|  |  |
| --- | --- |
| 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 №1**

**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.

Изучить синтаксис и семантику определения и вызова конструкторов и деструкторов. Приобрести практические навыки создания и уничтожения объектов. Изучить особенности применения различных видов конструкторов.

**ЗАДАНИЯ**

*8 часов*

1. Ознакомьтесь с УМП по ООП часть 1 раздел IV.
2. Разработайте алгоритм и программу, реализующую абстрактный тип данных (АТД) – класс, согласно варианту задания (см. прил. 1):
   1. Предусмотрите закрытую реализацию и открытый интерфейс.
   2. Интерфейс должен содержать функции вывода содержимого и функции доступа к некоторым полям.
   3. Элементы данных должны быть представлены из 5 полей, один из которых должен быть статическим, второй иметь тип указатель, а третий тип перечисление, остальные на свое усмотрение.
   4. Примените основные виды конструкторов и определитесь с деструктором класса.
   5. Проверьте работоспособность АТД на тестовом наборе данных.

Файл color.cpp

enum class color {

Red,

White,

Black,

Green

};

Файл airTransport.h

#include "color.cpp"

#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(); // деструктор класса

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

};

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) {

cout << "Лимит по созданию объектов данного класса" << endl;

return;

}

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) {

cout << "Лимит по созданию объектов данного класса" << endl;

return;

}

Engine\_power = power;

Color\_transport = colorTransport;

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

strcpy(Model, m);

Capacity = capactity\_transport;

}

airTransport::~airTransport() {

Count--;

cout << this->Model << " уничтожен" << endl;

delete[] Model;

}

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;

}

Файл source.cpp

#include "airTransport.h"

#include <iostream>

using namespace std;

int main() {

setlocale(LC\_ALL,"rus");

airTransport plane;

cout << plane.get\_capacity() << endl;

airTransport helicopter(100, color::Black, "helicopter", 8);

cout << helicopter.get\_capacity() << endl;

return 1;

}

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Дайте определение понятия «класс». Сформулируйте правила доступа к его элементам.
2. С какой целью в классе объединены компонентные данные и компонентные функции?
3. Каким образом осуществляется доступ к открытым и закрытым элементам?
4. Опишите назначение дружественных функций, назовите их разновидности.
5. Что понимается под указателем this?
6. Каковы особенности использования статических компонентных данных?
7. В чем заключается синтаксис и семантика компонентных функций static и const?
8. Каким образом могут изменяться компонентные данные объектов, объявленных константами?
9. Каковы особенности создания вложенных классов?

1. *Класс* – это гетерогенный агрегатный тип, имеющий элементы, данные и элементы функций, обеспечивающий сокрытие данных и наследование.

Структура класса имеет вид:

class имя\_типа {public: // элементы доступны и наследуются

----- ------ -------

----- ------ -------

protected: //элементы защищены и наследуются

----- ------ -------

----- ------ -------

private: //элементы закрыты, не наследуются

----- ------ -------

----- ------ -------

};

Спецификаторы доступа public, protected и private определяют степень открытости элементов, которые следуют за ними.

По умолчанию элементы структуры являются открытыми, элементы класса – закрытыми.

2. Объединение компонентных данных и функций в классе позволяет создавать объекты с определенным состоянием и поведением, предоставляет удобный способ организации кода и обеспечивает гибкость и модульность при разработке программного обеспечения.

3. Если поле или метод находится в public, то обращаться через «.». Если приватные данные, то обращаться к компонентным функциям, если они созданы.

4. Дружественная функция должна быть объявлена внутри объявления класса, по отношению к которому она является дружественной, с использованием спецификатора friend. Такая функция имеет доступ к закрытым элементам класса наряду с компонентными функциями. Дружественными могут быть как внешние, так и компонентные функции, в этом случае для указания имени функции в дружественном классе используется оператор разрешения области видимости **::** .

5. Ключевое слово this неявно объявляет указатель на себя. Может иметь место только в нестатических компонентных функциях. Заменяет объявление закрытого элемента:

6. Статические элементы данных хранятся в одном месте, являются общими для всех объектов класса и существуют независимо от них. Статическая компонентная функция доступа к компонентным данным не имеет, за исключением статических.

7. Обращение через оператор разрешения области видимости к статическим компонентным данным и функциям является предпочтительным:

class\_type :: статический\_элемент

class\_type :: статическая\_функция ( )

Именно так отражается тот факт, что элемент и функция являются статическими.

Компонентная функция, объявленная как const, не может изменять свои неявные аргументы, она получает неявный указатель:

const class\_type \* const this.

8. Объекты класса могут быть объявлены как константы, и это гарантирует постоянство компонентных данных. Отказаться можно с помощью const\_cast или индивидуально для отдельных компонентных данных с использованием ключевого слова mutable.

9. Вложение классов позволяет вкладывать определения функций - компонентная функция должна быть определена внутри своего локального класса, и на данную функцию нельзя ссылаться вне области видимости ее локального класса.