Оглавление

[1 Постановка задачи 3](#_Toc514507645)

[2 Описание работы 5](#_Toc514507646)

[2.1 Функции 5](#_Toc514507647)

[2.2 Классы 9](#_Toc514507648)

[2.3 Блок-схема 12](#_Toc514507649)

[3 Инструкция пользователя 13](#_Toc514507650)

[4 Пример выполнения 19](#_Toc514507651)

[Приложение А. Листинг программы 21](#_Toc514507652)

[areals.cpp 21](#_Toc514507653)

[Region.h 29](#_Toc514507654)

[Region.cpp 29](#_Toc514507655)

[Country.h 30](#_Toc514507656)

[Country.cpp 30](#_Toc514507657)

[Province.h 31](#_Toc514507658)

[Province.cpp 31](#_Toc514507659)

[Animal.h 32](#_Toc514507660)

[Animal.cpp 33](#_Toc514507661)

[Mammal.h 33](#_Toc514507662)

[Mammal.cpp 33](#_Toc514507663)

[Reptile.h 34](#_Toc514507664)

[Reptile.cpp 34](#_Toc514507665)

# 1 Постановка задачи

Необходимо проанализировать предметную область - ареалы обитания животных - и разработать объективно отражающую ей иерархию классов. По На основе составленной иерархии, необходимо разработать программу на языке C++, позволяющую автоматизировать учёт ареалов обитания млекопитающих и пресмыкающихся.

Задачи:

1. Связать отношением наследования иерархию классов в соответствии с вариантом задания (п.1 задания). Класс, стоящий во главе иерархии, определить как абстрактный.
2. Определить в классах конструкторы (включая конструкторы с параметрами и без параметров) и деструкторы. Деструкторы должны выводить информационные сообщения о том, какой объект его вызвал.
3. Определить в классах компоненты-функции для просмотра и установки полей данных.
4. Определить в классах компоненты-функции для выполнения запросов (п.2 задания).
5. Определить в классах статическое поле, предназначенное для хранения количества объектов данного класса.
6. В демонстрационной программе создать иерархию объектов. Объекты создаются динамически по запросу пользователя, при этом указатели на вновь созданные объекты записываются в статические массивы (достаточно размерности n=5…10).
7. Предусмотреть вывод на экран пользовательского меню, предусматривающего создание или удаление элементов (удаление – это удаление объекта посредством операции delete, а также удаление его адреса в статическом массиве посредством смещения значений элементов массива влево на один элемент).Создание объектов должно осуществляться от объектов верхнего уровня к нижнему, в соответствии с иерархией классов.
8. Предусмотреть вывод на экран данных по запросу (п.2 задания), а также количества элементов заданного типа ( из статического поля).

Запросы:

* Вывести перечень животных указанного вида, обитающих в указанной стране;
* Вывести перечень млекопитающих, обитающих в территориальной единице указанного вида;
* Вывести перечень ядовитых пресмыкающихся, обитающих в указанной территориальной единице;
* Вывести количество видов млекопитающих с массой тела больше указанной.

# 2 Описание работы

# 2.1 Функции

**int main();**

Основная функция программы. Выводит на экран пользовательское меню и вызывает выбранные пользователем функции.

**int InputRegionClass()**

Запрашивает у пользователя класс региона (страна или область).

**int InputAnimalClass()**

Запрашивает у пользователя класс животного (млекопитающее или пресмыкающееся).

**template<class T>**

**size\_t InputRegion(const string& title)**

Выводит на экран список территориальных единиц заданного в шаблонном параметре T класса и запрашивает у пользователя индекс интересующего региона.

**template<class T>**

**void ListRegions(bool brief, bool showCount)**

Перечисляет все элементы статического массива с указателями на созданные территориальные единицы и вызывает для каждого экземпляра функцию Display(). Параметр шаблона служит для указания конкретного класса, экземпляры которого интересуют (Страна или Область). Если параметр brief равен true, то выводится сокращенная инормация о территориальной единице. Если параметр showCount равен true, то будет также выведено количество животных (значение статической переменной regionsCount, а также равное ей значение, возвращаемое статической функцией InstanceCount() - членом класса Region).

**template<class T>**

**void ListAnimals(bool showCount)**

Перечисляет все элементы статического массива с указателями на созданных животных и вызывает для каждого экземпляра функцию Display(). Параметр шаблона служит для указания конкретного класса, экземпляры которого интересуют (Млекопитающее или Пресмыкающееся). Если параметр showCount равен true, то будет также выведено количество животных (значение статической переменной animalsCount, а также равное ей значение, возвращаемое статической функцией InstanceCount() - членом класса Animal).

**template<class Filter, class Operation>**

**void ProcessAnimals(Filter filter, Operation op) {**

Перечисляет все элементы статического массива с указателями на созданных животных и вызывает для каждого экземпляра функцию обратного вызова filter(), для проверки его соответствия требуемым условиям. Экземпляры животных, прошедшие проверку, передаются в функцию обратного вызова op() для последующей обработки. Параметры шаблона служат для вывода типов проверяющей и обрабатывающей функций.

**void ListAnimalsInCountry()**

Выводит на экран список животных указанного вида, обитающих в указанной стране.

**void ListMammalsInRegion()**

Выводит на экран список млекопитающих, обитающих в указанной территориальной единице.

**void ListToxicReptiles()**

Выводит на экран список ядовитых пресмыкающихся, обитающих в указанной территориальной единице

**void ListHeavyMammals()**

Выводит на экран список млекопитающих с массой тела больше указанной.

**void RegisterRegion(Region\* region)**

Вспомогательная функция для размещения указателя на созданный экземпляра территориальной единицы в статическом массиве.

**void AddProvince(size\_t countryIndex, const std::string& name)**

Вспомогательная функция-обёртка, создаёт экземпляр «Области» с заданным названием и размещает его в указанной стране.

**void AddRegion()**

Функция добавления территориальной единицы. Запрашивает у пользователя, регион какого класса необходимо создать (Страна или Область), а также название территориальной единицы. В случае добавления области, требуется также указать страну, в которой эта область находится.

**void DeleteRegion()**

Функция удаления территориальной единицы.

**void RegisterAnimal(Animal\* animal)**

Вспомогательная функция для размещения указателя на созданный экземпляра животного в статическом массиве.

**void AddMammal(size\_t regionIndex, const std::string& kind, double weight)**

Вспомогательная функция-обёртка, создаёт экземпляр «Млекопитающего» заданного вида и массы, и размещает его в указанной области.

**void AddReptile(size\_t regionIndex, const std::string& kind, double weight, bool toxic)**

Вспомогательная функция-обёртка, создаёт экземпляр «Пресмыкающегося» заданного вида, массы и степени ядовитости, и размещает его в указанной области.

**void AddAnimal()**

Функция добавления животного. Запрашивает у пользователя ареал обитания животного (необходимо выбрать конкретную область из числа созданных ранее), класс животного (Млекопитающее или Пресмыкающееся), а также конкретный вид (Бурый медведь, китайский аллигатор и т.д.). В случае добавления пресмыкающегося, требуется также указать, является ли оно ядовитым.

**void DeleteAnimal()**

Функция удаления животного.

**void TestData();**

Водит предопределённый набор тестовых данных. Удобная функция для отладки программы и проверки запросов.

# 2.2 Классы

**Region**

Абстрактный базовый класс, представляющий территориальную единицу.

Содержит статическое поле - счетчик экземпляров, строковую переменную для названия территориальной единицы (все они характеризуются, как минимум, названием) и функции чтения/записи этой переменной. Единственный конструктор принимает обязательный параметр, задающий название.

Непосредственное создание его экземпляров запрещается наличием чисто виртуальных функций (как минимум, деструктора).

Все территориальные единицы наследуют виртуальную функцию вывода на экран.

**Country**

Класс, представляющий страну. Наследник класса Region (Страна *является* территориальной единицей).

Содержит два конструктора. Один, с параметром, задаёт пользовательское название страны (соответствующее поле унаследовано от базового класса), второй, конструктор по умолчанию, задаёт пустое название.

Унаследованный метод Display() выводит название на экран с пометкой, что эта территориальная единица является страной.

**Province**

Класс, представляющий область (провинцию). Наследник класса Region (Область *является* территориальной единицей).

Содержит два конструктора. Один, с параметром, задаёт пользовательское название области (соответствующее поле унаследовано от базового класса), второй, конструктор по умолчанию, задаёт пустое название.

Унаследованный метод Display() выводит название на экран с пометкой, что эта территориальная единица является областью.

Класс содержит дополнительное поле country, являющееся указателем на объект страны, в которой размещается данная область. Для установки и чтения этого поля существуют функции доступа ParentCountry().

**Animal**

Абстрактный базовый класс, представляющий животное.

Содержит статическое поле - счетчик экземпляров, строковую переменную для хранения вида животного, переменную типа double для хранения массы тела и указатель на объект области - ареала обитания. Также содержит функции чтения/записи этих переменных. Единственный конструктор принимает два обязательных параметра, задающий название вида и массу тела.

Непосредственное создание его экземпляров запрещается наличием чисто виртуальных функций (как минимум, деструктора).

Все классы животных наследуют виртуальную функцию вывода на экран.

**Mammal**

Класс, представляющий млекопитающее. Наследник класса Animal (Млекопитающее *является* животным).

Содержит два конструктора. Один принимает в качестве обязательных параметров название вида и массу тела (соответствующие поля унаследованы от базового класса), второй, конструктор по умолчанию, инициализирует поля пустыми значениями.

Унаследованный метод Display() выводит название на экран с пометкой, что это животное является млекопитающим.

**Reptile**

Класс, представляющий пресмыкающееся. Наследник класса Animal (Млекопитающее *является* животным).

Содержит два конструктора. Один принимает в качестве обязательных параметров название вида, массу тела (соответствующие поля унаследованы от базового класса) и признак ядовитости. Конструктор по умолчанию инициализирует поля пустыми значениями.

Унаследованный метод Display() выводит название на экран с пометкой, что это животное является пресмыкающимся.

# 2.3 Блок-схема

Блок-схема главного меню программы представлена на рисунке 1.

Начало

Цикл А.

Ввод menu

нет

да

menu == 1

ListRegions

нет

да

menu == 2

ListAnimals

нет

да

menu == 3

ListAnimalsInCountrty

нет

да

menu == 4

ListMammalsInRegion

нет

да

menu == 5

ListToxicRepriles

нет

да

menu == 6

ListHeavyMammals

нет

да

menu == 7

AddRegion

нет

да

menu == 8

DeleteRegion

нет

да

menu == 9

AddAnimal

да

menu == 10

DeleteAnimal

Цикл А.

menu != 0

Конец

Рисунок 1. Блок схема

# 3 Инструкция пользователя

Вид главного меню программы представлен на рисунке 2.

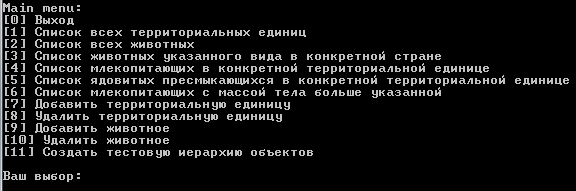


Рисунок 2. Главное меню программы

Для завершения программы служит специальный пункт меню под номером ноль.

1. Список всех территориальных единиц. Этот пункт выводит на экран общее количество и перечень всех территориальных единиц, которые были добавлены после запуска программы. Пример выполнения этого пункта представлен на рисунке 3.

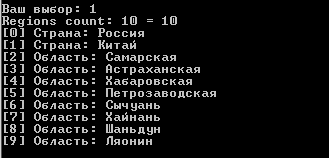


Рисунок 3. Перечень добавленных территориальных единиц

2. Список всех животных. Этот пункт выводит на экран общее количество и перечень всех животных, которые были добавлены после запуска программы. . Пример выполнения этого пункта представлен на рисунке 4.

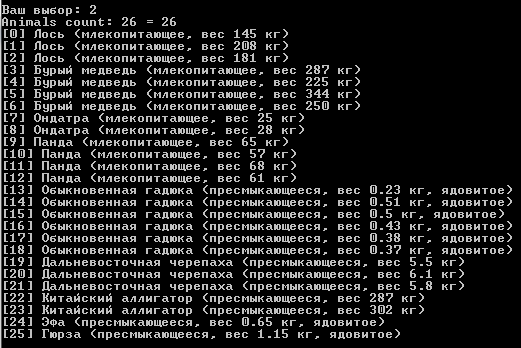


Рисунок 4. Перечень добавленных животных

3. Список животных указанного вида в конкретной стране. Этот пункт реализует первый запрос задания и выводит перечень животных указанного вида, обитающих в указанной стране. Пример вывода на экран изображен на рисунке 5.

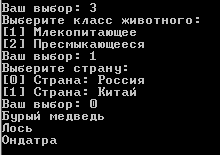


Рисунок 5. Перечень животных заданного вида в указанной стране

4. Список млекопитающих, обитающих в конкретной территориальной единице. Этот пункт реализует второй запрос задания и выводит перечень млекопитающих, обитающих в указанной территориальной единице. Пример выполнения этого пункта представлен на рисунке 6.

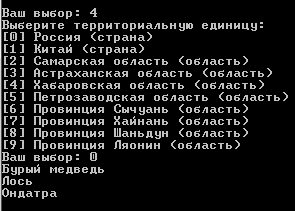


Рисунок 6. Перечень млекопитающих в указанном регионе

5. Список ядовитых пресмыкающихся, обитающих в конкретной территориальной единице. Этот пункт реализует третий запрос задания и выводит перечень ядовитых пресмыкающихся, обитающих в указанной территориальной единице. Пример вывода на экран изображен на рисунке 7.

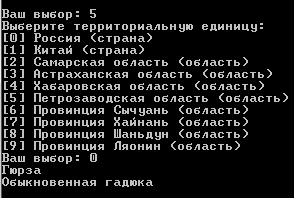


Рисунок 7. Перечень ядовитых пресмыкающихся в указанном регионе

6. Список млекопитающих с массой тела больше указанной. Этот пункт реализует четвёртый запрос задания и выводит перечень млекопитающих с массой тела больше указанной. Пример вывода на экран изображен на рисунке 8.

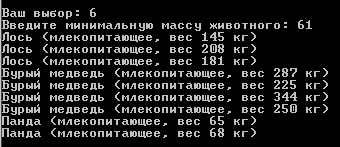


Рисунок 8. Перечень млекопитающих с массой тела больше указанной

7. Добавление новой территориальной единицы. Запрашивает у пользователя, регион какого класса необходимо создать (Страна или Область), а также название. В случае добавления области, требуется также указать страну, в которой эта область находится. Пример выполнения этого пункта представлен на рисунке 9.

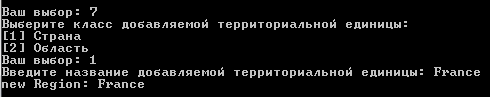


Рисунок 9. Пример добавления территориальной единицы

8. Удаление территориальной единицы. На экран для удобства будет выведен перечень существующих единиц и нужно будет выбрать подходящий пункт списка. Пример выполнения этого пункта представлен на рисунке 10.

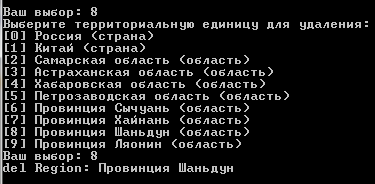


Рисунок 10. Пример удаления территориальной единицы

9. Добавление нового животного. Запрашивает у пользователя ареал обитания животного (необходимо выбрать конкретную область из числа созданных ранее), класс животного (Млекопитающее или Пресмыкающееся), а также конкретный вид (Лось, Амурский тигр и т.д.). В случае добавления пресмыкающегося, требуется также указать, является ли оно ядовитым. Пример вывода на экран изображен на рисунке 11.

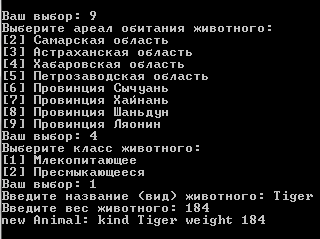


Рисунок 11. Пример добавления животного

10. Удаление животного. На экран для удобства будет выведен перечень существующих животных и нужно будет выбрать интересующий пункт списка. Пример выполнения этого пункта представлен на рисунке 12.

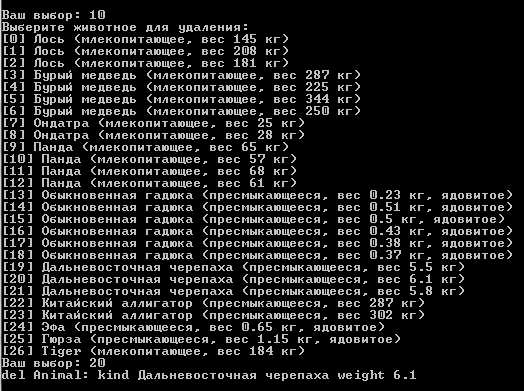


Рисунок 12. Пример удаления животного

11. Для удобства отладки и тестирования служит последний пункт меню, «Создать тестовую иерархию объектов». Этот пункт избавляет от необходимости каждый раз вручную вводить большой объём данных. Вместо этого он заполняет базу данных программы заранее предопределённым набором данных.

Сперва добавляются две страны - Россия и Китай. В Россию добавляются Самарская, Астраханская, Хабаровская и Петрозаводская области. В Китай добавляются провинции Сычуань, Хайнань, Шаньдун и Ляонин. Затем по этим областям распределяются 26 животных - 13 млекопитающих и столько же пресмыкающихся.

Все операции добавления сопровождаются соответствующим выводом на экран консоли, вид которой изображен на рисунке 13.

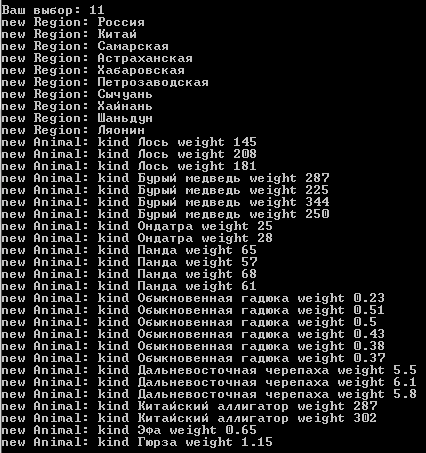


Рисунок 13. Журнал заполнения программы тестовыми данными.

После заполнения памяти программы тестовыми данными можно приступать к изучению результатов запросов.

# 4 Пример выполнения

Одним из сценариев работы с демонстрационной программой может быть следующий.

В первую очередь заполняем память программы тестовыми данными (пункт 11 главного меню). Экран консоли должен выглядеть так, как представлено на рисунке 13.

После этого можно выполнять запросы по существующим данным.

Для начала запросим список пресмыкающихся в России. Результат, полученный на основе тестовых данных, приведён на рисунке 14.

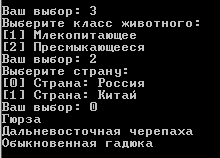


Рисунок 14. Список пресмыкающихся в России

Далее поинтересуемся списком млекопитающих в Астраханской области. Результат запроса изображен на рисунке 15.

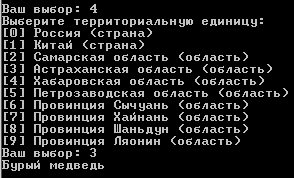


Рисунок 15. Список млекопитающих в Астраханской области.

Следующий запрос будет на получения списка млекопитающих с массой тела больше 200кг. Список самых тяжелых тестовых животных приведён на рисунке 16.

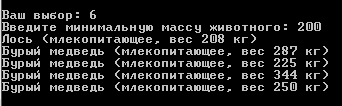


Рисунок 16. Перечень млекопитающих с массой тела больше 200кг.

Завершающим запросом выведем на экран список ядовитых пресмыкающихся в провинции Хайнань. Результат выполнения запроса приведён на рисунке 17.

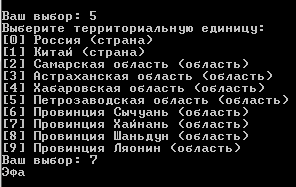


Рисунок 17. Список ядовитых пресмыкающихся в провинции Хайнань.

Перед завершением сеанса проверим пункт 9 и удалим животное. Для удаления выберем Дальневосточную черепаху весом 6.1кг. Результат выполнения запроса на удаление изображен на рисунке 12.

# Приложение А. Листинг программы

# areals.cpp

#include <iostream>

#include <set>

#include "Country.h"

#include "Province.h"

#include "Reptile.h"

#include "Mammal.h"

using namespace std;

static const size\_t MAX\_REGIONS\_COUNT = 20;

static Region\* regions[MAX\_REGIONS\_COUNT] = { 0 };

static size\_t regionsCount = 0;

static const size\_t MAX\_ANIMALS\_COUNT = 100;

static Animal\* animals[MAX\_ANIMALS\_COUNT] = { 0 };

static size\_t animalsCount = 0;

typedef set< string > AnimalKinds;

int InputRegionClass() {

int result = 0;

cout << "Выберите класс добавляемой территориальной единицы:" << endl;

cout << "[1] Страна" << endl;

cout << "[2] Область" << endl;

cout << "Ваш выбор: ";

cin >> result;

if (result != 1 && result != 2) {

cerr << "Ошибка: введено недопустимое значение" << endl;

result = -1;

}

return result;

}

int InputAnimalClass() {

int result = 0;

cout << "Выберите класс животного:" << endl;

cout << "[1] Млекопитающее" << endl;

cout << "[2] Пресмыкающееся" << endl;

cout << "Ваш выбор: ";

cin >> result;

if (result != 1 && result != 2) {

cerr << "Ошибка: введено недопустимое значение" << endl;

result = -1;

}

return result;

}

template<class T>

size\_t InputRegion(const string& title) {

size\_t result = 0;

cout << title << endl;

ListRegions<T>(false);

cout << "Ваш выбор: ";

cin >> result;

if (result >= regionsCount) {

cerr << "Ошибка: введено недопустимое значение" << endl;

result = regionsCount;

}

return result;

}

template<class T>

void ListRegions(bool brief, bool showCount = false) {

if (showCount)

cout << "Regions count: " << regionsCount << " = " << Region::InstanceCount() << endl;

for (size\_t i = 0; i < regionsCount; i++) {

if (dynamic\_cast< T\* >(regions[i])) {

cout << "[" << i << "] ";

regions[i]->Display(brief);

}

}

}

template<class T>

void ListAnimals(bool showCount = false) {

if (showCount)

cout << "Animals count: " << animalsCount << " = " << Animal::InstanceCount() << endl;

for (size\_t i = 0; i < animalsCount; i++) {

if (dynamic\_cast< T\* >(animals[i])) {

cout << "[" << i << "] ";

animals[i]->Display();

}

}

}

template<class Filter, class Operation>

void ProcessAnimals(Filter filter, Operation op) {

for (size\_t i = 0; i < animalsCount; i++) {

if (filter(animals[i])) {

op(animals[i]);

}

}

}

// Список животных указанного вида в конкретной стране

void ListAnimalsInCountry() {

AnimalKinds kinds;

int animalClass = InputAnimalClass();

if (animalClass < 0) {

return;

}

size\_t index = InputRegion<Country>("Выберите страну:");

if (index == regionsCount)

return;

ProcessAnimals(

// функция проверки каждого животного

[animalClass, index](const Animal\* animal) {

return animal->Home()->ParentCountry() == regions[index] &&

(animalClass == 1 ?

dynamic\_cast< const Mammal\* >(animal) != nullptr :

dynamic\_cast< const Reptile\* >(animal) != nullptr);

},

// функция обработки каждого животного, прошедшего проверку

[&kinds](const Animal\* animal) {

kinds.insert(animal->Kind());

}

);

for (const auto& kind : kinds) {

cout << kind << endl;

}

}

// Список млекопитающих в конкретной территориальной единице

void ListMammalsInRegion() {

AnimalKinds kinds;

size\_t index = InputRegion<Region>("Выберите территориальную единицу:");

if (index == regionsCount)

return;

ProcessAnimals(

// функция проверки каждого животного

[index](const Animal\* animal) {

return dynamic\_cast< const Mammal\* >(animal) != nullptr &&

(dynamic\_cast< Country\* >(regions[index]) ?

animal->Home()->ParentCountry() == regions[index] :

animal->Home() == regions[index]);

},

// функция обработки каждого животного, прошедшего проверку

[&kinds](const Animal\* animal) {

kinds.insert(animal->Kind());

}

);

for (const auto& kind : kinds) {

cout << kind << endl;

}

}

// Список ядовитых пресмыкающихся в конкретной территориальной единице

void ListToxicReptiles() {

AnimalKinds kinds;

size\_t index = InputRegion<Region>("Выберите территориальную единицу:");

if (index == regionsCount)

return;

ProcessAnimals(

// функция проверки каждого животного

[index](const Animal\* animal) {

return dynamic\_cast< const Reptile\* >(animal) != nullptr &&

dynamic\_cast< const Reptile\* >(animal)->Toxic() &&

(dynamic\_cast< Country\* >(regions[index]) ?

animal->Home()->ParentCountry() == regions[index] :

animal->Home() == regions[index]);

},

// функция обработки каждого животного, прошедшего проверку

[&kinds](const Animal\* animal) {

kinds.insert(animal->Kind());

}

);

for (const auto& kind : kinds) {

cout << kind << endl;

}

}

// Список млекопитающих с массой тела больше указанной

void ListHeavyMammals() {

double weight = 0;

cout << "Введите минимальную массу животного: ";

cin >> weight;

ProcessAnimals(

// функция проверки каждого животного

[weight](const Animal\* animal) {

return dynamic\_cast< const Mammal\* >(animal) != nullptr && animal->Weight() > weight;

},

// функция обработки каждого животного, прошедшего проверку

[](const Animal\* animal) {

animal->Display();

}

);

}

void RegisterRegion(Region\* region) {

if (region) {

regions[regionsCount++] = region;

}

}

void AddProvince(size\_t countryIndex, const std::string& name) {

Province\* province = new Province(name);

province->ParentCountry(dynamic\_cast< Country\* >(regions[countryIndex]));

RegisterRegion(province);

}

void AddRegion() {

string regionName;

if (regionsCount == MAX\_REGIONS\_COUNT) {

cerr << "Достигнуто предельное количество экземпляров: " << regionsCount << endl;

return;

}

int regionClass = InputRegionClass();

if (regionClass < 0) {

return;

}

Region\* region = nullptr;

cout << "Введите название добавляемой территориальной единицы: ";

cin >> regionName;

size\_t index = 0;

switch (regionClass) {

case 1:

region = new Country(regionName);

break;

case 2:

index = InputRegion<Country>("Выберите страну, которой принадлежит добавляемая область:");

if (index == regionsCount)

return;

Country\* country = dynamic\_cast< Country\* >(regions[index]);

if (country) {

Province\* province = new Province(regionName);

province->ParentCountry(country);

region = province;

}

break;

}

RegisterRegion(region);

}

void DeleteRegion() {

size\_t index = 0;

if (regionsCount == 0) {

cout << "Список территориальных единиц пуст, нечего удалять" << endl;

return;

}

else if (regionsCount > 1) {

index = InputRegion<Region>("Выберите территориальную единицу для удаления:");

if (index == regionsCount) {

return;

}

} // В противном случае, есть только одна территориальная единица для удаления

Region\* region = regions[index];

regionsCount--;

for (size\_t i = index; i < regionsCount; i++) {

regions[i] = regions[i + 1];

}

delete region;

}

void RegisterAnimal(Animal\* animal) {

if (animal) {

animals[animalsCount++] = animal;

}

}

void AddMammal(size\_t regionIndex, const std::string& kind, double weight) {

Animal\* animal = new Mammal(kind, weight);

animal->Home(dynamic\_cast< Province\* >(regions[regionIndex]));

RegisterAnimal(animal);

}

void AddReptile(size\_t regionIndex, const std::string& kind, double weight, bool toxic) {

Animal\* animal = new Reptile(kind, weight, toxic);

animal->Home(dynamic\_cast< Province\* >(regions[regionIndex]));

RegisterAnimal(animal);

}

void AddAnimal() {

string animalKind;

double weight = 0.0;

bool toxic = false;

Animal\* animal = nullptr;

Province\* province = nullptr;

if (animalsCount == MAX\_ANIMALS\_COUNT) {

cerr << "Достигнуто предельное количество экземпляров: " << animalsCount << endl;

return;

}

size\_t index = 0;

cout << "Выберите ареал обитания животного:" << endl;

ListRegions<Province>(true);

cout << "Ваш выбор: ";

cin >> index;

if (index >= regionsCount) {

cerr << "Ошибка: введено недопустимое значение" << endl;

return;

}

province = dynamic\_cast< Province\* >(regions[index]);

int animalClass = InputAnimalClass();

if (animalClass < 0)

return;

cout << "Введите название (вид) животного: ";

cin >> animalKind;

cout << "Введите вес животного: ";

cin >> weight;

switch (animalClass) {

case 1:

animal = new Mammal(animalKind, weight);

break;

case 2:

cout << "Является ли рептилия ядовитой (1/0)?: ";

cin >> toxic;

cout << "toxic: " << toxic << endl;

animal = new Reptile(animalKind, weight, toxic);

break;

}

animal->Home(province);

RegisterAnimal(animal);

}

void DeleteAnimal() {

size\_t index = 0;

if (animalsCount == 0) {

cout << "Список животных пуст, нечего удалять" << endl;

return;

}

else if (animalsCount > 1) {

cout << "Выберите животное для удаления:" << endl;

ListAnimals<Animal>();

cout << "Ваш выбор: ";

cin >> index;

} // otherwise, there is only one animal (with index [0]) to delete

if (index < animalsCount) {

Animal\* animal = animals[index];

animalsCount--;

for (size\_t i = index; i < animalsCount; i++) {

animals[i] = animals[i + 1];

}

/\*cout << "Region::dtor(" << animal << "):";

for (size\_t i = 0; i < animalsCount; i++)

cout << " " << animals[i];

cout << endl;\*/

delete animal;

}

}

void TestData();

int main()

{

setlocale(LC\_ALL, "RUS");

int counter = 0;

int menu = -1;

while (menu != 0) {

cout << endl << endl << "Main menu:" << endl;

cout << "[0] Выход" << endl;

cout << "[1] Список всех территориальных единиц" << endl;

cout << "[2] Список всех животных" << endl;

cout << "[3] Список животных указанного вида в конкретной стране" << endl;

cout << "[4] Список млекопитающих в конкретной территориальной единице" << endl;

cout << "[5] Список ядовитых пресмыкающихся в конкретной территориальной единице" << endl;

cout << "[6] Список млекопитающих с массой тела больше указанной" << endl;

cout << "[7] Добавить территориальную единицу" << endl;

cout << "[8] Удалить территориальную единицу" << endl;

cout << "[9] Добавить животное" << endl;

cout << "[10] Удалить животное" << endl;

cout << "[11] Создать тестовую иерархию объектов" << endl;

cout << endl;

cout << "Ваш выбор: ";

cin.sync();

cin >> menu;

switch (menu) {

case 1:

ListRegions<Region>(false, true);

break;

case 2:

ListAnimals<Animal>(true);

break;

case 3:

ListAnimalsInCountry();

break;

case 4:

ListMammalsInRegion();

break;

case 5:

ListToxicReptiles();

break;

case 6:

ListHeavyMammals();

break;

case 7:

AddRegion();

break;

case 8:

DeleteRegion();

break;

case 9:

AddAnimal();

break;

case 10:

DeleteAnimal();

break;

case 11:

TestData();

break;

}

// защита от непредвиденных ошибок ввода

if (counter++ == 10) {

cout << "emergency exit" << endl;

break;

}

//break; // for debug

}

for (size\_t i = 0; i < regionsCount; i++) {

delete regions[i];

}

for (size\_t i = 0; i < animalsCount; i++) {

delete animals[i];

}

cout << "Работа завершена" << endl;

return 0;

}

// Создать тестовую иерархию объектов

void TestData() {

RegisterRegion(new Country("Россия")); // [0]

RegisterRegion(new Country("Китай")); // [1]

AddProvince(0, "Самарская область"); // [2]

AddProvince(0, "Астраханская область"); // [3]

AddProvince(0, "Хабаровская область"); // [4]

AddProvince(0, "Петрозаводская область"); // [5]

AddProvince(1, "Провинция Сычуань"); // [6]

AddProvince(1, "Провинция Хайнань"); // [7]

AddProvince(1, "Провинция Шаньдун"); // [8]

AddProvince(1, "Провинция Ляонин"); // [9]

AddMammal(2, "Лось", 145.0); // [0]

AddMammal(4, "Лось", 208.0); // [1]

AddMammal(6, "Лось", 181.0); // [2]

AddMammal(2, "Бурый медведь", 287.0); // [3]

AddMammal(3, "Бурый медведь", 225.0); // [4]

AddMammal(4, "Бурый медведь", 344.0); // [5]

AddMammal(5, "Бурый медведь", 250.0); // [6]

AddMammal(2, "Ондатра", 25.0); // [7]

AddMammal(8, "Ондатра", 28.0); // [8]

AddMammal(6, "Панда", 65.0); // [9]

AddMammal(7, "Панда", 57.0); // [10]

AddMammal(8, "Панда", 68.0); // [11]

AddMammal(9, "Панда", 61.0); // [12]

AddReptile(2, "Обыкновенная гадюка", 0.23, true); // [13]

AddReptile(3, "Обыкновенная гадюка", 0.51, true); // [14]

AddReptile(4, "Обыкновенная гадюка", 0.5, true); // [15]

AddReptile(5, "Обыкновенная гадюка", 0.43, true); // [16]

AddReptile(6, "Обыкновенная гадюка", 0.38, true); // [17]

AddReptile(9, "Обыкновенная гадюка", 0.37, true); // [18]

AddReptile(4, "Дальневосточная черепаха", 5.5, false); // [19]

AddReptile(6, "Дальневосточная черепаха", 6.1, false); // [20]

AddReptile(7, "Дальневосточная черепаха", 5.8, false); // [21]

AddReptile(8, "Китайский аллигатор", 287.0, false); // [22]

AddReptile(9, "Китайский аллигатор", 302.0, false); // [23]

AddReptile(7, "Эфа", 0.65, true); // [24]

AddReptile(3, "Гюрза", 1.15, true); // [25]

}

# Region.h

#ifndef REGION\_H\_

#define REGION\_H\_

// Регион (территориальная единица)

#include <string>

class Region {

public:

Region(const std::string& name);

virtual ~Region() = 0; // запрещаем создавать экземпляры абстрактного класса

inline void Name(const std::string& value) {

name = value;

}

inline std::string Name() const {

return name;

}

static size\_t InstanceCount();

virtual void Display(bool brief) const = 0;

protected:

// название

std::string name;

private:

static size\_t instCount;

};

#endif /\* REGION\_H\_ \*/

# Region.cpp

#include "Region.h"

#include <iostream>

using namespace std;

size\_t Region::instCount = 0;

Region::Region(const std::string& name) : name(name) {

cout << "new Region: " << name << endl;

instCount++;

}

// пустая заглушка для чисто виртуального деструктора

Region::~Region() {

cout << "del Region: " << name << endl;

instCount--;

}

//static

size\_t Region::InstanceCount() {

return instCount;

}

# Country.h

#ifndef COUNTRY\_H\_

#define COUNTRY\_H\_

// Страна

#include "Region.h"

class Country: public Region {

public:

Country(const std::string& name);

virtual ~Country();

// override: наследуемый метод обязан быть объявленным в базовом классе

void Display(bool brief) const override;

};

#endif /\* COUNTRY\_H\_ \*/

# Country.cpp

#include <iostream>

#include "Country.h"

#include "Province.h"

using namespace std;

Country::Country(const std::string& name) : Region(name) {

}

Country::~Country() {

}

void Country::Display(bool brief) const {

cout << name;

if (!brief)

cout << " (страна)";

cout << endl;

}

# Province.h

#ifndef PROVINCE\_H\_

#define PROVINCE\_H\_

// Область

#include "Region.h"

class Country;

class Province: public Region {

public:

Province(const std::string& name);

virtual ~Province();

// override: наследуемый метод обязан быть объявленным в базовом классе

void Display(bool brief) const override;

void ParentCountry(Country\* parent) {

country = parent;

}

Country\* ParentCountry() {

return country;

}

private:

// hosting country

Country\* country = nullptr;

};

#endif /\* PROVINCE\_H\_ \*/

# Province.cpp

#include <iostream>

#include "Province.h"

using namespace std;

Province::Province(const std::string& name) : Region(name) {

}

Province::~Province() {

}

void Province::Display(bool brief) const {

cout << name;

if (!brief)

cout << " (область)";

cout << endl;

}

# Animal.h

#ifndef ANIMAL\_H\_

#define ANIMAL\_H\_

// Животное

#include <string>

class Province;

class Animal {

public:

Animal(const std::string& kind, double weight);

virtual ~Animal() = 0; // запрещаем создавать экземпляры абстрактного класса

inline std::string Kind() const {

return kind;

}

inline void Weight(double value) {

weight = value;

}

inline double Weight() const {

return weight;

}

inline void Home(Province\* value) {

home = value;

}

inline Province\* Home() const {

return home;

}

virtual void Display() const = 0;

static size\_t InstanceCount();

protected:

// вид

std::string kind;

// вес

double weight;

//

Province\* home;

private:

static size\_t instCount;

};

#endif /\* ANIMAL\_H\_ \*/

# Animal.cpp

#include "Animal.h"

#include "Province.h"

#include <iostream>

using namespace std;

size\_t Animal::instCount = 0;

Animal::Animal(const std::string& kind, double weight) : kind(kind), weight(weight) {

cout << "new Animal: kind " << kind << " weight " << weight << endl;

instCount++;

}

// пустая заглушка для чисто виртуального деструктора

Animal::~Animal() {

cout << "del Animal: kind " << kind << " weight " << weight << endl;

instCount--;

}

//static

size\_t Animal::InstanceCount() {

return instCount;

}

# Mammal.h

#ifndef MAMMAL\_H\_

#define MAMMAL\_H\_

// Млекопитающее

#include "Animal.h"

class Mammal: public Animal {

public:

Mammal();

Mammal(const std::string& kind, double weight);

virtual ~Mammal();

// override: наследуемый метод обязан быть объявленным в базовом классе

void Display() const override;

};

#endif /\* MAMMAL\_H\_ \*/

# Mammal.cpp

#include "Mammal.h"

#include <iostream>

using namespace std;

Mammal::Mammal() : Animal("mammal", 0.0) {

}

Mammal::Mammal(const std::string& kind, double weight) : Animal(kind, weight) {

}

Mammal::~Mammal() {

}

void Mammal::Display() const {

cout << kind << " (млекопитающее, вес " << weight << " кг)" << endl;

}

# Reptile.h

#ifndef REPTILE\_H\_

#define REPTILE\_H\_

// Пресмыкающееся

#include "Animal.h"

class Reptile: public Animal {

public:

Reptile();

Reptile(const std::string& kind, double weight, bool toxic);

virtual ~Reptile();

inline void Toxic(bool value) {

toxic = value;

}

inline bool Toxic() const {

return toxic;

}

// override: наследуемый метод обязан быть объявленным в базовом классе

void Display() const override;

private:

bool toxic;

};

#endif /\* REPTILE\_H\_ \*/

# Reptile.cpp

#include "Reptile.h"

#include <iostream>

using namespace std;

Reptile::Reptile() : Animal("reptile", weight), toxic(false) {

}

Reptile::Reptile(const std::string& kind, double weight, bool toxic) : Animal(kind, weight), toxic(toxic) {

}

Reptile::~Reptile() {

}

void Reptile::Display() const {

cout << kind << " (пресмыкающееся, вес " << weight << " кг";

if(toxic)

cout << ", ядовитое";

cout << ")" << endl;

}