom;
    else {
        string += '(';
        outputRecursion(list, string);
        string += ')';
    }
}

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

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

    delete list;
}

int getMin(int left, int right) {
    return left > right ? right : left;
}

bool isMatch(ListP left, ListP right, std::string &strLeft,
std::string &strRight) {
    space++;
    auto length = getMin(strLeft.size(), strRight.size());
    auto offset = space > length ? length : space;
    std::cout << "The First list: " << strLeft.substr(offset,
strLeft.size() - offset) <<
        " and the second list: " << strRight.substr(offset,
strRight.size() - offset) << '\n';

    if (isNull(left) != isNull(right)) { // if one of them is null and
the second not then they don't have the same structure
        space--;
        std::cout << "One node is null and the other not" << '\n';

```

```

        return false;
    }

    if (isAtom(left) != isAtom(right)) { // check for the atom in
nodes
        space--;
        std::cout << "One is atom and the other is not. Return" << '\
n';
        return false;
    }

    if (isNull(left) && isNull(right)) { // if both are checked and no
errors occurred then they have the same structure
        space--;
        std::cout << "Returning" << '\n';
        return true;
    }

    std::cout << "Keep comparing in recursion" << '\n';
    return isMatch(getTail(left), getTail(right), strLeft, strRight)
&&
        isMatch(getHead(left), getHead(right), strLeft,
strRight);
}

int getAction() {
    int action = 0;

    std::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: ";
    std::cin >> action;

    return action;
}

void execProgram() {
    int action;
    ListP listA = nullptr;
    ListP listB = nullptr;
    std::ifstream file;
    std::string fileName;
    std::string first;
    std::string second;

    while ((action = getAction()) != 3) {
        switch (action) {
            case 1:
                std::cout << "Input the first list: ";
                if (!readList(listA, std::cin)) {
                    std::cout << "Error during reading, please try
again" << '\n';
                    return;
                }
            case 2:
                // ... (code for case 2)
            case 3:
                // ... (code for case 3)
        }
    }
}

```

```

        std::cout << "Input the second list: ";
        if (!readList(listB, std::cin)) {
            std::cout << "Error during reading, please try
again" << '\n';
            return;
        }
        break;
    case 2:
        std::cout << "Input the path to your file: ";
        std::cin >> fileName;
        file.open(fileName);

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

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

output(listA, first);
output(listB, second);

std::cout << "The first list: ";
std::cout << first;
std::cout << '\n';
std::cout << "The second list: ";
std::cout << second;
std::cout << '\n' << '\n';

std::cout << "Start comparing" << '\n';
std::cout << (isMatch(listA, listB, first, second) ? GREEN
"The lists match" :
    RED "The lists don't match") <<
    RESET << '\n' << '\n';

    freeMemory(listA);
    freeMemory(listB);
}

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

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

```