

**Цель работы:** получить практические навыки создания иерархии классов, использования статических компонентов класса и работы с потоковым вводом/выводом на C++.

**Задание:**

- 1) Написать программу, в которой создается иерархия классов. Включить полиморфные объекты в список, связанный список или контейнер. Показать использование виртуальных функций.
- 2) Написать программу, в которой создаются и разрушаются объекты, определенного пользователем класса. При выходе и запуске программы заново, данные не теряются.

**Вариант задания:**

- 1) Млекопитающие, амфибии, птицы, рыбы, пресмыкающиеся;

**Ход работы**

**Теория:**

1) Программа выполнена в среде разработки Visual Studio 2019. Microsoft Visual Studio - это программная среда по разработке приложений для ОС Windows, как консольных, так и с графическим интерфейсом.

2) Наследование — один из четырех важнейших механизмов объектно-ориентированного программирования (наряду с инкапсуляцией, полиморфизмом и абстракцией), позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом.

`private` — член класса может использоваться только функциями — членами данного класса и функциями — “друзьями” своего класса. В производном классе он недоступен.

`protected` — то же, что и `private`, но дополнительно член класса с данным атрибутом доступа может использоваться функциями-членами и функциями — “друзьями” классов, производных от данного.

`public` — член класса может использоваться любой функцией, которая является членом данного или производного класса, а также к `public` - членам возможен доступ извне через имя объекта.

3) Виртуальная функция — это функция, которая определяется в базовом классе, а любой порожденный класс может ее переопределить. Виртуальная функция вызывается только через указатель или ссылку на базовый класс.

Чистая виртуальная функция ничего не делает и недоступна для вызовов. Ее назначение — служить основой для подменяющих ее функций в производных классах. Абстрактный класс может использоваться только в качестве базового для производных классов.

4) Абстрактные классы.

Абстрактным называется класс, в котором есть хотя бы одна чистая (пустая) виртуальная функция. Чистой виртуальной функцией называется компонентная функция, которая имеет следующее определение: *virtual имя\_функции (список\_формальных\_параметров) = 0;*

Механизм абстрактных классов разработан для представления общих понятий, которые в дальнейшем предполагается конкретизировать. При этом построение иерархии классов выполняется по следующей схеме. Во главе иерархии стоит абстрактный базовый класс. Он используется для наследования интерфейса. Производные классы будут конкретизировать и реализовать этот интерфейс. В абстрактном классе объявлены чистые виртуальные функции, которые по сути есть абстрактные методы.

5) Для работы с файлами необходимо подключить заголовочный файл `<fstream>`. В нем определены несколько классов и подключены заголовочные файлы

`<ifstream>` — файловый ввод ;

`<ofstream>` — файловый вывод.

Файловый ввод-вывод аналогичен стандартному вводу-выводу, единственное отличие — это то, что ввод-вывод выполняются не на экран, а в файл.

Если ввод-вывод на стандартные устройства выполняется с помощью объектов `cin` и `cout`, то для организации файлового ввода-вывода достаточно создать собственные объекты, которые можно использовать аналогично этим операторам.

## UML диаграмма классов:

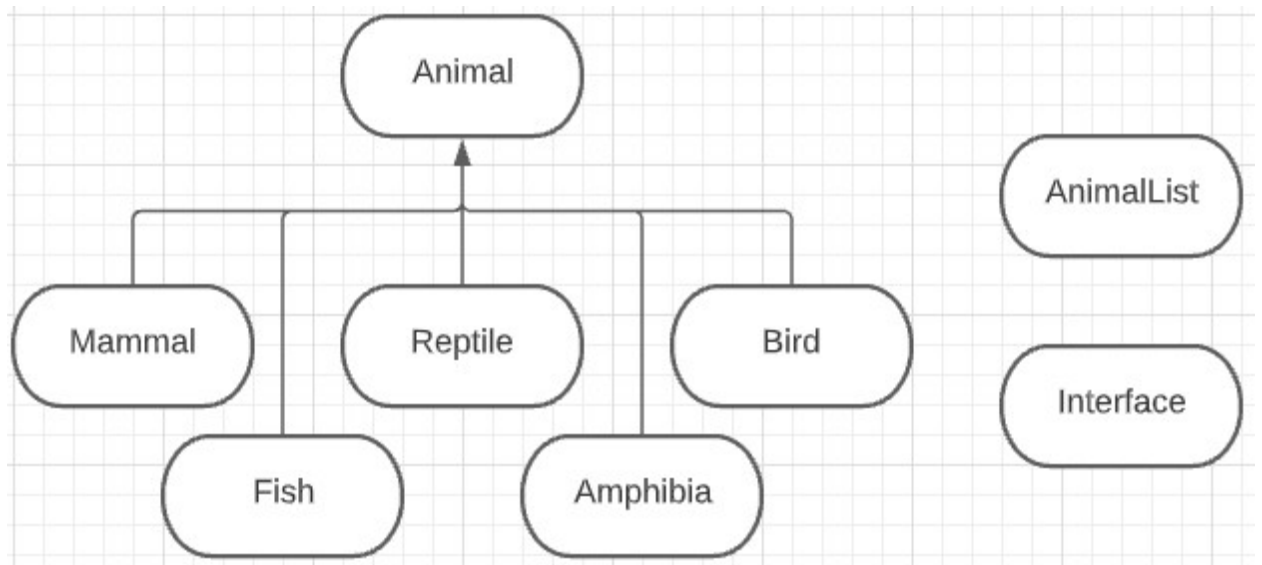


Рис.1 Диаграмма классов

### Определение классов:

Animal - базовый класс, содержащий общий для всех функционал (название животного, средний вес, класс животного)

Fish - наследует функционал класса Animal, а так же добавляется количество плавников.

Amphibia - наследует функционал класса Animal, а так же добавляется строка с местом обитания животного.

Reptile - наследует функционал класса Animal, а так же проверяет наличие панциря.

Bird – наследует функционал класса Animal, а так же добавляет поле «длина тела».

Interface - класс для удобного представления программы в консоли

AnimalList - класс для манипуляций над объектами класса (считывание, запись, отображение).

**Реализация конструкторов с параметрами и деструктора, методов добавления объектов в список, просмотра списка:**

```
=====
1) Добавить элемент
2) Показать данные всех элементов списка
3) Удалить элемент списка
4) Удалить все элементы в списке
5) Вывести данные элементов по типу данных
6) Выход
7) Чтение с файла
Выберите действие: _
```

Рис.2 Запуск программы

```
===== Доступные к добавлению элементы =====
1) Amphibia
2) Fish
3) Reptile
4) Bird
5) Mammal
Введите номер объекта, который хотите создать: _
```

Рис.3 Добавление объекта в список

```
Создание животного с классом: Amphibia
Введите вес животного: 1
Введите название животного: Лягушка
Место обитания: Водоём
```

Рис.4 Конструктор с параметрами

```
=====
ID объекта: 1
Название животного: Лягушка
Класс животного: Amphibia
Средний вес животного: 1
Место обитания :Водоём
=====
ID объекта: 2
Название животного: Черепаха
Класс животного: Reptile
Средний вес животного: 5
Наличие панциря :1
=====
ID объекта: 3
Название животного: Ворон
Класс животного: Bird
Средний вес животного: 1
Длина тела (в см.) :65_
```

Рис.5 Просмотр списка

```
Введите ID объекта, чтобы его удалить: 2
0151EA98 Вы удалили животное: Черепаха с ID: 2
```

Рис.6 Удаление объекта по параметру ID

```

=====
ID объекта: 1
Название животного: Лягушка
Класс животного: Amphibia
Средний вес животного: 1
Место обитания :Водоём
=====
ID объекта: 3
Название животного: Ворон
Класс животного: Bird
Средний вес животного: 1
Длина тела (в см.) :65_

```

Рис.7Список после удаления объекта

```

1) Amphibia
2) Fish
3) Reptile
4) Bird
5) Mammal
Выберите тот класс, элементы которого будут выведены:
1
=====
ID объекта: 1
Название животного: Лягушка
Класс животного: Amphibia
Средний вес животного: 1
Место обитания :Водоём_

```

Рис.8 Просмотр списка с параметром

```

=====
1) Добавить элемент
2) Показать данные всех элементов списка
3) Удалить элемент списка
4) Удалить все элементы в списке
5) Вывести данные элементов по типу данных
6) Выход
7) Чтение с файла
Выберите действие: 6
File opened!Объекты сохранены в data_base.txt
00C08FE0 Вы удалили животное: Лягушка с ID: 1
00C1BA88 Вы удалили животное: Ворон с ID: 3

```

Рис.9 Запись и сохранение объектов в текстовом файле

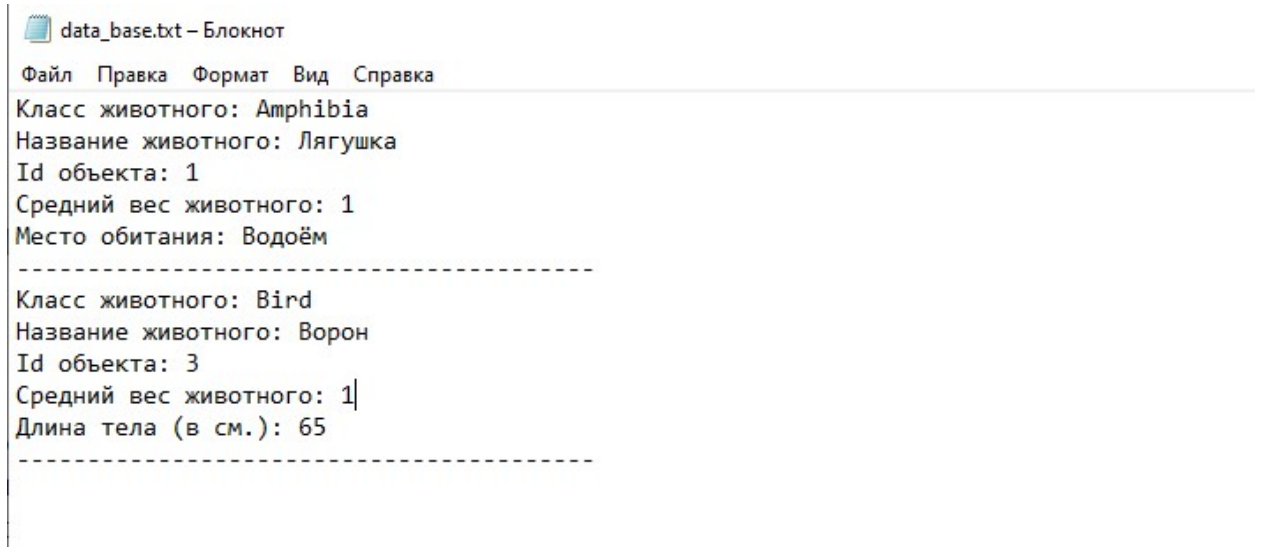


Рис.10 Хранение объектов в текстовом файле data\_base.txt

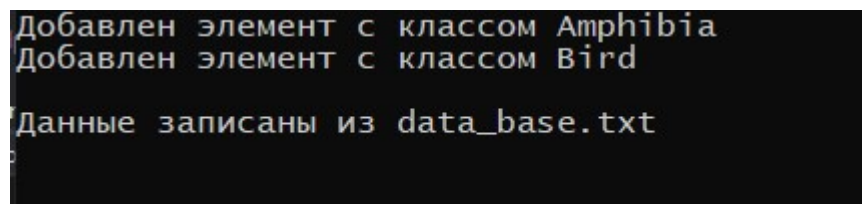


Рис. 11 Прочитаем созданный файл

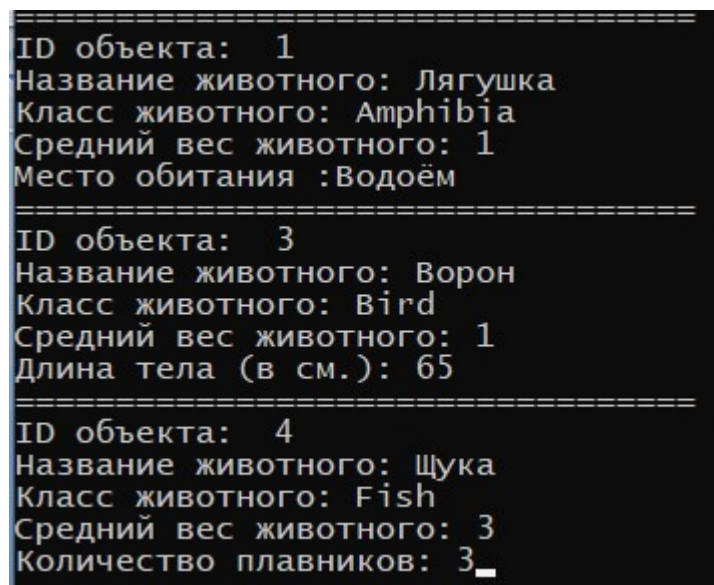
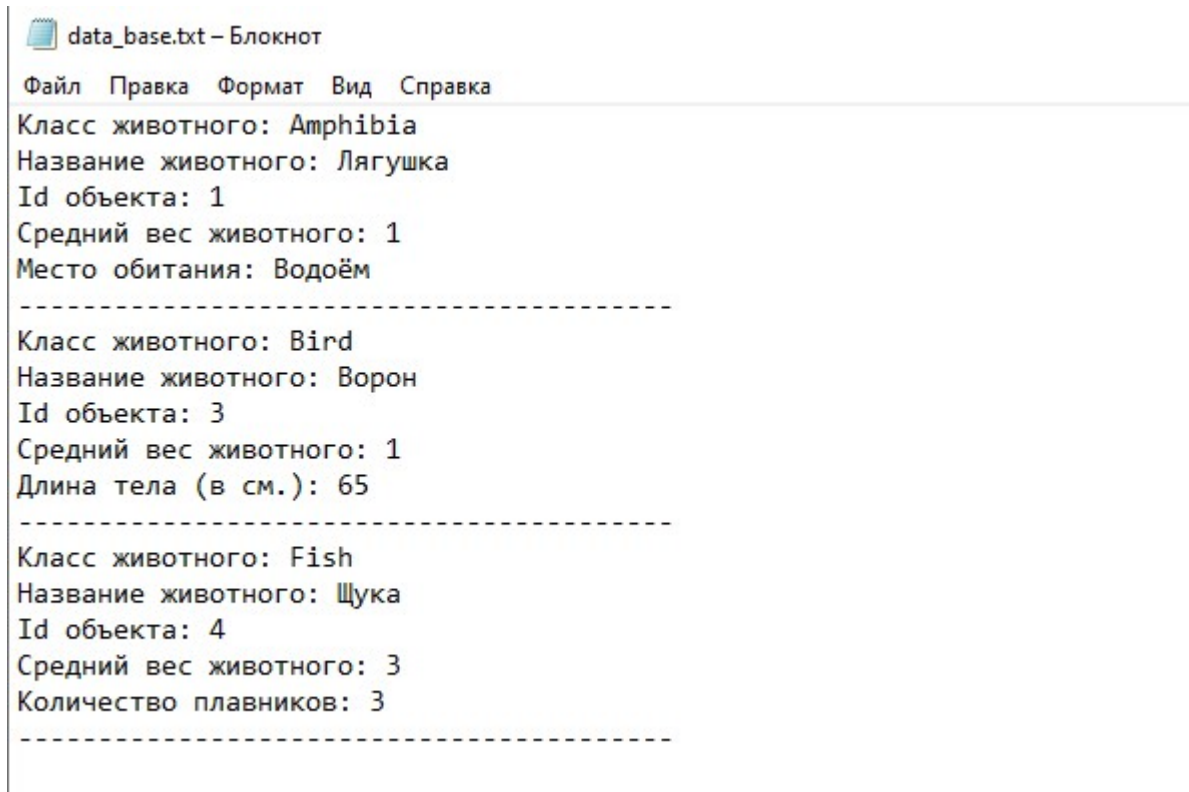


Рис. 12 Добавим новый объект в список





```
data_base.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
Класс животного: Amphibia
Название животного: Лягушка
Id объекта: 1
Средний вес животного: 1
Место обитания: Водоём
-----
Класс животного: Bird
Название животного: Ворон
Id объекта: 3
Средний вес животного: 1
Длина тела (в см.): 65
-----
Класс животного: Fish
Название животного: Щука
Id объекта: 4
Средний вес животного: 3
Количество плавников: 3
-----
```

Рис. 13 Выход из программы - обновление файла data\_base.txt

**Заключение:** мы получили практические навыки создания иерархии классов, использования статических компонентов класса и освоили работу с потоковым вводом/выводом на C++.

## Приложение

### Код программы:

#### Source.cpp

```
#include <iostream>
#include <locale>
#include "windows.h"
#include "Interface.h"
#include "fstream"
int main(int argc, string* argv[]) {
    SetConsoleCP(1251);
    setlocale(LC_ALL, "RUS");
    Interface interf;
    interf.drawInterface();
    return 0;
}
```

#### Animal.cpp

```
#include <iostream>
#include <string>
#include "Animal.h"
#include "fstream"
#include <windows.h>
#include "AnimalList.h"
#include "Amphibia.h"
#include "Bird.h"
#include "Fish.h"
#include "Reptile.h"
#include "Interface.h"

using namespace std;

int Animal::animalCount = 0;
int Animal::currentId = 0;

void Animal::showAnimalData() {
    std::cout << "\n===== \n";
    std::cout << "ID объекта: " << animalId << "\n";
    std::cout << "Название животного: " << animalName << "\n";
    std::cout << "Класс животного: " << animalType << "\n";
    std::cout << "Средний вес животного: " << avgWeight << "\n";
}

int Animal::getAnimalCount() {
    return animalCount;
}

Animal::Animal(string animalType, string animalName, int avgWeight) {
    animalCount++;
    currentId++;
    this->animalType = animalType;
    this->animalName = animalName;
    this->avgWeight = avgWeight;
    animalId = currentId;
    std::cout << this << " Вы создали животное под названием " << animalName << " с ID: " << animalId << "\n";
}

Animal::Animal( string animalType) {
    this->animalType = animalType;
```



```

        animalCount++;
        currentId++;
        animalId = currentId;
        std::cout << "Создание животного с классом: " << animalType << "\n";
        std::cout << "Введите вес животного: ";
        //std::cin.ignore();
        std::cin >> avgWeight;
        std::cout << "Введите название животного: ";
        std::cin >> animalName;
    }

    void Animal::writeToFile()
    {
        std::ofstream out;
        out.open("data_base.txt", ios::app);

        if (out.is_open())
        {
            out << "Класс животного: " << *animalType << std::endl;
            out << "Название животного: " << *animalName << std::endl;
            out << "Id объекта: " << *animalId << std::endl;
            out << "Средний вес животного: " << *avgWeight << std::endl;
        }
        out.close();
    }

    Animal::Animal() : avgWeight(0), animalName(""), animalType("") {
        animalCount++;
        currentId++;
        animalId = currentId;
        std::cout << this << " Вы создали животное без параметров с ID: " << animalId <<
        "\n";
    }

    Animal::Animal(const Animal& animal) {
        animalCount++;
        currentId++;
        this->animalType = animal.animalType;
        this->animalName = animal.animalName;
        this->avgWeight = animal.avgWeight;
        animalId = currentId;
        std::cout << this << " Вы создали животное под названием " << animalName << " с
        ID: " << animalId << " через конструктор копирования\n";
    }

    Animal::~Animal() {
        std::cout << this << " Вы удалили животное: " << animalName << " с ID: " <<
        animalId << "\n";
        animalCount--;
    }

    void Animal::setAnimalType(string type) {
        animalType = type;
    }

    void Animal::setAnimalName(string name) {
        animalName = name;
    }

    void Animal::setAvgWeight(int weight) {
        avgWeight = weight;
    }

    string Animal::getAnimalType() {
        return animalType;
    }

```

```

}

string Animal::getAnimalName() {
    return animalName;
}

int Animal::getAvgWeight() {
    return avgWeight;
}

int Animal::getAnimalId() {
    return animalId;
}

Animal::Animal(std::string TypeFromFile, std::string NameFromFile, int WeightFromFile,
int IdFromFile){
    this->animalType = TypeFromFile;
    animalCount++;
    currentId=IdFromFile;
    this->animalId = IdFromFile;
    this->avgWeight = WeightFromFile;
    this->animalName = NameFromFile;
}

```

## Animal.h

```

#pragma once

#include "string"
#include "fstream"
#include "iostream"

using namespace std;

enum animalTypes{ amph, bird, rept, fish };
class Animal
{
public:
    virtual void showAnimalData();
    static int getAnimalCount();
    Animal(string animalType, string animalName, int avgWeight);
    Animal(string animalType);
    Animal();
    Animal(std::string TypeFromFile, std::string NameFromFile, int WeightFromFile, int
IdFromFile);
    Animal(const Animal& animal);
    virtual ~Animal();
    void setAnimalType(string type);
    void setAnimalName(string name);
    void setAvgWeight(int weight);
    string getAnimalType();
    string getAnimalName();
    virtual void writeToFile();
    int getAvgWeight();
    int getAnimalId();
private:
    string animalName;
    int avgWeight;
    string animalType;
    int animalId = 0;
    static int animalCount;
    static int currentId;
};

```

## Amphibia.h

```
#include "Animal.h"
#include "string"

#pragma once
class Amphibia : public Animal
{
public:
    Amphibia(string name, int avgWeight, string habitat);
    Amphibia();
    Amphibia(std::string InfTxt, std::string NameFromFile, int WeightFromFile, int
    IdFromFile);
    ~Amphibia();
    void showAnimalData();
    void setHabitat(string hab);
    string getHabitat(string hab);
    void writeToFile();
private:
    string habitat;
};
```

## Amphibia.cpp

```
#include "Amphibia.h"
#include "iostream"
#include "string"

using namespace std;

Amphibia::Amphibia(string name, int weight, string habitat) :Animal("Amphibia", name ,
weight) {

}

Amphibia::Amphibia() : Animal("Amphibia") {
    cout << "Место обитания: ";
    cin >> this->habitat;
}

Amphibia::Amphibia(std::string InfTxt, std::string NameFromFile, int WeightFromFile, int
IdFromFile): Animal("Amphibia", NameFromFile, WeightFromFile, IdFromFile) {
    this->habitat = InfTxt;
}

Amphibia::~Amphibia() {

}

void Amphibia::setHabitat(string hab)
{
    habitat = hab;
}

string getHabitat(string hab) {
    return hab;
}

void Amphibia::showAnimalData() {
```

```

        Animal::showAnimalData();
        cout << "Место обитания : "<<this->habitat;
    }

void Amphibia::writeToFile() {
    Animal::writeToFile();
    std::ofstream out;           // поток для записи
    out.open("data_base.txt", ios::app);

    if (out.is_open())
    {
        out << "Место обитания: " << this->habitat << std::endl;
        out << "-----\n";
    }
    out.close();
}

```

## Bird.h

```

#include "Animal.h"
#include "string"

#pragma once
class Bird : public Animal
{
public:
    Bird(string name, int avgWeight, int bodyLength);
    Bird(int InfTxt, std::string NameFromFile, int WeightFromFile, int IdFromFile);
    Bird();
    ~Bird();
    void showAnimalData();
    void setBodyLength(int body);
    int getBodyLength(int body);
    void writeToFile();
private:
    int bodyLength;
};

```

## Bird.cpp

```

#include "Bird.h"
#include "iostream"
#include "string"

using namespace std;

Bird::Bird(string name, int weight, int bodyLength) :Animal("Bird", name, weight) {

}

Bird::Bird(int InfTxt, std::string NameFromFile, int WeightFromFile, int IdFromFile) :
Animal("Bird" , NameFromFile, WeightFromFile, IdFromFile) {
    this->bodyLength = InfTxt;
}

Bird::Bird() : Animal("Bird") {
    cout << "Длина тела (в см.) : ";
    cin >> this->bodyLength;
}

Bird::~Bird() {

```

```

}

void Bird::setBodyLength(int body)
{
    bodyLength = body;
}

int Bird::getBodyLength(int body) {
    return body;
}

void Bird::showAnimalData() {
    Animal::showAnimalData();
    cout << "Длина тела (в см.): " << this->bodyLength;
}

void Bird::writeToFile() {
    Animal::writeToFile();
    std::ofstream out;           // поток для записи
    out.open("data_base.txt", ios::app);

    if (out.is_open())
    {
        out << "Длина тела (в см.): " << this->bodyLength << std::endl;
        out << "-----\n";
    }
    out.close();
}

```

## Reptile.h

```

#include "Animal.h"
#include "string"

#pragma once
class Reptile : public Animal
{
public:
    Reptile(string name, int avgWeight, bool shell);
    Reptile(int InFTxt, std::string NameFromFile, int WeightFromFile, int IdFromFile);
    Reptile();
    ~Reptile();
    void setShell(bool shell);
    void showAnimalData();
    int getShell();
    void writeToFile();
private:
    bool beShell;
};

```

## Reptile.cpp

```

#include "Reptile.h"
#include "iostream"
using namespace std;

Reptile :: Reptile(string name, int avgWeight, bool shell){

};

Reptile::Reptile(): Animal("Reptile") {
    cout << "Наличие панциря(1/0) : ";
}

```

```

        cin >> this->beShell;
    };
    Reptile::~Reptile() {

    };

    Reptile::Reptile(int InfTxt, std::string NameFromFile, int WeightFromFile, int
    IdFromFile) : Animal("Reptile", NameFromFile, WeightFromFile, IdFromFile) {
        this->beShell = InfTxt;
    }

    void Reptile::setShell(bool shell) {
    };

    void Reptile::showAnimalData() {
        Animal::showAnimalData();
        cout << "Наличие панциря(1/0): " << this->beShell;
    };

    int Reptile::getShell() {
        return beShell;
    };

    void Reptile::writeToFile() {
        Animal::writeToFile();
        std::ofstream out;           // поток для записи
        out.open("data_base.txt", ios::app);

        if (out.is_open())
        {
            out << "Наличие панциря(1/0): " << this->beShell << std::endl;
            out << "-----\n";
        }
        out.close();
    }

```

## Fish.h

```

#include "Animal.h"

#pragma once
class Fish : public Animal
{
public:
    Fish(string name, int avgWeight, int fins);
    Fish(int InfTxt, std::string NameFromFile, int WeightFromFile, int IdFromFile);
    Fish();
    ~Fish();
    void setFins(int fins);
    void showAnimalData();
    int getFins();
    void writeToFile();
private:
    int fishFins;
};

```

## Fish.cpp

```

#include "Fish.h"

#include "iostream"
#include "string"

```



```

using namespace std;

Fish::Fish(string name, int weight, int fins) :Animal("Fish", name, weight) {

}

Fish::Fish() : Animal("Fish") {
    cout << "Количество плавников : ";
    cin >> this->fishFins;
}

Fish::~Fish() {

}

Fish::Fish(int InfTxt, std::string NameFromFile, int WeightFromFile, int IdFromFile) :
Animal("Fish", NameFromFile, WeightFromFile, IdFromFile) {
    this->fishFins = InfTxt;
}

void Fish::setFins(int fins)
{
    fishFins = fins;
}

int getfishFins(int body) {
    return body;
}

void Fish::showAnimalData() {
    Animal::showAnimalData();
    cout << "Количество плавников: " << this->fishFins;
}

void Fish::writeToFile() {
    Animal::writeToFile();
    std::ofstream out;           // поток для записи
    out.open("data_base.txt", ios::app);

    if (out.is_open())
    {
        out << "Количество плавников: " << this->fishFins << std::endl;
        out << "-----\n";
    }
    out.close();
}

```

## Mammal.h

```

#include "Animal.h"

using namespace std;

#pragma once
class Mammal : public Animal
{
public:
    Mammal(string name, int avgWeight);
    Mammal();
    ~Mammal();
private:
};

```

## Mammal.cpp

```
#include "Mammal.h"

Mammal::Mammal() : Animal("Mammal") {};
Mammal::Mammal(string name, int avgWeight) {

};
Mammal::~Mammal() {};
```

## Interface.h

```
#pragma once
#include "Amphibia.h"
#include "Bird.h"
#include "Reptile.h"
#include "Fish.h"
#include "Mammal.h"
#include "AnimalList.h"
#include "string"

class Interface {
public:
    void drawInterface();
private:
    AnimalList list;
    void doAction(int choice);
    void processAnimalChoice(int choice);
    void drawAddingElement();
    void deleteElementById();
    void drawShowingElementsByClassName();
    void clearscr();
    void initializeWriteToFile();
};
```

## Interface.cpp

```
#include <typeinfo>
#include <iostream>
//#include <string>
#include <Windows.h>
#include <conio.h>
#include "Interface.h"
#include "Animal.h"

void Interface::drawInterface() {
    while (true) {
        clearscr();
        SetConsoleCP(1251);
        int choice = 0;
        std::cout << "=====\n";
        std::cout << "1) Добавить элемент\n";
        std::cout << "2) Показать данные всех элементов списка\n";
        std::cout << "3) Удалить элемент списка\n";
        std::cout << "4) Удалить все элементы в списке\n";
        std::cout << "5) Вывести данные элементов по типу данных\n";
        std::cout << "6) Выход\n";
```

```

        std::cout << "7) Чтение с файла\n";
        std::cout << "Выберите действие: ";
        std::cin >> choice;
        if (choice == 6) {
            initializeWriteToFile();
            break;
        }
        this->doAction(choice);
    }
}

void Interface::initializeWriteToFile()
{
    std::ofstream file;
    file.open("data_base.txt");
    file.clear();
    file.close();
    list.writeToFile();
}

void Interface::doAction(int choice) {
    clrscr();
    switch (choice)
    {
        case 1: this->drawAddingElement();
            break;
        case 2: list.showAllItems();
            break;
        case 3: this->deleteElementById();
            break;
        case 4: list.deleteAllItems();
            break;
        case 5: this->drawShowingElementsByClassName();
            break;
        case 7: list.readFile();
            break;
        default:
            std::cout << "Нет такого действия!\n";
            break;
    }
    _getch();
}

void Interface::processAnimalChoice(int choice) {
    clrscr();
    switch (choice)
    {
        case 1: list.addItem(new Amphibia());
            break;
        case 2: list.addItem(new Fish());
            break;
        case 3: list.addItem(new Reptile());
            break;
        case 4: list.addItem(new Bird());
            break;
        case 5: list.addItem(new Mammal());
            break;
        default:
            std::cout << "Нет такого объекта!\n";
            break;
    }
}

void Interface::drawAddingElement() {
    int choice = 0;

```

```

        std::cout << "==== Доступные к добавлению элементы =====\n";
        std::cout << "1) Amphibia\n";
        std::cout << "2) Fish\n";
        std::cout << "3) Reptile\n";
        std::cout << "4) Bird\n";
        std::cout << "5) Mammal\n";
        std::cout << "6) Чтение из файла\n";
        std::cout << "Введите номер объекта, который хотите создать: ";
        std::cin >> choice;
        this->processAnimalChoice(choice);
    }

    void Interface::deleteElementById() {
        int animalId = 0;
        std::cout << "Введите ID объекта, чтобы его удалить: ";
        std::cin >> animalId;
        list.deleteItem(animalId);
    }

    void Interface::drawShowingElementsByClassName() {
        clrscr();
        int choice = 0;
        std::cout << "1) Amphibia\n";
        std::cout << "2) Fish\n";
        std::cout << "3) Reptile\n";
        std::cout << "4) Bird\n";
        std::cout << "5) Mammal\n";
        std::cout << "Выберите тот класс, элементы которого будут выведены: \n";
        std::cin >> choice;
        switch (choice)
        {
            case 1: list.showItemsByClassName(typeid(Amphibia));
                    break;
            case 2: list.showItemsByClassName(typeid(Fish));
                    break;
            case 3: list.showItemsByClassName(typeid(Bird));
                    break;
            case 4: list.showItemsByClassName(typeid(Reptile));
                    break;
            case 5: list.showItemsByClassName(typeid(Mammal));
                    break;
            default:
                std::cout << "Нет такого класса!\n";
                break;
        }
    }

    void Interface::clrscr() {
        HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
        COORD coord = { 0, 0 };
        DWORD count;
        CONSOLE_SCREEN_BUFFER_INFO csbi;
        GetConsoleScreenBufferInfo(hStdOut, &csbi);
        FillConsoleOutputCharacter(hStdOut, ' ', csbi.dwSize.X * csbi.dwSize.Y, coord,
        &count);
        SetConsoleCursorPosition(hStdOut, coord);
    }
}

```

## AnimalList.h

```

#pragma once
#include <vector>
#include "Animal.h"
#include "fstream"

```

```

using namespace std;

class AnimalList
{
public:
    void addItem(Animal* animal);
    void showAllItems();
    void showItem(int animalId);
    void deleteAllItems();
    void deleteItem(int animalId);
    void showItemsByClassName(const type_info& type);
    ~AnimalList();
    void writeToFile();
    void readFromFile();
private:
    std::vector<Animal*> animalList;
};

```

## AnimalList.cpp

```

#include <typeinfo>
#include <iostream>
#include <vector>
#include <string>
#include "AnimalList.h"
#include <windows.h>
#include "fstream"
#include "Animal.h"
#include "Interface.h"

void AnimalList::addItem(Animal* animal) {
    animalList.push_back(animal);
}

void AnimalList::showAllItems() {
    if (!animalList.empty()) {
        for (auto element : animalList) {
            element->showAnimalData();
        }
    }
    else {
        std::cout << "\nСписок пуст!\n";
    }
}

void AnimalList::showItem(int AnimalId) {
    if (!animalList.empty()) {
        for (auto element : animalList) {
            if (element->getAnimalId() == AnimalId) {
                element->showAnimalData();
            }
        }
    }
    else {
        std::cout << "Список пуст!\n";
    }
}

void AnimalList::deleteAllItems() {
    if (!animalList.empty()) {
        std::vector<Animal*>::iterator itt;
        for (itt = animalList.begin(); itt != animalList.end(); ) {
            delete* itt;
            itt = animalList.erase(itt);
        }
    }
}

```

```

        }
        std::cout << "\nВсе элементы были удалены из списка!\n";
    }
    else {
        std::cout << "\nВы не можете очистить список, так как он уже пуст!\n";
    }
}

void AnimalList::deleteItem(int AnimalId) {
    if (!animalList.empty()) {
        std::vector<Animal*>::iterator itt;
        for (itt = animalList.begin(); itt != animalList.end(); ) {
            if ((*itt)->getAnimalId() == AnimalId) {
                delete* itt;
                itt = animalList.erase(itt);
            }
            else {
                itt++;
            }
        }
    }
    else {
        std::cout << "\nВы не можете удалить элемент списка, так как он уже
пуст!\n";
    }
}

void AnimalList::showItemsByClassName(const type_info& type) {
    for (auto element : animalList) {
        if (typeid(*element) == type) {
            element->showAnimalData();
        }
    }
}

AnimalList::~AnimalList() {
    if (!animalList.empty()) {
        std::vector<Animal*>::iterator itt;
        for (itt = animalList.begin(); itt != animalList.end(); ) {
            delete* itt;
            itt = animalList.erase(itt);
        }
    }
}

void AnimalList::writeToFile()
{
    std::ofstream out;
    out.open("data_base.txt");
    if (!out.is_open())
    {
        cout << "File don't open!";
        exit(1);
    }
    else
    {
        cout << "File opened!";
        for (Animal* element : animalList) {
            element->writeToFile();
        }
    }
    cout << "Объекты сохранены в "<<"data_base.txt" << endl;
    out.close();
}

```



```

void AnimalList::readFile() {
    std::string line;
    std::string tmpTxt;
    std::ifstream in("data_base.txt"); // открываем файл для чтения
    if (in.is_open())
    {
        while (getline(in, line))
        {
            std::string typeOfAnimal = line.substr(line.find(':') + 2,
line.length() - line.find(':'));
            getline(in, line);
            std::string nameOfAnimal = line.substr(line.find(':') + 2,
line.length() - line.find(':'));
            getline(in, line);
            std::string chkId = line.substr(line.find(':') + 2, line.length() -
line.find(':'));
            int idOfAnimal = stoi(chkId);
            getline(in, line);
            int WeightOfAnimal = stoi(line.substr(line.find(':') + 2,
line.length() - line.find(':')));
            getline(in, line);
            if (typeOfAnimal == "Amphibia") {
                std::string infTxt = line.substr(line.find(':') + 2,
line.length() - line.find(':'));
                addItem(new Amphibia(infTxt, nameOfAnimal, WeightOfAnimal,
idOfAnimal));
            }
            else if (typeOfAnimal == "Bird") {
                std::string infTxt = line.substr(line.find(':') + 2,
line.length() - line.find(':'));
                addItem(new Bird(stoi(infTxt), nameOfAnimal, WeightOfAnimal,
idOfAnimal));
            }
            else if (typeOfAnimal == "Fish") {
                std::string infTxt = line.substr(line.find(':') + 2,
line.length() - line.find(':'));
                addItem(new Fish(stoi(infTxt), nameOfAnimal, WeightOfAnimal,
idOfAnimal));
            }
            else if (typeOfAnimal == "Reptile") {
                std::string infTxt = line.substr(line.find(':') + 2,
line.length() - line.find(':'));
                addItem(new Reptile(stoi(infTxt), nameOfAnimal,
WeightOfAnimal, idOfAnimal));
            }
            else if (typeOfAnimal == "Mammal") {
                addItem(new Animal("Mammal", nameOfAnimal, WeightOfAnimal,
idOfAnimal));
            }
            if (line[1] != '-')getline(in, line);
            cout << "Добавлен элемент с классом " << typeOfAnimal << endl;
        }
    }
    in.close(); // закрываем файл
    cout << "\nДанные записаны из data_base.txt";
}

```