**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «Стеки и очереди»**

Студент гр. 7382 Лящевская А. П.

Преподаватель Фирсов М. А.

Санкт-Петербург

2018

**Задание.**

В заданиях 1 – 3 следует использовать очередь и операции над ней; при этом

очередь может быть реализована как на базе вектора, так и в связанной памяти

(ссылочная реализация).

**Вариант 3 - В.**

Рассматриваются следующие типы данных:

***type*** *имя* =(*Анна*, ..., *Яков*);

*дети* =***array***[*имя, имя*]***of*** *Boolean*;

*потомки* =***file of*** *имя*.

Задан массив *Д* типа дети (*Д* [*x*, *y*] = *true*, если человек по имени *y* является  
ребенком человека по имени *x*). Для введенного пользователем имени *И* записать  
в файл *П* типа *потомки* имена всех потомков человека с именем *И* в следующем  
порядке: сначала имена всех его детей, затем всех его внуков, затем всех

правнуков и т. д.

**Пояснение задания.**

На вход программе подаётся файл с последовательностью имен и файл с таблицей смежности(родства). Итогом завершения программы должен быть файл, в котором будут записаны все потомки указанного для этого действия имени *И* (в тестировании для первого имени из файла имен): сначала имена всех детей *И*, затем имена всех внуков *И*, правнуков и т.д.

**Описание алгоритма.**

На вход в качестве аргументов *argv*[*1*]и *argv*[*2*] программа получает файлы для считывания имен и таблицы родства соответственно. Данные из этих файлов записываются в массивы, назовем их *names* и *concts*. Так как от того, сколько будет записано имен в *names* зависит считывание для *concts,* то в случае нехватки данных программа выведет об этом сообщение и остановит работу.

Следующим шагом будет обработка полученной информации. Для этого нам требуется получить на вход имя *name*, для которого будут применены нижеописанные действия. (В режиме тестов этот параметр заполняется автоматически первым именем из списка имен *names*.) Также создадим объект очереди и указатель на него *Queue*. Посылаем в функцию *find\_child* все собранные данные. Только вместо *name* отправим индекс этого имени. Получим его через специальную функцию *index\_person*. Специально для случая, когда введенное *name* не соответствует ни одному значению из *names* определены действия экстренного выхода из программы и вывода сообщения о соответствующей ошибке.

Отдельно стоит пояснить как будут храниться элементы в очереди. Для каждого из элементов определено свое колено (степень родства с *names*). Все элементы будут размещены в очереди по возрастанию соответствующих им колен. Но сам элемент не хранит информации о его колене. Эта информация хранится в специальном массиве *i\_remove,* в котором в значении по индексу *i* хранится количество элементов с коленом номера *i*. Так, для каждого из элементов очереди можно узнать какому из колен он соответствует.

Оказавшись в функции *find\_child,* занесем первый элемент в список с помощью метода *push* для очереди с учетом того, что колено (степень родства с *names*) в этот момент 0, этот элемент всегда будет на первом месте. Далее воспользуемся данными из массива *concts.* Рассмотрим столбик номера индекса искомого *name.* Если находится ненулевая ячейка строки *i* и выполняется ряд условий (1-е: по обратному адресу найденной ячейки нельзя сказать, что родство замыкается в одном колене; 2-е: в очереди не хранится элемент с найденным индексом ребенка с коленом ниже, чем колено текущего имени), то поиск потомства продолжается рекурсивно этой же функцией для найденного ребенка, только с параметрами колена на 1 больше. Происходит обязательный пуш как и для первого элемента. Только теперь, если колено найденного элемента не является наибольшим, элемент запишется прямо перед первым элементом следующего колена.

Записав все элементы, перейдем к «печати». В нашем случае под «печатью» подразумевается запись в файл *Result.txt*. Для этого в функцию *print\_queue* посылается искомое *name, names* (массив доступных имен)и переменная количества имен. И так как поп очереди всегда выводит первый элемент очереди, то по количеству элементов каждого из колен выведем элементы с распределением по коленам.

Программа завершает свое действие после освобождения памяти выделенной динамически и закрытия файла на запись.

**Описание функций**

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Выходной параметр** | **Входные данные** | **Описание** |
| **index\_person** | int | **String –** строка содержащая искомое имя;  **String \* –** массив доступных имен(строк);  **Int –** количество имен; | Функция определяющая индекс по имени. |
| **find\_child** | - | **Int –** текущее колено;  **Int –** индекс имени для поиска детей;  **Int** \*\***–** дв. массив смежности(родства);  **Queue** \* **–** указатель на очередь, куда записываются дети;  **Int –** количество имен; | Функция нахождения детей по индексу имени. |
| **print\_queue** | - | **Int –** текущее колено;  **Queue \* –** указатель на очередь, откуда берутся индексы детей;  **String \* –** указатель на массив имен для вывода их по индексу из очереди;  **Oftream &–** файл для записи результата; | Функция печати очереди. |

**Описание очереди**

Специально для очереди определены следующие глобальные переменные:

**#define START\_SIZE 10**

**#define COUNT\_REMOVE 10**

Очередь была реализован на основе класса.

У класса *Queue* были определены следующие **объекты**:

|  |  |  |
| --- | --- | --- |
| **Объект** | **Тип** | **Описание** |
| int \*data; | Private | Указатель под массив имен. |
| int \*i\_remove; | Private | Указатель под массив количества элементов колена по индексу. |
| int size; | Private | Текущий размер очереди. |

Также были определены **методы** класса *Queue*:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Метод** | **Тип** | **Выходные данные** | **Входные данные** | **Описание** |
| Queue() | Public | - | - | Конструктор очереди. Выделяет память под *data*, *i\_remove* и устанавливает *size* в 0. |
| ~Queue() | Public | - | - | Деструктор очереди. |
| Push | Public | - | **Int** val – значение для записи;  **Int** remove – колено для значения. | Пуш для индексов имен учитывая соответствующее им колено. |
| Pop | Public | **Int** | - | Поп первого элемента со смещением очереди в сторону удаляемого элемента. |
| Count\_remove | Public | **Int** | **Int** remove – колено для значения. | Возвращает количество элементов с номером колена *remove.* |
| isEmpty | Public | **Bool** | - | Возвращает *true* в случае непустого очереди и *false* в обратном. |
| Is\_in\_queue | Public | **Bool** | **Int** el – элемент для проверки;  **Int** remove – колено элемента. | Возвращает *true* если элемент встретился в очереди с коленом меньшим, чем *remove*, и *false* если нет. |
| Resize | Private | - | - | Увеличивает размер массива данных, если это требуется. |

**Тестирование.**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер  теста | Файл Names | Файл Connections: | Результат: |
| 1 | Маша Глаша Валя | 0 0 1  1 0 0  0 1 0 | Ошибка: замкнутая цепь родства |
| 2 | Маша Глаша Валя Паша Зита Гита Рита | 0 0 0 0 0 0 0  1 0 0 0 0 0 0  0 1 0 0 0 0 0  0 0 0 0 1 1 0  0 0 1 0 0 0 0  0 0 0 0 0 0 1  0 1 0 0 0 0 0 | Потомки для Маша  Потомки 1-го поколения: Глаша  Потомки 2-го поколения: Рита Валя  Потомки 3-го поколения: Гита Зита  Потомки 4-го поколения: Паша |
| 3 |  |  | Ошибка: Имени нет в списке |
| 4 | Влад Сергей Ростивлав Таня Мира | 0 0 0 0 0  0 0 1 0 0  1 0 0 1 0  0 0 0 0 0  0 1 0 | Ошибка: недостаточно данных для таблицы родства |
| 5 | Рита Петя Дима | 0 1 0  1 0 1  0 1 0 | Ошибка: ребенок не может быть отцом его родителя |
| 6 | Миша Лена Катя | 0 0 0  0 0 0  0 0 0 | Потомки для Миша |

**Вывод.**

В процессе выполнения лабораторной работы были продуманы, созданы и реализованы на практике алгоритмы и методы работы с очередью на основе вектора, а также по работе с ней. Также была решена задача нахождения всех потомков заданного пользователем имени исходя из данных имен и сводной таблицы родства.

**Приложение 1. Код программы.**

* [**Run\_test.sh**](https://github.com/makometr/AiSD/pull/37/files#diff-240c2792fd92b595432e18629f6e16b6)

#!/bin/bash

g++ ./Source/Lab3.cpp -o Lab3

echo -e '\_\_\_\_\_\_\_\nTest 1:\n'

./Lab3 ./Tests/Names1.txt ./Tests/Connections1.txt

echo -e '\_\_\_\_\_\_\_\nResult 1:\n'

cat Result.txt

echo -e '\_\_\_\_\_\_\_\nTest 2:\n'

./Lab3 ./Tests/Names2.txt ./Tests/Connections2.txt

echo -e '\_\_\_\_\_\_\_\nResult 2:\n'

cat Result.txt

echo -e '\_\_\_\_\_\_\_\nTest 3:\n'

./Lab3 ./Tests/Names3.txt ./Tests/Connections3.txt

echo -e '\_\_\_\_\_\_\_\nResult 3:\n'

cat Result.txt

echo -e '\_\_\_\_\_\_\_\nTest 4:\n'

./Lab3 ./Tests/Names4.txt ./Tests/Connections4.txt

echo -e '\_\_\_\_\_\_\_\nResult 4:\n'

cat Result.txt

echo -e '\_\_\_\_\_\_\_\nTest 5:\n'

./Lab3 ./Tests/Names5.txt ./Tests/Connections5.txt

echo -e '\_\_\_\_\_\_\_\nResult 5:\n'

cat Result.txt

echo -e '\_\_\_\_\_\_\_\nTest 6:\n'

./Lab3 ./Tests/Names6.txt ./Tests/Connections6.txt

echo -e '\_\_\_\_\_\_\_\nResult 6:\n'

cat Result.txt

* [**Queue.c**](https://github.com/makometr/AiSD/pull/37/files#diff-240c2792fd92b595432e18629f6e16b6)**pp**

#include <iostream>

#define START\_SIZE 10

#define COUNT\_REMOVE 10

using namespace std;

class Queue{

public:

Queue() { //Конструктор выделяет память под данные.

size = 0;

data = new int[START\_SIZE];

i\_remove = new int[COUNT\_REMOVE];

for(int i = 0; i < COUNT\_REMOVE; i++)

i\_remove[i] = 0;

}

~Queue() {} //Деструктор пуст, так как очередь подается в функции.

void push(int val, int remove) { //Пуш элемента с учетом колена.

if(is\_in\_queue(val, remove + 1)) //В случае если в этом колене уже встречался этот потомок, не записывать его

return;

resize(); //Увеливаем память, если нужно.

int i\_el = 0;

for(int i = 0; i < remove; i++) //Считаем количество элементов до колена remove

i\_el += i\_remove[i];

if(i\_el != size) { //Если элемент не последний,

for(int i = size; i > i\_el; i--)

data[i] = data[i-1]; //сместим все элементы на одну позицию,

} //выделенную под новый элемент.

data[i\_el] = val; //Запишем новый элемент

i\_remove[remove]++; //Для колена remove увеличим количество его элементов

size++;

}

int pop() { //Поп первого элемента со смещением всех элементов влево

int poped = data[0];

for(int i = 0; i < size - 1; i++)

data[i] = data[i+1];

size--;

return poped;

}

int count\_remove(int remove) {return i\_remove[remove];} //Возвращает количество элементов колена

int isEmpty() {return size;} //Проверяет очередь на пустоту

int is\_in\_queue(int el, int remove) { //Проверяет встречался ли элемент в предыдущих коленах

int i\_el = 0;

for(int i = 0; i < remove; i++)

i\_el += i\_remove[i];

for(int i = 0; i < i\_el; i++)

if(data[i] == el)

return 1;

return 0;

}

private:

int \*data; //Указатель под массив имен

int \*i\_remove; //Указатель под массив количества элементов колена по индексу

int size; //Размер очереди

void resize() { //Динамическое увеличение массива data

if(size % START\_SIZE == 0 && size) {

cout << "+10 to data"<< endl;

int \*pTmp = new int[size + 10];

for(int i = 0; i < size; i++)

pTmp[i] = data[i];

delete[] data;

data = pTmp;

}

}

};

* [**Lab3.c**](https://github.com/makometr/AiSD/pull/37/files#diff-240c2792fd92b595432e18629f6e16b6)**pp**

#include <iostream>

#include <fstream>

#include <string>

#include "Queue.cpp"

//#define USER

#define TEST

using namespace std;

int index\_person(string, string \*, int); //Функция нахождения индекса имени

void find\_child(int, int, int \*\*, Queue\*, int); //Функция нахождения детей и записи в очередь

void print\_queue(int, Queue\*, string \*, ofstream &); //Функция печати поколений из очереди

int main(int argc, char \* argv[]) {

ifstream file (argv[1]); //В первый раз открываем файл с именами

string tmp; //для нахождения количества имен persons

int persons;

for(persons = 0;; persons++) {

file >> tmp;

if(file.eof()) break;

}

file.close(); //end.

ifstream file1 (argv[1]); //Во второй раз открываем тот же файл

string \*names = new string[persons]; //для записи имен в массив строк names.

cout << "Индекс - имя:" << endl;

for(int i = 0; i < persons; i++) {

file1 >> names[i];

cout << i << ' ' << names[i] << endl;

}

file1.close(); //end.

ifstream file2 (argv[2]); //Открываем файл с матрицей родства

int \*\*concts = new int \*[persons]; //и записываем ее в двумерный массив concts.

for(int i = 0 ; i < persons; i++)

concts[i] = new int[persons];

cout << endl << "Таблица смежности:" << endl << "\x1b[4m"<< " |";

for(int i = 0; i < persons; i++)

cout << " " << i;

cout << "\x1b[0m" << endl;

for(int i = 0; i < persons; i++) {

cout << i << "| " ;

for(int j = 0; j < persons; j++) {

file2 >> concts[i][j];

if(file2.eof() && j != persons) { //Обработка ошибки нехватки данных из откртого файла

cout << endl << "Ошибка: недостаточно данных для таблицы родства" << endl;

exit(1);

}

cout << concts[i][j] << ' ';

}

cout << endl;

}

file2.close(); //end.

cout << endl;

string name;

#ifdef USER

cout << "Введите имя: "; //Запрос имени name для предоставления потомков.

cin >> name;

#endif

#ifdef TEST

name = names[0];

#endif

Queue \*queue = new queue;

find\_child(0, index\_person(name, names, persons), concts, queue, persons); //Найдем и запишем рекурсивно детей для nameю

ofstream fout;

fout.open("Result.txt");

print\_queue(0, queue, names, fout); //Выведем список потомков.

fout.close();

for(int i = 0; i < persons; i++) //Освобождение памяти под concts.

free(concts[i]);

free(concts);

return 0;

}

int index\_person(string name, string \*names, int persons) { //Функция определяющая индекс имени.

for(int i = 0; i < persons; i++)

if(name == names[i]) return i;

cout << "Ошибка: Имени нет в списке" << endl; //Вывод ошибки в случае неправильного имени.

exit(1);

}

void find\_child(int remove,int i\_name, int \*\*concts, Queue \*queue, int persons) { //Функция нахождения детей по индексу имени.

queue->push(i\_name, remove); //Пуш в очередь индекса найденного имени

for(int i = 0; i < persons; i++){ //с учетом колена потомка.

if(concts[i][i\_name]) { //Для найденного имени

if(concts[i\_name][i]) {

cout << "Ошибка: ребенок не может быть отцом его родителя" << endl;

exit(1);

}

if(queue->is\_in\_queue(i, remove)) {

cout << "Ошибка: замкнутая цепь родства" << endl;

exit(1);

}

find\_child(remove + 1, i, concts, queue, persons); //поиск детей для детей name.

}

}

}

void print\_queue(int remove, Queue \*queue, string \*names, ofstream &fout) {//Функция печати очереди.

if(remove) fout << "Потомки "<< remove << "-го поколения: ";

else fout << "Потомки для ";

for(int i = 0; i < queue->count\_remove(remove); i++) //По количеству потомков текущего колена remove

fout << names[queue->pop()] << ' '; //вывод имени по индексу полученному из очереди.

fout << endl;

if(queue->isEmpty()) //Если очередь не пуст

print\_queue(remove+1, queue, names, fout); //идем печатать следующее колено.

}