|  |  |
| --- | --- |
| Институт (факультет) | Институт информационных технологий |
| Кафедра | Кафедра математического и программного обеспечения ЭВМ |

# РАЗРАБОТКА АБСТРАКТНЫХ ТИПОВ ДАННЫХ

|  |  |
| --- | --- |
| **Дисциплина:** | ООП |
| **Темы:** | Инкапсуляция; классы; поля и компонентные функции; конструкторы и деструкторы класса |

**Среда разработки:** Microsoft Visual Studio

**Язык программирования:** C++

**Тип проекта:** Консольное приложение

**Задание на лабораторную работу №2**

**ТРЕБОВАНИЯ К РАЗРАБОТКЕ**

1. Каждый класс должен быть оформлен в отдельных файлах: заголовочный (.h) и файл с кодом (.cpp).
2. Запрещается использовать обработку исключительных ситуаций и генерировать исключения.
3. Придерживайтесь принципа DRY (Don’t repeat yourself).
4. Обязательно наличие комментариев.

**ЦЕЛЬ РАБОТЫ**

Изучить способы определения классов, правила доступа к элементам; приобрести практические навыки работы с объектами класса; изучить принципы и механизмы создания абстрактных типов данных.

**ЗАДАНИЯ**

1. *часов*
2. Разработайте АТД – класс, обеспечивающий хранение объектов класса из ЛР1, согласно варианту задания (см. раздел прил. 1).
3. Интерфейс класса может содержать конструкторы и должен содержать деструктор.
4. Интерфейс класса должен содержать функции:
   1. Добавления;
   2. Удаления;
   3. Вывода содержимого;
   4. Обработки (сортировка по полю класса, поиск по условию).

**ПРИМЕЧАНИЯ**

* У класса-контейнера не должно быть прямого доступа к полям класса предметной области (ЛР1).

Файл airTransport.h

#include "color.cpp"

#include <iostream>

#pragma once

class airTransport

{

private:

int Engine\_power; // мощность транспорта

color Color\_transport; // цвет транспорта

char\* Model; // модель транспорта

static int Count; // кол-во созданных объектов

int Capacity; // вместимость транспорта

public:

static const int max\_count\_airTransport = 10; // максимальное кол-во созданных объектов класса airTransport

airTransport(); // конструктор по-умолчанию

airTransport(int power, color colorTranport, const char\* m, int capactity\_transport); // конструктор класса

airTransport(const airTransport& transport); // копирующий контсруктор

~airTransport(); // деструктор классаефк

int get\_engine\_power(); // получить значение engine\_power

void set\_engine\_power(int power); // задать значение engine\_power

color get\_color\_transport(); // получить значение color\_transport

void set\_color\_transport(color colorTransport); // задать значение color\_transport

char\* get\_model(); // получить значение model

void set\_model(const char\* m); // задать значение model

int get\_capacity(); // получить значение capacity

void set\_capacity(int capacityTranport); // задать значение capacity

int get\_count(); // получить сколько создано объектов класса

friend std::ostream& operator<<(std::ostream& out, const airTransport& transport);

airTransport& operator=(const airTransport& transport);

bool operator==(const airTransport& transport);

};

Файл airTransport.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "airTransport.h"

#include <iostream>

using namespace std;

int airTransport::Count = 0;

int airTransport::get\_count()

{

return airTransport::Count;

}

airTransport::airTransport()

{

if (airTransport::Count > airTransport::max\_count\_airTransport) {

throw exception("Лимит по созданию объектов данного класса");

}

Engine\_power = 10;

Color\_transport = color::Red;

const char\* m = "plane";

Model = new char[strlen(m) + 1];

strcpy(Model, m);

Capacity = 10;

airTransport::Count++;

}

airTransport::airTransport(int power, color colorTransport, const char\* m, int capactity\_transport)

{

if (airTransport::Count > airTransport::max\_count\_airTransport) {

throw exception("Лимит по созданию объектов данного класса");

}

Engine\_power = power;

Color\_transport = colorTransport;

Model = new char[strlen(m) + 1];

strcpy(Model, m);

Capacity = capactity\_transport;

}

airTransport::~airTransport()

{

Count--;

delete this->Model;

}

airTransport::airTransport(const airTransport& transport)

{

Engine\_power = transport.Engine\_power;

Color\_transport = transport.Color\_transport;

Model = transport.Model;

Capacity = transport.Capacity;

}

int airTransport::get\_engine\_power()

{

return Engine\_power;

}

void airTransport::set\_engine\_power(int power)

{

Engine\_power = power;

}

int airTransport::get\_capacity()

{

return Capacity;

}

void airTransport::set\_capacity(int capacityTranport)

{

Capacity = capacityTranport;

}

char\* airTransport::get\_model()

{

return Model;

}

void airTransport::set\_model(const char\* m)

{

strcpy(Model, m);

}

color airTransport::get\_color\_transport()

{

return Color\_transport;

}

void airTransport::set\_color\_transport(color colorTranport)

{

Color\_transport = colorTranport;

}

ostream& operator<<(ostream& out, const airTransport& transport)

{

out << "Engine power: " << transport.Engine\_power << endl;

out << "Color: " << static\_cast<int>(transport.Color\_transport) << endl;

out << "Model: " << transport.Model << endl;

out << "Capacity: " << transport.Capacity << endl;

return out;

}

airTransport& airTransport::operator=(const airTransport& transport)

{

if (&transport != this)

{

this->Engine\_power = transport.Engine\_power;

this->Capacity = transport.Capacity;

this->Color\_transport = transport.Color\_transport;

this->Model = transport.Model;

}

return \*this;

}

bool airTransport::operator==(const airTransport& transport)

{

return this->Engine\_power == transport.Engine\_power && this->Capacity == transport.Capacity && this->Color\_transport == transport.Color\_transport && this->Model == transport.Model;

}

Файл List.h

#pragma once

#include "airTransport.h"

class List // Линейный односвязный список

{

private:

#pragma region Node

struct Node // Узел, в которм хранятся данные и указатель на следующий элемент, если он есть

{

airTransport data;

Node\* next;

};

#pragma endregion

int size; // размер списка

Node \*head; // первый элемент

public:

List(); // констурктор по-умолчанию

~List(); // деструктор

void add(const airTransport& transport); // добавление элемента

void pop\_front(); // удаление первого элемента

int length(); // длина списка

void clear(); // очистка списка

void show(); // вывод содержимого

void sort\_by\_engine\_power(); // сортировка по Engine\_power

int search(airTransport transport); // поиск элемента, если элемент есть вовзвращает индекс, если нет, то -1

airTransport& operator[](const int index); // перегрузка оператора

};

Файл List.cpp

#include "List.h"

#include <iostream>

using namespace std;

List::List()

{

size = 0;

head = nullptr;

}

List::~List()

{

clear();

}

void List::add(const airTransport& transport)

{

size++;

if (head == nullptr)

{

head = new Node;

head->data = transport;

head->next = nullptr;

}

else {

Node \*current = this->head;

while (current->next != nullptr)

{

current = current->next;

}

current->next = new Node;

current->next->data = transport;

current->next->next = nullptr;

}

}

void List::pop\_front()

{

if (head == nullptr) {

// обработка ошибки или возврат из метода

return;

}

Node\* temp = head;

head = head->next;

delete temp;

size--;

}

int List::length()

{

return size;

}

void List::clear()

{

while (size)

{

pop\_front();

}

}

void List::show()

{

for (int i = 0; i < this->size; i++) {

cout << this->operator[](i) << endl << endl;

}

}

void List::sort\_by\_engine\_power()

{

for (int i = 0; i < this->length(); i++)

{

bool flag = true;

for (int j = 0; j < this->length() - (i + 1); j++) {

if (this->operator[](j).get\_engine\_power() > this->operator[](j + 1).get\_engine\_power()) {

flag = false;

std::swap(this->operator[](j), this->operator[](j + 1));

}

}

if (flag) {

break;

}

}

}

int List::search(airTransport transport)

{

for (int i = 0; i < this->length(); i++)

{

if (this->operator[](i) == transport)

return i;

}

return -1;

}

airTransport& List::operator[](const int index)

{

int counter = 0;

Node\* current = this->head;

while (current->next != nullptr)

{

if (counter == index)

return current->data;

current = current->next;

counter++;

}

}

Файл source.cpp

#include "airTransport.h"

#include "List.h"

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL,"rus");

airTransport a;

airTransport x(500, color::White, "rocket", 20);

airTransport b(23, color::Black, "helicopter", 8);

airTransport d;

List lst;

lst.add(a);

lst.add(x);

lst.add(b);

cout << lst.search(a) << endl;

return 1;

}

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Каковы особенности синтаксиса и семантики конструкторов и деструкторов?

2. В чем заключаются особенности применения конструкторов по умолчанию?

3. В каких случаях вызывается копирующий конструктор?

4. При каких условиях конструктор может использоваться для преобразования типа?

5. В каких ситуациях необходимы инициализаторы конструктора, синтаксис и семантика их использования?

1. *Конструктор* (*constructor*) – это компонентная функция, предназначенная для создания объектов типа своего класса. Под *созданием* понимается инициализация компонентных данных и при необходимости распределения под них свободной памяти с помощью new. Имя конструктора совпадает с именем класса.

*Деструктор* (*destructor*) – это компонентная функция, предназначенная для уничтожения объектов типа своего класса. Под *уничтожением* понимается освобождение свободной памяти с помощью delete. Имя деструктора – это предваряемое символом «~» Имя класса.

2. Отсутствие параметров: конструктор по умолчанию не принимает никаких параметров при создании объекта; Автоматическое создание: если в классе не определен явный конструктор, компилятор автоматически создаст конструктор по умолчанию.

3. Копирующий конструктор предназначен для копирования одного значения типа класса в другое и имеет вид

type :: type (const type & x)

Копирующий конструктор вызывается в следующих случаях:

* если объект типа type инициализируется значением типа type;
* когда параметр типа type передается функции по значению;
* когда функция возвращает значение типа type.

При отсутствии копирующего конструктора эти действия по умолчанию выполняются поэлементно.

4. Конструктор с одним параметром автоматически является функцией преобразования типа параметра к типу класса.

Н а п р и м е р :

type :: type (int); // преобразует int к type.

Использование explicit в начале объявления запрещает такое преобразование.

5. Класс, содержащий элементы, тип которых требует конструктор, использует инициализаторы – разделенный запятыми список вызовов конструкторов, стоящий после двоеточия. Конструктор вызывается с помощью имени элемента, за которым в круглых скобках следует список аргументов. Элементы инициализируются в порядке объявления их в классе.

**Лр3.**

1. Назовите алгоритм соответствия для каждого аргумента.
2. С какой целью применяется перегрузка операторов?
3. Для каких операторов нельзя использовать перегрузку?
4. В каком случае используется функция-член, имеющая пустой список аргументов?
5. Что произойдет, если перегруженная операция [ ] будет иметь тип возвращаемого значения int, а не int&?

1.