C++

```
#include <iostream>   (исправлено с ~~conio.h~~)
int main() {
std::cout << "Hello world" << std::endl;
}
```

std::endl — перенос строки

удобнее, чем в C тк не надо указывать формат

поток вывода / pipe

<< - отправляем на поток

чтение перем.:

```
int a = 0;
std::cin >> a;
```

std - namespace std { int a; }

(область видимости

standart обл. видимости

вместо того чтобы везде писать std ~~нужно~~ можно наверху (перед main) написать

use namespace std;

Работа со строками

```
#include <string.h>
string str = "Hello world!";
```

```
string str2 = "abc";
str = str2;
str2 = str;
str.length();
```

↑ доступ к всяким ф-ям через .

Объектно ориент. прогр.

Классы

class - тип данных, кот. мы создаем

object - переменная для типа данных class

~~class student~~ ← поля кот. он содержит

```
class User {
public:
    int age;
private:
    int exp;
public: void User();
    void increase_age();
private:
    void increase_exp(int points);
};
```

3 обл. видимости
public
private
protected

конструктор класса

- метод класса (ф-я класса)

```
void main () {
User Bob;

void User :: User () {
   age = 18;
   exp = 0;
  }
void User :: increase_age () {
      age ++;
     increase_exp (100);
     }
void User :: increase_exp (int point) {
      exp += points;
  }
```

```
void main () {
  User Bob;
Bob. increase_age ();
std:: cout << Bob. age << std :: endl; }
```

наследование:

```
class User_with_name : public User{
public:
    string name;
};
```

Задание: написать текст. квест Кв.пр. на кот надо дать ответы и в итоге вывести результат

① чтение пров. об. с клавиат.

② выбор из вар (номер вв с клав) вар в слуг пор.

③ для вопросов использовать класс

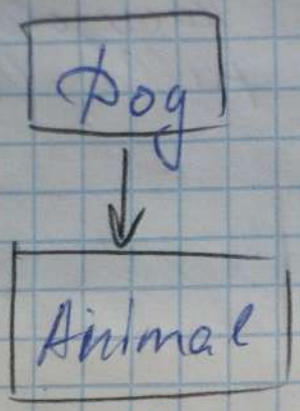
для опис. полу. использ. клас.

исп. string

29.11.

Наследование в ООП.

```
class Animal{
public:
string name;
public:
    Animal(string new_name = "");
    name(new_name);
    sound(); };
class Dog : pablic Animal{ →
```

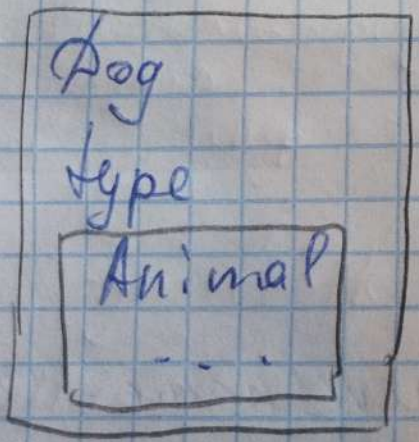


public:
    string type;

```
public:
    Dog (string new_name = "-");
    Sound ();
};

int main () {
    Dog Bobik ("Bobik");
}
```

создается объект Bobik класса Dog



При создании объектов неявно вызывается конструктор родителя

Если есть наследование, то создавать конструктор без аргументов

Но если хотим вызвать конкретный конструктор, то...

```
class Dog : public Animal {
public:
    string type;
public:
    Dog(string new_name="n"): Animal(new_name){};
    sound();
};
```

Ограничение доступа

public - доступна снаружи класса

protected - доступна снаружи

private - доступна только внутри класса

Наследование:

```
class Dog : public Animal {
```

public => public

protected => protected

private => ~~private~~ недоступен

} изменение доступа при наследовании

Если хотим наслед. private

```
class Dog: private Animal {
```

public => private

protected => private

private => недоступен

Наследование protected

public ⇒ protected

protected ⇒ protected

private ⇒ недоступен

Можно в наследнике переписать уровень доступа

private переписать нельзя

1. Инкапсуляция - св-во, позволяющее объединить в класс и данные, и методы, работающие с ними и скрыть детали реализации от пользователя

2. Наследование - св-во, позволяющее создать новый класс-потомок на основе уже существующего, при этом все характеристики класса родителя "присваиваются" классу-потомку

3. Полиморфизм - св-во классов, позволяющее использовать объекты классов с одинак интерфейсом без инф о типе и внутр. структуре объекта

```
Dog bobik ("bobic");
Animal creature ("хэз");
creature = bobik; // в перем. creature
                  скопируется часть перем. bobik
                  кот. относится к Animal
```

Dog bobik("bobik");
Animal *creature01 = &bobic;

копирование не происх.
Animal &creature02 = bobik;

bobik.sound(); // вызовется sound() из Dog
creature01->sound(); -//- из Animal
creature02->sound(); -//- из Animal

void sound_th_times(Animal &creature) {
creature.sound();
creature.sound();
creature.sound();

Можно представить указ-ль Animal как указатель на Dog
объект класса

(dynamic_cast<Dog*>(creature)).sound();
вызовется sound() из Dog

final - запрещ. в дальнейшем его наследовать класс и переопределять ф-цию

virtual sound() override final;

class Dog final: public Animal {...}

Абстрактные классы

Идея интерфейса: это данные/методы/переменные в виде кода, но без реализации методов.

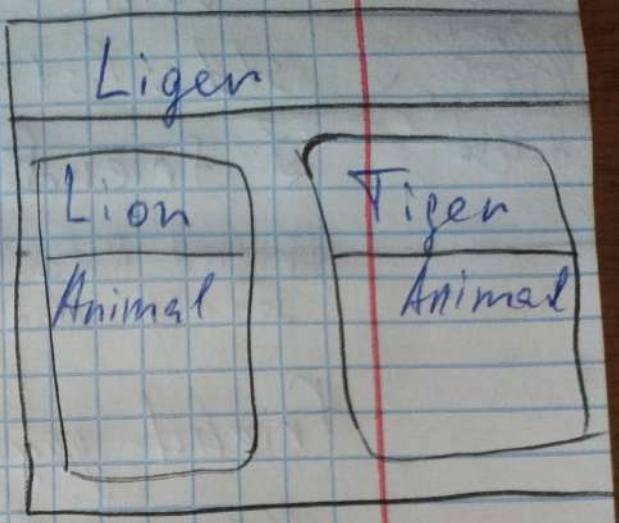
```
class Animal{
public: private:
    string name;
    int age;
public:
    string get_name(){};
virtual void sound()=0; // абстрактная ф-я
```

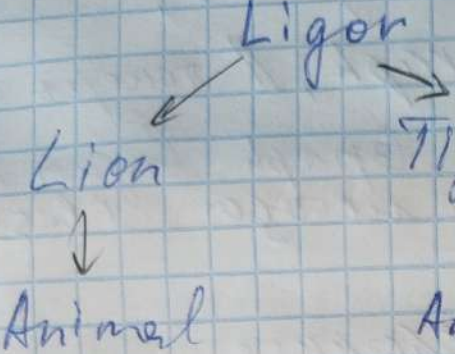
Абстрактный класс - класс у кот хотябы 1 абстрактная ф-я

Множественное наследование.

```
class Animal{ ...
...
};
class Tiger : public Animal{
    int tail_length;
};
class Lion: public Animal{
    int tail_length;
};
```

class Liger : public Lion, public Tiger {
};





Problems:

① Два экземпляра Animal внутри класса

② Два одинак поле tail_length;

Нужно делать без неопределенностей т.е. не должно быть одинак полей

Дружественные функции - и классы

class [redacted] Graph {

private:

Node * root;

public:

Node * Search (Node * node);

};

```
class Node {
private:
    void * data;
    std::list < Node* > neighbors;
    friend class Graph; // теперь ну
friend Node Graph::
    search (Node* node); } ( Graph видит поля
                             и методы class Node
```

friend - дружба односторонняя

Можно сделать дружественным метод

Анонимные объекты

```
Dog ("bobik"); // создание анонимного объекта
Dog bobik = Dog ("bobik");
Dog ("bobik").sound();
```