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

ОТЧЕТ

по лабораторной работе № 3

по дисциплине «Алгоритмы и структуры данных»

Тема: Линейные структуры данных: стек, очередь и дек

Студент гр. 7383 _____ Васильев А.И.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

Цель работы

Ознакомиться с часто используемыми на практике линейными структурами данных, обеспечивающими доступ к элементам последовательности только через её начало и конец, и способами реализации этих структур, освоить на практике использование очереди для решения практических задач.

Постановка задачи

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.

Уточнение задачи

На вход программе подается файл с последовательностью имен и файл с таблицей смежности (родства). Итогом завершения программ долджен быть файл, в котором будут записаны все потомки указанного для этого действия имени *И*: сначала имена всех детей *И*, затем имена всех внуков *И*, правнуков и так далее.

Описание алгоритма

На вход программа получает файлы для считывания имен (*Names.txt*) и таблицы родства (*Connections.txt*). Данные из этих файлов записываются в массивы `string names[]` и `int concts[][]` соответственно. 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 завершит свою работу.

Следующим шагом будет обработка полученной информации. Для этого требуется получить на вход имя `name`, для которого будут применены нижеописанные действия. Так же создается объект очереди и указатель на него `Queue`. Далее идет передача функции `find_child` всех собранных данных.

Только вместо `name` передается индекс этого имени, который получается через функцию `index_person`. Специально для случая, когда введенное `name` не соответствует ни одному значению из `names[]` определены действия выхода из программы и вывода сообщения о соответствующей ошибке.

Хранение элементов в очереди обусловлено следующими соображениями: для каждого из элементов определено свое колено (степень родства с `name`). 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 `name` в список с помощью метода `push` для очереди с учетом того, что колено в этот момент равняется 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 (во-первых: по обратному адресу найденной ячейки нельзя сказать, что родство замыкается в одном колене; во-вторых: 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 это было для первого элемента. Только теперь, если колено найденного элемента не является наибольшим, элемент запишется прямо перед первым элементом следующего колена.

Записав все элементы в очередь, программа переходит к выводу, в нашем случае к записи результата работы в файл *Result.txt*. Для этого в функцию `print_queue` передается массив имен `names[]`, очередь, которую требуется вывести, и переменная `remove`, которая содержит в себе количество имен. И так как «поп» очереди всегда выводит первый элемент очереди, то по количеству элементов каждого из колен выводятся элементы с распределением по коленам.

Программа завершает свою работу после освобождения динамически выделенной памяти и закрытия файла с результатами работы программы.

Описание функций

Описание функций, использованных в программе представлено в табл. 1.

Таблица 1 — Описание функций

Название	Тип функции	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	void	**int** — текущее колено; **int** — индекс имени для поиска детей; **int**** - двумерный массив смежности(родства); **queue*** - указатель на очередь, куда записываются потомки; **int** — количество имен.	Функция нахождения детей по индексу имени.
print_queue	void	**int** — текущее колено; **queue*** - указатель на очередь, которая выводится в файл; **string*** - указатель на массив имен для вывода их по индексу из очереди; **Ofstream&** - 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 реализована на основе класса `Queue`, объекты которого описаны в табл. 2. Методы данного класса описаны в табл. 3.

Таблица 2 — Описание объектов класса `Queue`

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.

Таблица 3 — 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.	Добавление в очередь для индексов имен, учитывая соответствующее им колено
Pop	Public	int	-	Получение первого элемента из очереди, удаление этого элемента из очереди со смещением очереди в сторону удаляемого элемента.

Окончание таблицы 3

count_remove	Public	int	int remove — колено для значения	Возвращает количество элементов с номером колена *remove*
isEmpty	Public	bool	-	Возвращает *true* в случае непустой очереди и *false* в обратном случае
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* в обратном случае.
Resize	Private	-	-	Увеличивает размер массива данных, если это требуется.

Тестирование

Было создано несколько тестов для проверки работы программы. Помимо тестов, демонстрирующих работу алгоритма, были написаны тесты, содержащие некорректные входные данные, для демонстрации вывода сообщений об ошибках введенных данных. Тестовые случаи представлены в приложении А.

Исходный код программы представлен в приложении Б.

Вывод

в процессе выполнения лабораторной работы были продуманы, созданы и реализованы на практике алгоритмы и методы работы с очередью на основе вектора. Так же была решена задача нахождения всех потомков заданного пользователем имени исходя из данных имен и сводной таблицы родства.

ПРИЛОЖЕНИЕ А

Тестирование

№ теста	Файл *Names.txt*	Файл *Connections.txt*	Результат (файл *Result.txt*)
1	Мария Иван Валентина	0 0 1 1 0 0 0 1 0	Ошибка: замкнутая цепь родства
2	Мария Иван Валентина Павел Зина Галина 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 Мария Потомки 1-го поколения: Иван Потомки 2-го поколения: Рита Валентина Потомки 3-го поколения: Галина Зина Потомки 4-го поколения: Павел
3	Влад Сергей Ростислав Татьяна Игорь	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
4	Рита Петр Дмитрий	0 1 0 1 0 1 0 1 0	Ошибка: ребенок не может быть отцом его родителя
5			Ошибка: имени нет в списке
6	Миша Лена Катя	0 0 0 0 0 0 0 0 0	Потомки для Миша

ПРИЛОЖЕНИЕ Б

Код программы

Файл main.cpp

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

#include "Queue.cpp"

using namespace std;

int index_of_name(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()
{

    ofstream fout;
//Открываем файл на запись результата
    fout.open("Result.txt");

    ifstream file ("Names.txt");
//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 ("Names.txt");
//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 ("Connections.txt");
//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[m" << 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 открытого 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;

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

    Queue *queue = new Queue;

    find_child(0, index_of_name(name, names, persons), concts, queue, persons);
//Найдем и запишем рекурсивно детей для namею

    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_of_name(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 пуста
        print_queue(remove+1, queue, names, fout);
//идем печатать следующее колено.
}
```

Файл Queue.cpp

```
#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  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();
//Увеличиваем 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)                        //Возвращает количество
элементов колена
    {
        return i_remove[remove];
    }

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

    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;
        }
    }
};
```