КОДЫ

#include <iostream>

using namespace std;

int main()

{

double a, b, c; //указали, что будем юзать три переменные типа double (34.5)

cout << "Enter a:";

cin >> a; //ввод с клавиатуры

cout << "Enter b:";

cin >> b;

cout << "Enter c:";

cin >> c;

//задаем условие

if ((a + b > c) && ((a + c) > b) && (b + c > a))

//ни одна сторона тр-ка не должна быть больше, чем сумма двух других сторон

{

cout << "Triangle is bro\n";

}

else cout << "Triangle is not bro\n";

return 0;

}

#include <iostream>

using namespace std;

int main()

{

for (int i = 0; i < 10; i++)

{

cout << i;

}

}

#include <iostream>

using namespace std;

int square(int a, int b) //a,b - formal

{

return a\* b;

}

int main()

{

cout << square(4, 5); //4,5 - actual

}

#include <iostream>

using namespace std;

class Human // Класс - пользовательский тип данных, состоящий из полей и методов

{

    int age = 30;         // Поле класса - это какие-либо данные, некая

    string name = "Ivan"; // информация, характеризующая данный класс

    void birthday() {  // Метод класса - функция, которая совершает

        age++;         // те или иные действия над полями своего класса

        cout << "Happy birthday, " << name << "!\n";

    }

};

int main()

{

    Human h; // Объект - конкретный экземпляр класса, по сути

    // являющийся переменной пользовательского типа данных (класса)

}

// Конструктор - специальный метод класса, который используется

// при создании объекта для корректной инициализации его полей

class Human

{

public:

    Human() {               // Конструктор по умолчанию, который

        age = 0;            // не принимает параметры

        name = "default";

    }

    Human(int age\_, string name\_) { // Конструктор от двух параметров

        age = age\_;     // Полям класса присваиваются значения параметров,

        name = name\_;   // передаваемых при вызове конструктора

    }

    Human(const Human& h) { // Копирующий конструктор позволяет создать

        age = h.age;        // полностью идентичный объект, получая входным

        name = h.name;      // параметром константную ссылку (&) на объект

    }                       // того же класса. Копирующий конструктор нужен

private:                    // если в классе динамически выделяется память

    int age;

    string name;

};

// Деструктор - метод, который вызывается при удалении объекта

class Human

{

public:

    ~Human() {      // Деструктор обязательно прописывать вручную

        delete age; // когда при удалении объекта нужно освобождать

    }               // ресурсы, например, если динамически

private:            // выделялась память, или велась работа с файлами

    int\* age = new int(0);

};

class Human

{

public:

    void set\_age(int age\_) { // Сеттеры - функции, которые позволяют присваивать

        if (age\_ > 0) {      // значения закрытым переменным класса

            age = age\_;

        }

    }

    int get\_age() { // Геттеры - функции, которые возвращают значения

        return age; // закрытых переменных классов

    }

private:

    int age;

};

// Безопаснее использовать сеттеры и геттеры, чем предоставлять доступ

// напрямую к полям класса, рискуя дать возможность повредить данные

// this - указатель на объект, владеющий функцией

class Human

{

public:

    Human() { cout << "!\n"; }

    Human& birthday() { // Через this-> можно обращаться

        this->age++;   // к полям и методам класса

        cout << this->age << " years old\n";

        return \*this;  // С помощью this можно возвращать

    }                  // текущий объект класса

    ~Human() { cout << "~\n"; }

private:

    int age = 0;

};

// Инкапсуляция - процесс скрытого хранения деталей реализации объекта

class Human

{

public:                      // public - открывает доступ к полям и

    void set\_age(int age\_) { // методам класса, через любой объект класса

        if (age\_ > 0) {      // можно изменить поля или вызвать методы

            age = age\_;

        }

    }

    int get\_age() {

        return age;

    }

private:                     // private - защищает поля и методы класса,

    int age;                 // они становятся доступны только внутри

    string name = "";        // определения класса, к ним нельзя

};                           // обратиться, используя объекты класса

class Dog {

    friend class Human; // Теперь класс Human имеет доступ

private:                // к private-членам класса Dog

    void speak() {

        cout << "Woof!\n";

    }

};

class Human {

public:

    void dog\_speak(Dog& d) {

        cout << "Speak!\n";

        d.speak();              // Вызов private-метода класса Dog

        cout << "Good boy!\n";  // внутри описания класса Human

    }

};

int main() {

    Human h;

    Dog d;

    h.dog\_speak(d); // Вызов private-метода из экземпляра дружественного класса

}

// Статические члены класса относятся ко всему классу

// сразу, а не к отдельным объектам этого класса

class Human {

public:

    static int population; // Объявление статической переменной

    Human() {

        population++;

    }

    static void new\_born() { // Статический метод может использовать

        population++;        // только статические переменные и не

    }                        // может использовать указатель this

};

int Human::population = 0; // Статические переменные должны быть

// дополнительно определены вне определения класса

int main() {

    Human h1, h2, h3;

    h1.new\_born();

    cout << Human::population; // Вывод значения статической переменной

}

1. Создаём класс Human, применяя все полученные знания:

#include <iostream>

using namespace std;

class Dog { // Объявляем класс Dog

public:

    friend class Human; // Делаем класс Dog дружественным классу Human

    Dog() {             // Прописываем конструктор по умолчанию

        cout << "Dog default Constructor\n";

    }

    ~Dog() {            // Прописываем деструктор

        cout << "Dog Destructor\n";

    }

private:

    void speak() {      // Прописываем приватный метод

        cout << "Woof!\n";

    }

};

class Human { // Объявляем класс Human

public:

    Human() { // Конструктор по умолчанию

        age = 0;

        name = "default";

        population++;

        cout << "Human default Constructor\n";

    }

    Human(int age\_, string name\_) { // Конструктор от двух параметров

        age = age\_;

        name = name\_;

        population++;

        cout << "Human age and name Constructor\n";

    }

    Human(const Human& h) { // Конструктор копирования

        age = h.age;

        name = h.name;

        population++;

        cout << "Human copy Constructor\n";

    }

    ~Human() { // Деструктор

        cout << "Human " << name << " Destructor\n";

        population--;

    }

    void set\_age(int age\_) { // Сеттер для возраста

        if (age\_ > 0) {

            age = age\_;

        }

    }

    void set\_name(string name\_) { // Сеттер для имени

        name = name\_;

    }

    int get\_age() { // Геттер для возраста

        return age;

    }

    string get\_name() { // Геттер для имени

        return name;

    }

    static int get\_population() { // Статический метод доступа к статической переменной

        return population;

    }

    void birthday() { // Прописываем методы класса

        this->age++;

        cout << "Happy birthday, " << this->name << "!\n";

    }

    void dog\_speak(Dog& d) {

        cout << "Speak!\n";

        d.speak();

        cout << "Good boy!\n";

    }

private:

    static int population; // Статическая переменная класса

    int age;

    string name;

};

int Human::population = 0; // Обязательное объявление статической переменной вне класса

int main()

{

    Human h1;               // Создаём объекты - экземпляры класса,

    Human h2(30, "Ivan");   // используя разные конструкторы

    Human h3 = h2;

    cout << "Total population is " << Human::get\_population() << endl; // Вызываем статический метод

    h1.birthday();          // Вызываем методы для объектов

    h2.birthday();

    h1.set\_age(35);

    h1.set\_name("Alexey");

    h1.birthday();

    cout << h1.get\_name() << " is " << h1.get\_age() << " years old!\n";

    cout << h2.get\_name() << " is " << h2.get\_age() << " years old!\n";

    cout << h3.get\_name() << " is " << h3.get\_age() << " years old!\n";

    Dog d;                  // Создаём объект класса Dog

    h1.dog\_speak(d);        // Вызываем метод, имеющий доступ к приватным полям класса Dog

}

#include <iostream>

using namespace std;

class Human // Класс - пользовательский тип данных, состоящий из полей и методов

{

    int age = 30;         // Поле класса - это какие-либо данные, некая

    string name = "Ivan"; // информация, характеризующая данный класс

    void birthday() {  // Метод класса - функция, которая совершает

        age++;         // те или иные действия над полями своего класса

        cout << "Happy birthday, " << name << "!\n";

    }

};

int main()

{

    Human h; // Объект - конкретный экземпляр класса, по сути

    // являющийся переменной пользовательского типа данных (класса)

}

\_\_\_\_\_\_\_\_\_\_\_\_

// Конструктор - специальный метод класса, который используется

// при создании объекта для корректной инициализации его полей

class Human

{

public:

    Human() {               // Конструктор по умолчанию, который

        age = 0;            // не принимает параметры

        name = "default";

    }

    Human(int age\_, string name\_) { // Конструктор от двух параметров

        age = age\_;     // Полям класса присваиваются значения параметров,

        name = name\_;   // передаваемых при вызове конструктора

    }

    Human(const Human& h) { // Копирующий конструктор позволяет создать

        age = h.age;        // полностью идентичный объект, получая входным

        name = h.name;      // параметром константную ссылку (&) на объект

    }                       // того же класса. Копирующий конструктор нужен

private:                    // если в классе динамически выделяется память

    int age;

    string name;

};

\_\_\_\_\_\_\_\_\_\_\_

// Деструктор - метод, который вызывается при удалении объекта

class Human

{

public:

    ~Human() {      // Деструктор обязательно прописывать вручную

        delete age; // когда при удалении объекта нужно освобождать

    }               // ресурсы, например, если динамически

private:            // выделялась память, или велась работа с файлами

    int\* age = new int(0);

};

\_\_\_\_\_\_\_\_\_\_\_\_\_

class Human

{

public:

    void set\_age(int age\_) { // Сеттеры - функции, которые позволяют присваивать

        if (age\_ > 0) {      // значения закрытым переменным класса

            age = age\_;

        }

    }

    int get\_age() { // Геттеры - функции, которые возвращают значения

        return age; // закрытых переменных классов

    }

private:

    int age;

};

// Безопаснее использовать сеттеры и геттеры, чем предоставлять доступ

// напрямую к полям класса, рискуя дать возможность повредить данные

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// this - указатель на объект, владеющий функцией

class Human

{

public:

    Human() { cout << "!\n"; }

    Human& birthday() { // Через this-> можно обращаться

        this->age++;   // к полям и методам класса

        cout << this->age << " years old\n";

        return \*this;  // С помощью this можно возвращать

    }                  // текущий объект класса

    ~Human() { cout << "~\n"; }

private:

    int age = 0;

};

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// Инкапсуляция - процесс скрытого хранения деталей реализации объекта

class Human

{

public:                      // public - открывает доступ к полям и

    void set\_age(int age\_) { // методам класса, через любой объект класса

        if (age\_ > 0) {      // можно изменить поля или вызвать методы

            age = age\_;

        }

    }

    int get\_age() {

        return age;

    }

private:                     // private - защищает поля и методы класса,

    int age;                 // они становятся доступны только внутри

    string name = "";        // определения класса, к ним нельзя

};                           // обратиться, используя объекты класса

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class Dog {

    friend class Human; // Теперь класс Human имеет доступ

private:                // к private-членам класса Dog

    void speak() {

        cout << "Woof!\n";

    }

};

class Human {

public:

    void dog\_speak(Dog& d) {

        cout << "Speak!\n";

        d.speak();              // Вызов private-метода класса Dog

        cout << "Good boy!\n";  // внутри описания класса Human

    }

};

int main() {

    Human h;

    Dog d;

    h.dog\_speak(d); // Вызов private-метода из экземпляра дружественного класса

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// Статические члены класса относятся ко всему классу

// сразу, а не к отдельным объектам этого класса

class Human {

public:

    static int population; // Объявление статической переменной

    Human() {

        population++;

    }

    static void new\_born() { // Статический метод может использовать

        population++;        // только статические переменные и не

    }                        // может использовать указатель this

};

int Human::population = 0; // Статические переменные должны быть

// дополнительно определены вне определения класса

int main() {

    Human h1, h2, h3;

    h1.new\_born();

    cout << Human::population; // Вывод значения статической переменной

}

1. Создаём класс Human, применяя все полученные знания:

#include <iostream>

using namespace std;

class Dog { // Объявляем класс Dog

public:

    friend class Human; // Делаем класс Dog дружественным классу Human

    Dog() {             // Прописываем конструктор по умолчанию

        cout << "Dog default Constructor\n";

    }

    ~Dog() {            // Прописываем деструктор

        cout << "Dog Destructor\n";

    }

private:

    void speak() {      // Прописываем приватный метод

        cout << "Woof!\n";

    }

};

class Human { // Объявляем класс Human

public:

    Human() { // Конструктор по умолчанию

        age = 0;

        name = "default";

        population++;

        cout << "Human default Constructor\n";

    }

    Human(int age\_, string name\_) { // Конструктор от двух параметров

        age = age\_;

        name = name\_;

        population++;

        cout << "Human age and name Constructor\n";

    }

    Human(const Human& h) { // Конструктор копирования

        age = h.age;

        name = h.name;

        population++;

        cout << "Human copy Constructor\n";

    }

    ~Human() { // Деструктор

        cout << "Human " << name << " Destructor\n";

        population--;

    }

    void set\_age(int age\_) { // Сеттер для возраста

        if (age\_ > 0) {

            age = age\_;

        }

    }

    void set\_name(string name\_) { // Сеттер для имени

        name = name\_;

    }

    int get\_age() { // Геттер для возраста

        return age;

    }

    string get\_name() { // Геттер для имени

        return name;

    }

    static int get\_population() { // Статический метод доступа к статической переменной

        return population;

    }

    void birthday() { // Прописываем методы класса

        this->age++;

        cout << "Happy birthday, " << this->name << "!\n";

    }

    void dog\_speak(Dog& d) {

        cout << "Speak!\n";

        d.speak();

        cout << "Good boy!\n";

    }

private:

    static int population; // Статическая переменная класса

    int age;

    string name;

};

int Human::population = 0; // Обязательное объявление статической переменной вне класса

int main()

{

    Human h1;               // Создаём объекты - экземпляры класса,

    Human h2(30, "Ivan");   // используя разные конструкторы

    Human h3 = h2;

    cout << "Total population is " << Human::get\_population() << endl; // Вызываем статический метод

    h1.birthday();          // Вызываем методы для объектов

    h2.birthday();

    h1.set\_age(35);

    h1.set\_name("Alexey");

    h1.birthday();

    cout << h1.get\_name() << " is " << h1.get\_age() << " years old!\n";

    cout << h2.get\_name() << " is " << h2.get\_age() << " years old!\n";

    cout << h3.get\_name() << " is " << h3.get\_age() << " years old!\n";

    Dog d;                  // Создаём объект класса Dog

    h1.dog\_speak(d);        // Вызываем метод, имеющий доступ к приватным полям класса Dog

}