**САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО**

**ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И КИБЕРБЕЗОПАСНОСТИ**

**ВЫСШАЯ ШКОЛА ПРОГРАММНОЙ ИНЖЕНЕРИИ**

**Лабораторная работа №9**

**«ООП. Виртуальные методы»**

**по предмету «Алгоритмизация и программирование»**

**Выполнил: студент гр. 5130904/30002 Севостьянова А.В.**

**Руководитель: Череповский Д.К.**

**Санкт-Петербург**

**2023 г.**

Оглавление

[Задание 1 3](#_Toc152882477)

[1. Общая постановка задачи 3](#_Toc152882478)

[2. Детальные требования и тест план 4](#_Toc152882479)

[a. Максимальное количество питомцев должно быть задано корректно(переменная имеет целочисленное значение) 4](#_Toc152882480)

[b. Реальное количество питомцев не должно быть больше их максимального количества. 4](#_Toc152882481)

[Таблица с детальными требованиями и тест планом 4](#_Toc152882482)

[3. Программа 5](#_Toc152882483)

[main.cpp 5](#_Toc152882484)

[Pets.h 6](#_Toc152882485)

[Pets.cpp 7](#_Toc152882486)

[Задание 2 10](#_Toc152882487)

[Приложение 11](#_Toc152882488)

[Вывод 13](#_Toc152882489)

# Задание 1

1. **Общая постановка задачи**

Создать абстрактный класс *Pet*, имеющий два чисто виртуальных метода voice и name

Создать класс *Cat*, производный от Pet, в котором определены виртуальные методы voice и name

Создать класс *Dog*, производный от Pet, в котором определены виртуальные методы voice и name

*Создать:*

* объект класса Cat (cat) и указатель на этот объект (pCat);
* объект класса Dog (dog) и указатель на этот объект (pDog);
* указатель на класс Pet (pPet).

*Вызвать виртуальные методы voice и name:*

* для объектов cat и dog;
* через указатели pCat и pDog;
* через указатель pPet, загружая в него адрес объекта cat;
* через указатель pPet, загружая в него адрес объекта dog;

*Создать класс FamilyPets, содержащий*

* *закрытые данные*– максимальное и реальное количество питомцев, массив ссылок на питомцев (массив указателей на объекты классов Dog и Cat).
* *открытые:*
* конструктор, создающий «пустую» семью
* метод voice, «прослушивание» голосов всех членов FamilyPets
* метод name, вывод имен всех членов FamilyPets
* перегруженную операцию + для добавления нового члена FamilyPets
* дружественную функцию:
* проверяющую есть ли член FamilyPets с указанным именем и определяющую, *к какому классу он относится (может быть несколько одинаковых имен)*
* перегружающую операцию >> для вывода информации о всех членах FamilyPets

Создать функцию main для тестирования разработанных классов

1. **Детальные требования и тест план**
2. ***Максимальное количество питомцев должно быть задано корректно(переменная имеет целочисленное значение)***

Иначе, сообщение «Некорректный ввод», аварийное завершение программы.

1. ***Реальное количество питомцев не должно быть больше их максимального количества.***

Иначе при попытке добавления нового объекта, сообщение «Достигнуто максимальное значение, объект не может быть добавлен», аварийное завершение программы.

### *Таблица с детальными требованиями и тест планом*

|  |  |  |  |
| --- | --- | --- | --- |
| Требования | Детальные требования | Данные | Ожидаемый результат |
| 1. ***Максимальное количество питомцев должно быть задано корректно*** | | | |
| *1. Переменная задана некорректно* (нечисловое значение, дробное значение, отрицательное значение) | Если переменная задана некорректно, то сообщение:  «Некорректный ввод», аварийное завершение программы. | Tyghrufh  5.5  -6 | Сообщение:  «Некорректный ввод»  Аварийное завершение программы. |
| *2.* *Переменная задана корректно* | Если переменная задана корректно, то продолжение работы программы | 5 | Продолжение работы программы |
| 1. ***Реальное количество питомцев не должно быть больше их максимального количества*** | | | |
| *1. Реальное количество питомцев больше максимального* | Если при добавлении очередного объекта в массив его размер превышает максимально возможный, то сообщение:  «Достигнуто максимальное значение, объект не может быть добавлен», аварийное завершение программы. | 4(>3) | Сообщение:  «Достигнуто максимальное значение, объект не может быть добавлен»  Аварийное завершение программы. |
| 2. Реальное количество питомцев не превышает максимального | Если при добавлении очередного объекта в массив его размер не превышает максимально возможный, то продолжение работы программы. | 4(<5) | Продолжение работы программы |

1. **Программа**

### *main.cpp*

#include"Pets.h"

int main()

{

setlocale(LC\_ALL , "ru");

Cat cat;

std::cout << "Объект класса: " << cat.name() << " Голос: " << cat.voice()<<"\n";

Dog dog;

std::cout << "Объект класса: " << dog.name() << " Голос: " << dog.voice() << "\n";

Cat\* pCat = &cat;

std::cout << "Указатель на объект класса Cat\n Класс: " << pCat->name() << " Голос: " << pCat->voice() << "\n";

Dog\* pDog = &dog;

std::cout << "Указатель на объект класса Dog\n Класс: " << pDog->name() << " Голос: " << pDog->voice() << "\n";

Pet\* pPet;

pPet = &cat;

std::cout << "Указатель на объект класса Pet\n Класс: " << pPet->name() << " Голос: " << pPet->voice() << "\n";

pPet = &dog;

std::cout << "Указатель на объект класса Pet\n Класс: " << pPet->name() << " Голос: " << pPet->voice() << "\n";

testClasses();

return 0;

}

### *Pets.h*

#pragma once

#include<iostream>

#include<regex>

#include<vector>

const std::string ERROR\_OVERFLOW = "Достигнуто максимальное значение,объект не может быть добавлен\n";

const std::string ERROR\_OF\_INPUT = "Некорректный ввод\n";

class Pet

{

friend std::istream& operator >> (std::istream& stream, Pet &pet);

public:

virtual std::string voice()const = 0;

virtual std::string name()const = 0;

std::string getName()const;

protected:

std::string name\_;

};

class Cat :public Pet

{

public:

Cat();

Cat(const std::string& name);

~Cat();

std::string voice()const override;

std::string name()const override;

};

class Dog :public Pet

{

public:

Dog();

Dog(const std::string& name);

~Dog();

std::string voice()const override;

std::string name()const override;

};

class FamilyPets

{

friend void isPetInFamily(FamilyPets &pets,std::string& name);

friend std::ostream& operator << (std::ostream& stream, FamilyPets& pets);

public:

FamilyPets();

FamilyPets(int countMax);

~FamilyPets();

void voice()const;

void name()const;

void operator + (Pet \*pet);

private:

int count\_, countMax\_;

std::vector <Pet\*> arr\_;

};

void testClasses();

int checkInput(std::string& count);

### *Pets.cpp*

#include "Pets.h"

std::string Pet::getName()const

{

return name\_;

}

// класс Cat

Cat::Cat() { name\_ = "Кличка отсутсвует"; }

Cat::Cat(const std::string& name) { name\_ = name; }

Cat::~Cat(){}

std::string Cat::voice()const{ return "\*ЗВУКИ КОТА\*";}

std::string Cat::name()const { return "CAT"; }

//Класс Dog

Dog::Dog() { name\_ = "Кличка отсутствует"; }

Dog::Dog(const std::string& name) { name\_ = name; }

Dog::~Dog(){}

std::string Dog::voice()const { return "\*ЗВУКИ СОБАКИ\*"; }

std::string Dog::name()const { return "DOG"; }

//Класс FamilyPets

FamilyPets::FamilyPets() { count\_ = 0; countMax\_ = 0; }

FamilyPets::FamilyPets(int countMax) { count\_ = arr\_.size(); countMax\_ = countMax; }

FamilyPets::~FamilyPets(){}

void FamilyPets::voice()const

{

for (Pet \*pet : arr\_) std::cout<<pet->voice()<<"\n";

}

void FamilyPets::name()const

{

for (Pet\* pet : arr\_) std::cout<<pet->name()<<"\n";

}

void FamilyPets::operator+(Pet \*pet)

{

if (count\_ < countMax\_)

{

arr\_.push\_back(pet);

count\_++;

}

else throw std::runtime\_error(ERROR\_OVERFLOW);

}

std::istream& operator>>(std::istream& stream, Pet& pet)

{

stream >> pet.name\_;

return stream;

}

//Дружественные функции для класса FamilyPets

void isPetInFamily(FamilyPets& pets, std::string& name)

{

bool petInFamily = false;

for (Pet\* pet : pets.arr\_)

{

if (pet->getName() == name)

{

std::cout << pet->name() << "\n";

petInFamily = true;

}

}

if (!petInFamily) std::cout << "Объект с таким именем отсутствует\n";

}

std::ostream& operator<<(std::ostream& stream, FamilyPets& pets)

{

if (pets.arr\_.size() == 0) stream << "Массив пуст\n";

else

{

for (Pet\* pet : pets.arr\_)

stream << "Класс: " << pet->name() << " Имя: " << pet->getName() << " Голос: " << pet->voice() << "\n";

}

return stream;

}

int checkInput(std::string& count)

{

std::regex regular("\\d+,?0\*");

std::regex replace("\\.");

count = std::regex\_replace(count, replace, ",");

if (std::regex\_match(count, regular)) return std::stoi(count);

else throw std::invalid\_argument(ERROR\_OF\_INPUT);

}

//Тестовая функция

void testClasses()

{

try

{

std::cout << "\n\n------------------------------------------------\n";

std::cout << "Введите максимальное количество объектов: ";

std::string value;

std::cin >> value;

FamilyPets emptyPets;

std::cout << "emptyPets: " << emptyPets;

FamilyPets pets (checkInput(value));

Cat cat1;

std::cout << "Введите кличку кота: ";

std::cin >> cat1;

pets + &cat1;

Cat cat2;

pets + &cat2;

Dog dog1;

std::cout << "Введите кличку собаки: ";

std::cin >> dog1;

pets + &dog1;

Dog dog2;

pets + &dog2;

std::cout << "\nТипы объектов в pets:\n";

pets.name();

std::cout << "\nГолоса объектов в pets:\n";

pets.voice();

std::cout<<"\n\nemptyPets:\n" << pets;

std::cout << "\n\nПроверка наличия. Введите кличку животного: ";

std::cin >> value;

isPetInFamily(pets, value);

}

catch (std::runtime\_error& ex)

{

std::cout << ex.what();

exit(EXIT\_FAILURE);

}

catch (std::invalid\_argument& ex)

{

std::cout << ex.what();

exit(EXIT\_FAILURE);

}

}

# Задание 2

*1. Данный код выведет ошибку компиляции.*

Класс A имеет конструктор с модификатором доступа private, поэтому объекты данного типа не могут быть созданы вне класса A.

2. Будет выведено A() A() B(), так как конструкторы обоих классов имеют модификатор доступа public, поэтому могут быть созданы вне классов.

3. *2 конструктора с параметрами для объектов ob1 и ob2:* A(р)\_ A(р)\_

*Конструктор копирования для объекта ob3*: A(соnst А&)\_

*Конструктор копирования для создания параметров функции:* A(соnst А&)\_ A(соnst А&)\_

*Деструктор для параметров функции:* ~A()\_ ~A()\_

*Деструктор для ob1, ob2,ob3:* ~A()\_ ~A()\_ ~A()\_

4. *2 конструктора с параметрами для объектов ob1 и ob2:* A(р)\_ A(р)\_

*Конструктор копирования для объекта ob3*: A(соnst А&)\_

*Деструктор для ob1, ob2,ob3:* ~A()\_ ~A()\_ ~A()\_

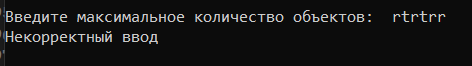
5. Для перегрузки унарной операции требуется *1 параметр*, над которым будет выполняться данная операция

6. *Нет, нельзя*. При перегрузке оператора изменяется только поведение оператора.

7. Нет, *не верно*, количество аргументов, необходимых для перегруженной операции, зависит от логики программы.

8. При передаче в функцию объект может изменять свое значение при передаче *по ссылке* или при передаче *указателя на данный объект*.

# Приложение

****

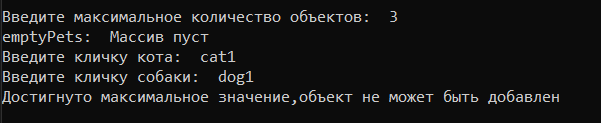
*Рис. 1 Максимальное количество объектов имеет нечисловое значение*

****

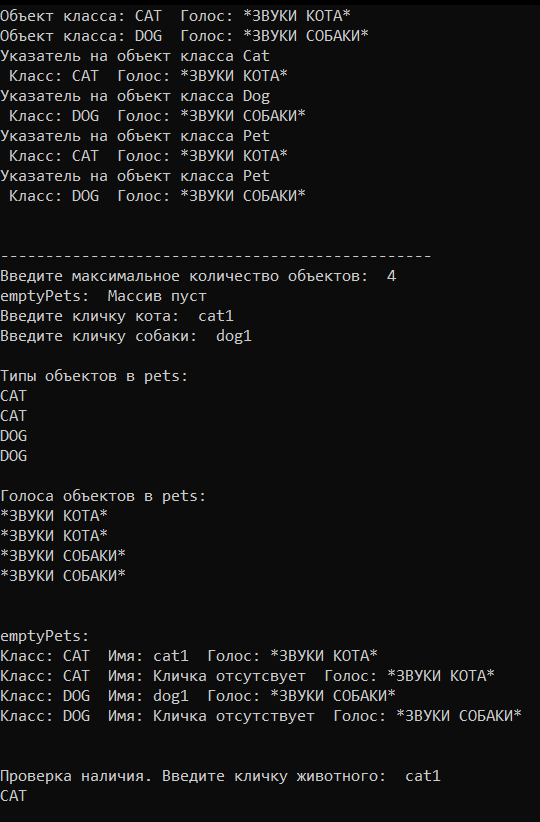
*Рис. 2Максимальное количество объектов имеет дробное значение*



*Рис. 3 Максимальное количество объектов принимает отрицательное значение*

****

*Рис. 4 Реальное количество объектов больше максимально возможного*

****

*Рис. 5 Корректная работа программы*

# Вывод

В ходе работы:

1. Были изучены абстрактные класса и реализован абстрактный класс Pet
2. Были изучены виртуальные методы и реализованы виртуальные методы name и voice в классе Pet
3. Было изучено наследование и реализованы классы Cat и Dog, производные от Pet
4. Был изучен полиморфизм, благодаря которому было переопределено поведение методов name и voice в производных классах.
5. Были перегружены операторы + и <<
6. Были реализованы дружественные функции
7. Была использована конструкция try … catch, которая позволяет эффективнее обрабатывать возникающие в ходе работы программы ошибки