#pragma once

#include<iostream>

#include<cstring>

#define tab " "

using namespace std;

//Конкретное нарушение автомобиля

class Violation

{

int Data; //дата нарушения

string Location; //место нарушения

int Article; //статья КоАП

Violation\* pPrev;

Violation\* pNext;

public:

int getData() { return Data; }

string getLocation() { return Location; }

int getArticle() { return Article; }

Violation\* getpPrev() { return pPrev; }

Violation(int data, string location, int article, Violation\* pPrev = nullptr, Violation\* pNext = nullptr) :

Data(data), Location(location), Article(article), pPrev (pPrev), pNext(pNext) {

cout << "Vcon" << endl;

};

Violation(const Violation& other) :

Data(other.Data), Location(other.Location), Article(other.Article), pPrev(other.pPrev), pNext(other.pNext) {};

~Violation() {};

Violation& operator=(const Violation& other)

{

if (this != &other)

{

Data = other.Data;

Location = other.Location;

Article = other.Article;

pPrev = other.pPrev;

pNext = other.pNext;

}

return \*this;

}

friend class ListViolationsCar;

};

#pragma once

#include "Violation.h"

//Лист нарушений конкретного автомобиля

class ListViolationsCar

{

Violation\* Head;

Violation\* Tail;

int Size;

public:

Violation\* getHead() { return Head; }

ListViolationsCar():Head(nullptr), Tail(nullptr), Size(0){ cout << "LVCcon" << endl; };

~ListViolationsCar() { while (Tail) pop\_back(); };

void push\_front(/\*ListViolationsCar& list,\*/ int data, string location, int article)

{

if (empty())

Head = Tail = new Violation(data, location, article);

else Head = Head->pPrev = new Violation(data, location, article, nullptr, Head);

Size++;

cout << "Size " <<Size<< endl;

}

void pop\_back()

{

if (Tail)

{

Violation\* Temp = Tail;

if (Temp->pPrev)

{

Tail = Tail->pPrev;

Tail->pNext = nullptr;

}

else Head = Tail = nullptr;

delete Temp;

Size--;

}

}

bool empty() const {return Head == nullptr && Tail == nullptr;}

void print()const

{

for (Violation\* Temp = Head; Temp != nullptr; Temp = Temp->pNext)

{

cout << Temp << tab << "Data - " << Temp->Data << tab;

cout << "Location - " << Temp->Location << tab;

cout << "Article - " << Temp->Article << endl;

}

cout << "Total violations by car - " << Size << endl;

}

#pragma once

#include "ListViolationsCar.h"

//автомобиль со списком своих нарушений

class CarViolator

{

int NamberCar;

ListViolationsCar\* List;

CarViolator\* pLeft;

CarViolator\* pRight;

public:

CarViolator(int namberCar, CarViolator\* pleft=nullptr, CarViolator\* pright=nullptr) :

NamberCar(namberCar), pLeft(pLeft), pRight(pRight)

{

List = new ListViolationsCar;

cout << "CVcon" << endl;

cout << List << endl;

};

~CarViolator() {};

bool is\_leaf() { return pLeft == pRight; }

friend class DBviolators;

};

#pragma once

#include "CarViolator.h"

//база данных автомобилей-нарушителей ПДД

class DBviolators

{

CarViolator\* Root;

public:

DBviolators() :Root(nullptr) { cout << "DBcon" << endl; };

~DBviolators() { cout << "DBde" << endl; };

void insert(int namberCar, int data, string location, int article)

{ insert(namberCar, data, location, article, this->Root);

cout << "DBins" << endl;

}

/\*

void clear()

{

clear(this->Root);

this->Root = nullptr;

}\*/

void print()

{

print(this->Root);

cout << endl;

}

/\*

void print(int namberCar)

{

print(this->Root);

cout << endl;

}\*/

private:

void insert(int namberCar, int data, string location, int article, CarViolator\*& root)

{

//if (!root) return;

if (!this->Root)

{

cout << "ins Root" << endl;

this->Root = new CarViolator(namberCar);

Root->List->push\_front(data, location,article);

}

//if (namberCar = Root->NamberCar) return;

//else

if (namberCar < Root->NamberCar)

{

if (!Root->pLeft)

{

Root->pLeft = new CarViolator(namberCar);

Root->List->push\_front(data, location, article);

}

else

insert(namberCar,data,location,article,Root->pLeft);

}

else if (namberCar > Root->NamberCar)

{

if (Root->pRight) insert(namberCar, data, location, article, Root->pRight);

else

{

Root->pRight = new CarViolator(namberCar);

Root->List->push\_front(data, location, article);

}

}

}

/\*

void clear(CarViolator\* Root)

{

if (Root == nullptr) return;

clear(Root->pLeft);

clear(Root->pRight);

delete Root;

}\*/

void print(CarViolator\* Root)

{

if (!Root) return;

print(Root->pLeft);

cout << Root->List<<tab<<Root->NamberCar << tab << Root->List->Size << endl;

//cout<< Root->List->print() << endl;

print(Root->pRight);

}

};

#include<iostream>

#include<cstring>

#include<cstdlib>

#include "DBviolators.h"

using namespace std;

int main()

{

// setlocale(LC\_ALL, "Russian");

ListViolationsCar Car1;

Car1.push\_front(20180512, "Orel", 911);

Car1.push\_front(20180602, "Perm", 918);

Car1.print();

//cout << Car1 << endl; error

cout << "/////////////" << endl;

DBviolators DB;

DB.insert(568269, 20190610, "Samara", 911);

cout << "------------------" << endl;

DB.insert(125784, 20200618, "Perm", 923);

cout << "------------------" << endl;

DB.insert(744784, 20180512, "Orel", 911);

cout << "/////////////" << endl;

DB.print();

// cout << DB << endl;

}

/////////////////////////////////////////////////////////

<https://www.cyberforum.ru/cpp-beginners/thread1888806.html>

#include <iostream>

#include <cstring>

#include <cstdlib>

using namespace std;

struct ListViolations

{

char\* violat;

ListViolations\* next;

};

void Push(ListViolations\*& node, char\* newStr)

{

ListViolations\* tmp = new ListViolations;

tmp->violat = new char[strlen(newStr) + 1];

strcpy(tmp->violat, newStr);

tmp->next = NULL;

if (node == NULL)

node = tmp;

else

{

tmp->next = node;

node = tmp;

}

}

void ClearList(ListViolations\*& Head)

{

ListViolations\* curr = Head;

while (curr)

{

ListViolations\* temp = curr->next;

delete[] curr->violat;

delete curr;

curr = temp;

}

}

void PrintList(ListViolations\* Head)

{

ListViolations\* curr = Head;

while (curr)

{

std::cout << curr->violat << std::endl;

curr = curr->next;

}

}

////////////////

///////////////////////////

struct TreeNode

{

TreeNode\* parent;

TreeNode\* left;

TreeNode\* right;

int number;

ListViolations\* list;

};

class database

{

private:

int size;

TreeNode\* root;

public:

database();

~database();

void Add(int value, char\*);

void Add(TreeNode\*&, TreeNode\*, int, char\*);

void PrintAll(TreeNode\*);

// void PrintNumber(int);

//void PrintRangeNumber(TreeNode\*);

TreeNode\* ReturnRoot();

void DeleteAll(TreeNode\* temp);

};

database::database()

{

root = NULL;

size = 0;

}

//деструктор

database::~database()

{

DeleteAll(root);

}

//метод получения корня

TreeNode\* database::ReturnRoot()

{

return root;

}

//метод удаления всего

void database::DeleteAll(TreeNode\* temp)

{

if (temp)

{

DeleteAll(temp->left);

DeleteAll(temp->right);

ClearList(temp->list);

delete temp;

}

}

//метод добавления

void database::Add(TreeNode\*& head, TreeNode\* parent, int value, char\* str)

{

TreeNode\* temp = new TreeNode;

temp->number = value;

temp->list = NULL;

temp->left = temp->right = temp->parent = NULL;

if (!head)

{

head = temp;

Push(head->list, str);

}

else

{

if (value < head->number)

Add(head->left, head, value, str);

else if (value > head->number)

Add(head->left, head, value, str);

else

Push(head->list, str);

}

}

void database::Add(int value, char\* str)

{

Add(root, NULL, value, str);

}

//метод печати полной базы данных

void database::PrintAll(TreeNode\* temp)

{

if (temp)

{

PrintAll(temp->left);

cout << "Номер машины: " << temp->number << endl;

cout << "Нарушения: " << endl;

PrintList(temp->list);

PrintAll(temp->right);

}

}

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

{

setlocale(LC\_ALL, "Russian");

database bd;

bd.Add(9999, "Нарушение А");

bd.Add(9999, "Нарушение В");

bd.Add(9999, "Нарушение С");

bd.Add(2099, "Нарушение А");

bd.Add(1000, "Нарушение С");

bd.Add(9333, "Нарушение С");

bd.Add(9055, "Нарушение А");

bd.Add(9055, "Нарушение В");

bd.PrintAll(bd.ReturnRoot());

}