МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Алгоритмы и структуры данных»

Тема: Рекурсия.

Студент гр. 7383 _____ Бергалиев М.А.

Преподаватель _____ Размочева Н.В.

Санкт-Петербург

2018

ОГЛАВЛЕНИЕ

1. ЦЕЛЬ РАБОТЫ

Цель работы: познакомиться с основными понятиями и приемами рекурсивного программирования, получить навыки программирования рекурсивных процедур и функций на языке программирования C++.

Формулировка задачи: Задано конечное множество имен жителей некоторого города, причем для каждого из жителей перечислены имена его детей. Жители X и Y называются родственниками, если (а) либо X – ребенок Y, (б) либо Y – ребенок X, (в) либо существует некоторый Z, такой, что X является родственником Z, а Z является родственником Y. Перечислить все пары жителей города, которые являются родственниками.

2. РЕАЛИЗАЦИЯ ЗАДАЧИ

В функции main выводится приглашение выбрать способ ввода входных данных либо выйти из программы. В случае выбора файла, программа предлагает ввести имя файла. Создается объект типа std::istream, который передается функции input_parser. В случае ввода информации с консоли, то функции input_parser передается объект std::cin. После успешной обработки входных данных вызывается функция couples_relatives, которая находит ответ на поставленную задачу. Далее ответ выводится на экран, а также количество пар в ответе, и процесс вновь повторяется с приглашения для выбора способа ввода данных. Если в ходе процесса были введены некорректные данные, то выводится сообщение о неправильных данных и процесс продолжается сначала.

Переменные, используемые в функции main:

- command — строка, содержащая номер команды, отвечающий за выбор способа ввода данных или выхода из программы.
- file — буффер потока файла, участвующий в создании объекта fin типа istream, передаваемого в функцию input_parser.
- filename — имя файла, из которого берутся входные данные.
- input — словарь, содержащий родителей и их детей, результат обработки входных данных.
- result — множество пар родственников, являющееся ответом на задачу.

Функция input_parser считывает построчно из входного потока исходные данные. Сперва считывается количество людей, у которых есть дети. В каждой последующей строчке считывается родитель и его дети в формате «*родитель*: *дети*».

Переменные, используемые в функции input_parser:

- input — хранит считанную строку.
- parent — хранит имя родителя.

- child — хранит имя ребенка.
- children — множество детей текущего родителя.
- n — общее число родителей.
- result — словарь родителей и их детей.

Функция couples_relatives принимает словарь родителей и их детей. В первую очередь создается множество всех людей, которые есть в переданном словаре. Далее для каждого вызывается функция вспомогательная функция find_relatives, которой передаются также словарь, переменная, в которую будет записываться результат, а также переменная, хранящая список промежуточных родственников, через которых уже был произведен поиск. Также выводится вспомогательная информация о том, для какого человека вызывается функция. Возвращается множество пар всех родственников.

Переменные, используемые в функции couples_relatives:

- result — результат, хранящий множество пар всех родственников.
- persons — множество всех людей из словаря.
- viewed — множество промежуточных родственников, которые были использованы для поиска родственников для данного человека.

Вспомогательная функция find_relatives принимает словарь родителей и их детей, человека, для которого идет поиск родственников, промежуточный родственник, через которого идет поиск, множество, в котором хранится результат, а также множество всех использованных промежуточных родственников. Если в множестве, хранящем результат, нет пары человека и промежуточного родственника, то в результат добавляется эта пара. Далее для каждого ребенка промежуточного родственника, которого нет в списке использованных промежуточных родственников, происходит рекурсивный вызов, в котором в качестве промежуточного родственника передается ребенок. Тоже самое происходит для каждого родителя промежуточного родственника. В процессе выводится информация о том, какой промежуточный родственник используется и какая пара была добавлена, в

зависимости от глубины рекурсии.

Переменные, используемые в функции find_relatives:

- tabs — переменная, содержащая отступ для данного уровня рекурсии.
- children — множество детей человека, среди которых ищется промежуточный родственник.
- parent — человек, являющийся родителем. Если в его множестве детей есть промежуточный родственник, то для родителя происходит рекурсивный вызов.

3. ТЕСТИРОВАНИЕ

Программа была собрана в компиляторе G++ с ключом -std=c++14 в OS Linux Ubuntu 16.04 LTS.

В ходе тестирования была найдена ошибка, из-за которой ответ на задачу был некорректен. Изначально, программа изменяла словарь родителей и детей в словарь родственников, однако при изменении словаря происходила инвалидация итераторов словаря. Поэтому было принято решение не изменять изначальный словарь, а сразу строить множество пар родственников.

Некорректный случай представлен в табл. 1, в котором пропущено двоеточие. В данном случае строка не несет никакой информации и не влияет на результат.

Таблица 1 — Некорректный случай с пропущенным двоеточием.

Входные данные	Результат
2 A B C D: E B	B D B E D E

Корректные тестовые случаи представлены в приложении А.

4. ВЫВОД

В ходе работы были получены навыки рекурсивного программирования на языке C++. Поскольку структуру родственных отношений можно считать деревом, которое в свою очередь является рекурсивной структурой, рекурсивный подход является простым и удобным способом поиска решения. Глубина рекурсии при поиске решения не превышает количества людей в дереве родственных отношений.

ПРИЛОЖЕНИЕ А.

ТЕСТОВЫЕ СЛУЧАИ

Таблица 2 — Корректные тестовые случаи

Входные данные	Результат
3 A: B C D: V G H: C	A B A C A H B C B H C H D G D V G V
4 A: B C C: D E F: E G: F H	A B A C A D A E A F A G A H B C B D B E B F B G B H C D C E C F C G C H D E D F D G D H E F E G

Продолжение таблицы

	E H F G F H G H
3 A: B C D E: F G H: I K	A B A C A D B C B D C D E F E G F G H I H K I K
5 A: B B: C D C: A D D: H E L: W Q R	A B A C A D A E A H B C B D B E B H C D C E C H D E D H E H L Q L R L W Q R Q W R W

ПРИЛОЖЕНИЕ Б.

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <fstream>
#include <unordered_map>
#include <set>

auto input_parser(std::istream& inpstream){
    std::string input;
    std::string parent;
    std::string child;
    std::set<std::string> children;
    std::unordered_map<std::string,
std::set<std::string>> result;
    int n = 0;
    getline(inpstream, input);
    n = std::stoi(input);
    for(int i = 0; i < n; ++i){
    children.clear();
    parent.clear();
    child.clear();
        getline(inpstream, input);
    std::string::iterator s;
        for(s = input.begin(); s != input.end() && *s !
= ':';  ++s)
        if(*s != ' ')
        parent.push_back(*s);
    if(*s == ':')
        ++s;
    for(; s != input.end(); ++s){
        if(*s == ' '){
        if(child != "")
            children.insert(child);
        child.clear();
        }else
        child.push_back(*s);
    }
```

```
    if(child != std::string())
        children.insert(child);
    result.insert(std::make_pair(parent, children));
    }
    return result;
}


int main(){
    std::string command;
    std::filebuf file;
    std::string filename;
    std::unordered_map<std::string,
std::set<std::string>> input;
    while(true){
    std::cout << "Enter 0 to read input from consol or
1 to read from file or 2 to exit: ";
    getline(std::cin, command);
    try{
        if(std::stoi(command) == 2)
        break;
    }
    catch(std::exception &e){
        std::cout << "Invalid command, try again" <<
std::endl;
        continue;
    }
    switch(std::stoi(command)){
        case 0:{
        try{
            input = input_parser(std::cin);
        }
        catch(std::exception& e){
            std::cout << "Invalid input" << std::endl;
            continue;
        }
        break;
        }
```

```
        case 1:{
        std::cout << "Enter file name: ";
        getline(std::cin, filename);
        if(file.open(filename, std::ios::in)){
            std::istream fin(&file);
            try{
            input = input_parser(fin);
            }
            catch(std::exception& e){
            std::cout << "Invalid file input" <<
std::endl;
            continue;
            }
            file.close();
        }
        else{
            std::cout << "Incorrect filename" <<
std::endl;
            file.close();
            continue;
        }
        break;
        }
        default:{
        std::cout << "Incorrect command, try again" <<
std::endl;
        continue;
        }
    }
    auto result = couple_relatives(input);
    for(auto i : result)
        std::cout << std::get<0>(i) << " " <<
std::get<1>(i) << std::endl;
    std::cout << result.size() << std::endl;
    }
    return 0;
}
```

```
static void
find_relatives(std::unordered_map<std::string,
std::set<std::string>> &relatives, std::string const
&person, std::string const &sub_relative,
std::set<std::pair<std::string, std::string>> &result,
std::set<std::string> &viewed){
    static std::string tabs;
    if(person != sub_relative &&
result.find(std::make_pair(sub_relative, person)) ==
result.end()){
    result.insert(std::make_pair(person,
sub_relative));
    std::cout << tabs << "Insert " << sub_relative <<
std::endl;
    }
    if(person != sub_relative)
    std::cout << tabs << "Sub relative " <<
sub_relative << "{" << std::endl;
    tabs.push_back('\t');
    viewed.insert(sub_relative);
    for(auto relative : relatives[sub_relative]){
    if(person != relative && viewed.find(relative) ==
viewed.end()){
        find_relatives(relatives, person, relative,
result, viewed);
    }
    }
    for(auto relation : relatives){
    auto children = std::get<1>(relation);
    auto parent = std::get<0>(relation);
        if(children.find(sub_relative) !=
children.end() && viewed.find(parent) == viewed.end()){
        find_relatives(relatives, person, parent,
result, viewed);
    }
    }
    tabs.pop_back();
    if(person != sub_relative)
```

```
    std::cout << tabs << "}" << std::endl;
}

auto couple_relatives(std::unordered_map<std::string,
std::set<std::string>> children){
    std::set<std::pair<std::string, std::string>>
result;
    std::set<std::string> persons;
    for(auto relation : children){
        persons.insert(std::get<1>(relation).begin(),
std::get<1>(relation).end());
    persons.insert(std::get<0>(relation));
    }
    for(auto person : persons){
    std::set<std::string> viewed;
    std::cout << "Person " << person << "{" <<
std::endl;
    find_relatives(children, person, person, result,
viewed);
    std::cout << "}" << std::endl;
    }
    return result;
}
```