PersonTree

Черкашина Елена БИБ221

# Алфавитный указатель классов

## Классы

Классы с их кратким описанием.

**Person (Person - структура для хранения персональных данных )** pagenum

**tree (Бинарное дерево, хранящее в узлах структуры Person с персональными данными )** pagenum

**tree\_node (Узел бинарного дерева )** pagenum

# Список файлов

## Файлы

Полный список файлов.

**Person.cpp**

**Person.h**  pagenum

**PersonTree.cpp**

**tree.cpp**  pagenum

**tree.h**  pagenum

# Классы

## Структура Person

**Person** - структура для хранения персональных данных.

#include <Person.h>

### Открытые члены

**Person** ()

*Конструктор по умолчанию. Создаёт пустую структуру.*

**Person** (const std::string &csv)

*Конструктор, создающий экземпляр структуры из CSV-строки, считанной из файла.*

**Person** (const **Person** &other)=default

*Копирующий конструктор*

std::string **toCsv** () const

*Преобразует объект в CSV-строку.*

bool **operator<** (const **Person** &other) const

*Оператор сравнения "меньше".*

bool **operator>** (const **Person** &other) const

*Оператор сравнения "больше".*

### Открытые атрибуты

**Phone** **m\_phone**

*Номер телефона*

std::string **m\_fio**

*ФИО*

unsigned int **m\_age**

*Возраст*

std::string **m\_address**

*Адрес по месту регистрации*

### Подробное описание

**Person** - структура для хранения персональных данных.

### Конструктор(ы)

#### Person::Person ()[inline]

Конструктор по умолчанию. Создаёт пустую структуру.

#### Person::Person (const std::string & *csv*)

Конструктор, создающий экземпляр структуры из CSV-строки, считанной из файла.

##### Аргументы

|  |  |
| --- | --- |
| *csv* | Строка данных из CSV-файла. |

#### Person::Person (const Person & *other*)[default]

Копирующий конструктор

##### Аргументы

|  |  |
| --- | --- |
| *other* | Источник для копирования. |

### Методы

#### bool Person::operator< (const Person & *other*) const

Оператор сравнения "меньше".

##### Аргументы

|  |  |
| --- | --- |
| *other* | Экземпляр для сравнения с данным. |

##### Возвращает

True если данный экземпляр меньше другого.

#### bool Person::operator> (const Person & *other*) const

Оператор сравнения "больше".

##### Аргументы

|  |  |
| --- | --- |
| *other* | Экземпляр для сравнения с данным. |

##### Возвращает

True если данный экземпляр больше другого.

#### std::string Person::toCsv () const

Преобразует объект в CSV-строку.

##### Возвращает

Объект в виде std::string.

### Данные класса

#### std::string Person::m\_address

Адрес по месту регистрации

#### unsigned int Person::m\_age

Возраст

#### std::string Person::m\_fio

ФИО

#### Phone Person::m\_phone

Номер телефона

#### Объявления и описания членов структур находятся в файлах:

#### Person.hPerson.cpp

## Структура tree

Бинарное дерево, хранящее в узлах структуры **Person** с персональными данными.

#include <tree.h>

### Открытые члены

**tree\_node** \* **insert** (const **Person** &val)

*Вставляет (встраивает) узел в структуру двоичного дерева*

bool **remove** (const **Person** &val)

*Ищет и удаляет узел из дерева, перестраивая его структуру*

std::vector< **Person** > **to\_vector** () const

*Преобразует данные дерева в сортированный вектор*

bool **load** (const std::string &path)

*Загружает двоичное дерево из CSV-файла*

bool **save** (const std::string &path) const

*Сохраняет двоичное дерево в CSV-файл*

**tree\_node** \* **find** (const **Phone** &phone) const

*Поиск в двоичном дереве узла с персональными данными по номеру телефона.*

### Открытые атрибуты

std::unique\_ptr< **tree\_node** > **root**

*Корневой узел двоичного дерева*

### Подробное описание

Бинарное дерево, хранящее в узлах структуры **Person** с персональными данными.

### Методы

#### tree\_node \* tree::find (const Phone & *phone*) const

Поиск в двоичном дереве узла с персональными данными по номеру телефона.

##### Аргументы

|  |  |
| --- | --- |
| *phone* | Номер телефона. |

##### Возвращает

Null в случае неудачи, иначе - указатель на найденный **tree\_node**.

#### tree\_node \* tree::insert (const Person & *val*)

Вставляет (встраивает) узел в структуру двоичного дерева

##### Аргументы

|  |  |
| --- | --- |
| *val* | Персональные данные. |

##### Возвращает

Null при неудаче, иначе - указатель на добавленный узел.

#### bool tree::load (const std::string & *path*)

Загружает двоичное дерево из CSV-файла

Конец заимствованного (с изменениями) фрагмента кода //////////////////////

##### Аргументы

|  |  |
| --- | --- |
| *path* | Путь к файлу для загрузки. |

##### Возвращает

True в случае успеха, false - в противном случае.

#### bool tree::remove (const Person & *val*)

Ищет и удаляет узел из дерева, перестраивая его структуру

##### Аргументы

|  |  |
| --- | --- |
| *val* | Данные, которые следует удалить. |

##### Возвращает

True в случае успеха, false - в противном случае.

#### bool tree::save (const std::string & *path*) const

Сохраняет двоичное дерево в CSV-файл

##### Аргументы

|  |  |
| --- | --- |
| *path* | Путь к файлу для сохранения. |

##### Возвращает

True в случае успеха, false - в противном случае.

#### std::vector< Person > tree::to\_vector () const

Преобразует данные дерева в сортированный вектор

##### Возвращает

Вектор структур **Person**.

### Данные класса

#### std::unique\_ptr<tree\_node> tree::root

Корневой узел двоичного дерева

#### Объявления и описания членов структур находятся в файлах:

#### tree.htree.cpp

## Структура tree\_node

Узел бинарного дерева.

#include <tree.h>

### Открытые члены

**tree\_node** (const **Person** &val, **tree\_node** \*parent)

*Конструктор, создающий узел дерева и инициализирующий его значение, а также указатель на родительский узел.*

### Открытые атрибуты

**Person** **value**

*Персональные данные*

**tree\_node** \* **up**

*Указатель на родительский узел, для корня дерева - nullptr.*

std::unique\_ptr< **tree\_node** > **left**

*Дочерний узел слева*

std::unique\_ptr< **tree\_node** > **right**

*Дочерний узел справа*

### Подробное описание

Узел бинарного дерева.

### Конструктор(ы)

#### tree\_node::tree\_node (const Person & *val*, tree\_node \* *parent*)[inline]

Конструктор, создающий узел дерева и инициализирующий его значение, а также указатель на родительский узел.

##### Аргументы

|  |  |
| --- | --- |
| *val* | Значение узла - экземпляр структуры **Person**. |
| *parent* | Указатель на родительский узел. |

### Данные класса

#### std::unique\_ptr<tree\_node> tree\_node::left

Дочерний узел слева

#### std::unique\_ptr<tree\_node> tree\_node::right

Дочерний узел справа

#### tree\_node\* tree\_node::up

Указатель на родительский узел, для корня дерева - nullptr.

#### Person tree\_node::value

Персональные данные

#### Объявления и описания членов структуры находятся в файле:

**tree.h**

# Файлы

## Файл Person.cpp

#include "Person.h"

#include <vector>

#include <sstream>

#include <iostream>

### Макросы

#define **delim**  '\x09'

### Функции

std::ostream & **operator<<** (std::ostream &s, const **Person** &person)

*Оператор вывода персональных данных в стандартный поток вывода*

std::istream & **operator>>** (std::istream &s, **Person** &person)

*Оператор ввода персональных данных из стандартного потока ввода*

### Макросы

#### #define delim  '\x09'

### Функции

#### std::ostream & operator<< (std::ostream & *s*, const Person & *person*)

Оператор вывода персональных данных в стандартный поток вывода

##### Аргументы

|  |  |  |
| --- | --- | --- |
| in,out | *s* | Поток std::ostream для ввода в него. |
| in,out | *person* | Структура с персональными данными. |

##### Возвращает

Ссылку на поток вывода.

#### std::istream & operator>> (std::istream & *s*, Person & *person*)

Оператор ввода персональных данных из стандартного потока ввода

##### Аргументы

|  |  |  |
| --- | --- | --- |
| in,out | *s* | Поток std::istream для ввода из него. |
| in,out | *person* | Структура для ввода персональных данных. |

##### Возвращает

Ссылку на поток ввода.

## Файл Person.h

#include <string>

### Классы

struct **Person*Person*** *- структура для хранения персональных данных.*

### Определения типов

typedef unsigned long long **Phone**

*Определение типа для представления номера телефона - Phone.*

### Функции

std::ostream & **operator<<** (std::ostream &s, const **Person** &person)

*Оператор вывода персональных данных в стандартный поток вывода*

std::istream & **operator>>** (std::istream &s, **Person** &person)

*Оператор ввода персональных данных из стандартного потока ввода*

### Типы

#### typedef unsigned long long Phone

Определение типа для представления номера телефона - Phone.

### Функции

#### std::ostream & operator<< (std::ostream & *s*, const Person & *person*)

Оператор вывода персональных данных в стандартный поток вывода

##### Аргументы

|  |  |  |
| --- | --- | --- |
| in,out | *s* | Поток std::ostream для ввода в него. |
| in,out | *person* | Структура с персональными данными. |

##### Возвращает

Ссылку на поток вывода.

#### std::istream & operator>> (std::istream & *s*, Person & *person*)

Оператор ввода персональных данных из стандартного потока ввода

##### Аргументы

|  |  |  |
| --- | --- | --- |
| in,out | *s* | Поток std::istream для ввода из него. |
| in,out | *person* | Структура для ввода персональных данных. |

##### Возвращает

Ссылку на поток ввода.

## Person.h

См. документацию.1 #pragma once

2 #include <string>

3

5 typedef unsigned long long Phone;

6

8 struct Person

9 {

11 Person() : m\_phone(0), m\_age(0)

12 {

13 }

14

15 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

20 Person(const std::string &csv);

21

22 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

27 Person(const Person &other) = default;

28

29 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

34 std::string toCsv() const;

35

36 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

43 bool operator <(const Person &other) const;

44

45 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

52 bool operator >(const Person &other) const;

53

54 Phone m\_phone;

55 std::string m\_fio;

56 unsigned int m\_age;

57 std::string m\_address;

58 };

59

60 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

68 std::ostream &operator << (std::ostream &s, const Person &person);

69

70 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

78 std::istream &operator >> (std::istream &s, Person &person);

## Файл PersonTree.cpp

#include "tree.h"

#include "doctest.h"

#include <iostream>

#include <numeric>

#include <vector>

### Функции

**Phone** **getPhone** ()

*Запросить ввод из консоли номера телефона*

int **main** ()

### Функции

#### Phone getPhone ()

Запросить ввод из консоли номера телефона

##### Возвращает

Введённый пользователем номер телефона.

#### int main ()

#### 

## Файл tree.cpp

#include "tree.h"

#include <fstream>

### Функции

**tree\_node** \* **InsertHelper** (std::unique\_ptr< **tree\_node** > &node, const **Person** &val, **tree\_node** \*parent)

*Начало заимствованного (с изменениями) фрагмента кода //////////////////////*

bool **RemoveHelper** (std::unique\_ptr< **tree\_node** > &node, const **Person** &val)

void **Tree\_to\_vector** (**tree\_node** const &t, std::vector< **Person** > &res)

**tree\_node** \* **FindHelper** (const std::unique\_ptr< **tree\_node** > &node, const **Phone** &phone, **tree\_node** \*parent=nullptr)

### Функции

#### tree\_node \* FindHelper (const std::unique\_ptr< tree\_node > & *node*, const Phone & *phone*, tree\_node \* *parent* = nullptr)

#### tree\_node \* InsertHelper (std::unique\_ptr< tree\_node > & *node*, const Person & *val*, tree\_node \* *parent*)

Начало заимствованного (с изменениями) фрагмента кода //////////////////////

#### bool RemoveHelper (std::unique\_ptr< tree\_node > & *node*, const Person & *val*)

#### void Tree\_to\_vector (tree\_node const & *t*, std::vector< Person > & *res*)

#### 

## Файл tree.h

#include "Person.h"

#include <memory>

#include <vector>

### Классы

struct **tree\_node***Узел бинарного дерева.*

struct **tree***Бинарное дерево, хранящее в узлах структуры* ***Person*** *с персональными данными.*

## tree.h

См. документацию.1 #include "Person.h"

2 #include <memory>

3 #include <vector>

4

6 struct tree\_node

7 {

8 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

15 tree\_node(const Person &val, tree\_node \*parent) : value(val), up(parent)

16 {

17 }

18

19 Person value;

20 tree\_node \*up;

21 std::unique\_ptr<tree\_node> left;

22 std::unique\_ptr<tree\_node> right;

23 };

24

26 struct tree

27 {

28 std::unique\_ptr<tree\_node> root;

29

30 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

37 tree\_node \* insert(const Person &val);

38

39 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

46 bool remove(const Person &val);

47

48 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

53 std::vector<Person> to\_vector() const;

54

55 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

62 bool load(const std::string &path);

63

64 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

71 bool save(const std::string &path) const;

72

73 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

80 tree\_node \* find(const Phone &phone) const;

81 };