

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

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Линейные структуры данных: стек, очередь, дек

Студент гр. 8304

Алтухов А.Д.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2019

Цель работы.

Ознакомиться с линейными структурами данных, обеспечивающими доступ к данным только через начало и/или конец структуры, способами их реализации.

Задание.

Вариант 3-д.

Рассматриваются следующие типы данных:

```
type имя = (Анна, ..., Яков);  
дети = array [имя, имя] of Boolean;  
потомки = file of имя.
```

Задан массив D типа *дети* ($D[x, y] = true$, если человек по имени y является ребенком человека по имени x). Для введенного пользователем имени I записать в файл P типа *потомки* имена всех потомков человека с именем I в следующем порядке: сначала – имена всех его детей, затем – всех его внуков, затем – всех правнуков и т. д.

Описание алгоритма.

Во время считывания данных целевое имя I записывается в очередь. Эта очередь передается в обрабатывающую функцию `findDescendants`. Функция работает следующим образом: для каждого члена очереди происходит итеративный перебор массива D , все найденные дети I записываются в новую очередь. Если новая очередь не пуста, происходит рекурсивный вызов `findDescendants` с передачей новой очереди в качестве аргумента. Последовательность повторяется, пока новая очередь не окажется пуста.

Существует более простая вариация алгоритма, которая добавляет найденных детей в уже существующую очередь и перебор происходит пока очередь не окажется пустой. К сожалению, для такого алгоритма невозможно организовать наглядный вывод требуемой информации.

Основные функции и структуры.

1. `struct Elem` представляет элемент очереди. Содержит следующие поля:
`std::string value` — значение, хранимое в очереди.

`Elem* next` — ссылка на следующий элемент, если таковой имеется.

У структуры есть примитивный конструктор, инициализирующий поле `next` значением `nullptr`. Деструктор уничтожает все элементы, на которые ссылается `next`.

2. `class Queue` обеспечивает необходимые методы для взаимодействия с очередью. Содержит следующие приватные поля:

`Elem* head`

`Elem* last`

Эти поля указывают на начальный и конечный элемент очереди соответственно.

Класс содержит базовые методы, необходимые для взаимодействия с очередью.

`Queue() : head(nullptr), last(nullptr) {}` — примитивный конструктор, инициализирующий поля.

`~Queue()` — деструктор вызывает деструктор структуры `Elem` для первого элемента.

`bool isEmpty() const` — проверяет, пуста ли очередь.

`void push(std::string value)` — добавляет в конец очереди новый элемент.

`std::string pop()` — удаляет из очереди первый элемент и возвращает его.

`std::string top() const` — возвращает первый элемент очереди.

`void print() const` — печатает все содержимое очереди.

3. `void readData(Queue* queue, ChildrenArr& children, std::ifstream& inputF)` — считывает данные для массива `children` из указанного файла.

`void findDescendants(Queue* queue, ChildrenArr& children, std::ofstream& outputF, int genNumber)` — работает так, как указано в пункте «Описание алгоритма».

Тестирование.

Некорректные данные:

№	Исходные данные	Ответ программы	Ожидаемый ответ
1	Darya Vanya	Не указан предок, чьих потомков надо найти.	Сообщение об ошибке

Корректные данные:

№	Исходные данные	Ответ программы	Ожидаемый ответ
2	Darya Vanya Vanya	Потомков не найдено.	Потомков не найдено.
3	Darya Vanya Darya	Поколение 1: Vanya;	Поколение 1: Vanya;
4	Darya Vanya Darya Serega Jenya Darya Darya	Поколение 1: Serega; Vanya;	Поколение 1: Serega; Vanya;
5	Darya Vanya Darya Serega Jenya Darya Serega Erjan Vanya Greka Erjan Alex Erjan Katya Darya Lesya Greka Reka Serega	Поколение 1: Erjan; Поколение 2: Alex; Katya;	Поколение 1: Erjan; Поколение 2: Alex; Katya;
6	Darya Vanya	Поколение 1: Serega;	Поколение 1: Serega;

	Darya Serega Jenya Darya Serega Erjan Vanya Greka Erjan Alex Erjan Katya Greka Reka Darya	Vanya; Поколение 2: Erjan; Greka; Поколение 3: Alex; Katya; Reka;	Vanya; Поколение 2: Erjan; Greka; Поколение 3: Alex; Katya; Reka;
--	--	--	--

Выводы.

В ходе выполнения лабораторной работы была исследована такая структура данных, как очередь, изучены методы работы с ней. Был реализован алгоритм поиска потомков по заданным связям.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД.

Файл queue.h

```
#ifndef QUEUE
#define QUEUE

#include <string>

struct Elem;

struct Elem {
    std::string value;
    Elem* next;

    Elem() : next(nullptr) {}
    ~Elem();
    void print() const;
};

class Queue {
private:
    Elem* head;
    Elem* last;

public:
    Queue() : head(nullptr), last(nullptr) {}
    ~Queue();
    bool isEmpty() const;
    void push(std::string value);
    std::string pop();
    std::string top() const;
    void print() const;
};

#endif
```

Файл queue.cpp

```
#include "queue.h"

#include <iostream>

Elem::~Elem() {
    if (this->next)
        delete this->next;
}

void Elem::print() const {
```

```

        std::cout << this->value << "; ";
        if (this->next != nullptr)
            this->next->print();
    }

Queue::~Queue() {

    if (!this->isEmpty())
        delete this->head;
}

bool Queue::isEmpty() const {
    if (this->head == nullptr)
        return true;
    return false;
}

void Queue::push(std::string value) {

    if (this->isEmpty()) {
        this->last = new Elem;
        this->last->value = value;
        this->head = this->last;
    }
    else {
        this->last->next = new Elem;
        this->last = this->last->next;
        this->last->value = value;
    }
}

std::string Queue::pop() {

    if (this->isEmpty()) {
        std::cerr << "Очередь пуста!\n";
        return "";
    }
    else {
        std::string returnVal = this->head->value;
        Elem* newHead = this->head->next;
        this->head->next = nullptr;
        delete this->head;

        this->head = newHead;
        if (this->head == nullptr) {
            this->last = nullptr;
        }
        return returnVal;
    }
}

```

```

std::string Queue::top() const {

    if (this->isEmpty()) {
        std::cerr << "Очередь пуста!\n";
        return "";
    }
    return this->head->value;
}

void Queue::print() const {

    if (this->isEmpty()) {
        std::cout << "()\n";
        return;
    }
    std::cout << "( ";
    this->head->print();
    std::cout << ")\n";
}

```

Файл main.cpp

```

#include <iostream>
#include <string>
#include <map>
#include <fstream>
#include <locale>

#include "queue.h"

typedef std::map<std::string, std::map<std::string, bool>> ChildrenArr;
//ассоциативный двумерный массив arr[string][string] = bool

void readData(Queue* queue, ChildrenArr& children, std::ifstream&
inputF);
void findDescendants(Queue* queue, ChildrenArr& children, std::ofstream&
outputF, int genNumber);

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

    setlocale(LC_ALL, "RU");
    std::ifstream inputF;
    if (argc > 1)
        inputF.open(argv[1]);
    else
        inputF.open("input.txt");

    if (!inputF.is_open()) {
        std::cerr << "Невозможно открыть файл со входными данными";
        return 0;
    }
}

```



```

    }

    std::ofstream outputF("output.txt");
    if (!outputF.is_open()) {
        std::cerr << "Невозможно открыть файл вывода";
        return 0;
    }

    Queue* queue = new Queue;
    ChildrenArr children;
    readData(queue, children, inputF);
    findDescendants(queue, children, outputF, 1);

    delete queue;
    inputF.close();
    outputF.close();

    return 0;
}

void readData(Queue* queue, ChildrenArr& children, std::ifstream& inputF)
{
    std::string nextLine;
    std::cout << "=====Начало ввода=====\n";
    while ((std::getline(inputF, nextLine))) {
        int separator = nextLine.find(" ");
        if (separator > -1) {
            std::string parent = nextLine.substr(0, separator);
            std::string child = nextLine.substr(separator + 1);
            if ((parent.size() > 1) && (child.size() > 1)) {
                children[parent][child] = true;
                std::cout << parent << " " << child << "\n";
            }
        }
        else { //знак окончания ввода данных — одиночное имя
            std::cout << "Поиск потомков " << nextLine << "\n";
            queue->push(nextLine);
            std::cout << "=====Конец ввода=====\n";
            return;
        }
    }

    std::cout << "Не указан предок, чьих потомков надо найти.\n";
    std::cout << "=====Конец ввода=====\n";
}

void findDescendants(Queue* queue, ChildrenArr& children, std::ofstream&
outputF, int genNumber) {

    Queue* nextGen = new Queue;

```

```

std::string names;
while (!queue->isEmpty()) {
    std::string ancestor = queue->pop();
    std::map<std::string, bool> ::iterator it =
children[ancestor].begin();
    for (int i = 0; it != children[ancestor].end(); it++, i++) {
        if (it->second) {
            names += it->first + "; ";
            std::cout << i << ") ПОТОМОК " << it->first << " ОТ
" << ancestor << "\n";
            nextGen->push(it->first);
        }
    }
}
if (!nextGen->isEmpty()) {
    std::cout << "Поколение " << genNumber << ": " << names <<
"\n";
    outputF << "Поколение " << genNumber << ": " << names << "\n";
    findDescendants(nextGen, children, outputF, genNumber + 1);
}
else if (genNumber == 1) {
    std::cout << "Потомков не найдено.\n";
    outputF << "Потомков не найдено.\n";
}

delete nextGen;
}

```