Констр-ры в С++ (чтобы иниц. поля)!

В С++ много что делается неявно

```
class Matrix {
private:
  int height;
  int width;
  double *data;
public:
```

Это не гарантир... Конструктор иниц-но нужно вместо начальн.

иниц-я и в разных

Констр-ры:

① если не написс. констр-р, то будет создан констр-р по умолчанию
Matrix() {...}

или

② неявно сгенерир. констр-р по умолч.
Matrix() {}
Нужен, чтобы проинициализ. все поля.
Выз-ся при созд. объектов класса.

Matrix A; // будет вызван констр-р по умолч-ю (неявно).

Matrix A(5,5); // будет вызван

③ Констр-р с параметрами.
Matrix(int height, int width);

Замеч.:
Если написал констр-р(ы) с парам, то констр-р по умолч. неявно генер. не будет.

Matrix A; // Error!

Если объект создан, то его констр-ра определена.

пример конструктора. Констр-ров с пар-ми м.б. много

```
Matrix (int height, int width){
 this->height = height;
 this->width = width;
 data = malloc (height * width * sizeof(double));
 }
```

//this - указ-ль на объект (текущий)

Работу с памятью лучше в констр-р.

Выделение памяти вообще всю делать в конструкторах.

Очищать память нужно в деструкторе.

При удалении объекта выз-ся дестр-р.

```
class Matrix {
  ...
 ~ Matrix () {free(data);} //дестр-р
```

будет вызван неявно, когда уд-ся объекта

```
double trace( Matrix A){...};
trace (A); // в ф-ю trace перед-ся копия объекта A.
```

Констр-р копирования - вместо этого объекта возвр-ет такой же объект

Позв-ет контр-ть то, как происх коп-е объекта.

о.5. конструктор копир-я

Если он не задан, то будет происх-ть побитовое коп-е объекта (методы класса) в кач-ве арг объек по ссылке

```
Matrix ( const Matrix &  m) { // вызван в созданной копии.
height = m. height;   ← для созд. объекта равного (для копии)
width = m. width;
data = malloc (height * width * size of (double));
memcopy(. .); }.
```

Когда нужно си-ть объект, класса д.б. констр. коп-я. (иначе побитовое коп-е) → будет создан новый объект.

Когда выз-ся констр-р копир-я (неявно):

1) Matrix A; // создаём объект из др. объекта

Matrix B (A); // вызван констр-р копирования. передать в ф-ю A

2) trace (A); // вызван констр-р копир-я

3) Return A; // вызван констр-р коп-я. вернуть зн-е из ф-ции.

A = B; // по умолч. происх побитовое присвоение. не констр-р коп-я! т.к. не создано новый объект.

оба объекта уже сущ.

Matrix B = A; // присвоение

Matrix B(A); // копир-е

Перегрузка операций: (метод класса)

```
class Matrix {
...
...                                  A = B

Matrix & operator = (const Matrix &m){

width = m.width;                     m ~ B
height = m.height;                   this ~ A

free(data);           ← перегрузка опера-
. data = malloc(...);   ции присваи-
mem copy(...);          вания.
return * this;
}
```

A=B; // вызывается перегрузка операция

! перегрузку операций надо делать аккуратно. и перегр. операций типа +, - стараться избегать.

* Важно:

самокопиров-е

делать пр-ку на A=A;

if(*m == this) return *this

Констр-ры д.б. public, но если хотим запретить какие-то операции, то надо сд собств. констр-ю private.

↓

• error будут показаны IDE.

```
class linear_Equation { //Ax=B
v  Matrix A;
v  Matrix B;
}
```

linear_Equation eq01; // будет вызван констр-р по умолчанию linear_Equation и вызваны констр-ры по умолч для полей: A, B. Писать констр-р по умолч-ю!

$A = \underbrace{B + C}$;

создан объект Atemp = B+C
поскольку после выч-я Atemp удалится, то можно его не удалять, а переместить в A

A=B+C; // констр-р перемещ-я. Разное
A = B; // операция присваивания.

```
class Matrix {
  ...
  ...
  Matrix(Matrix && m) {
  width = m.width;
   height = m.height;
  data = m.data;
} m.data = nullptr; // при вызове деостр не постр полей адает
```

! Если констр-р перем не ~~[illegible]~~ задан, то будет неяв-й констр-р копий ⇒ двусмысленность!

```
setlocale (LC_ALL, "ru");
```

– – – – – – – – – – – –

Перегрузка ф-й:

Можем реализовать совершенно разные ф-ции, но все они будут иметь и то же имя!

```
int sum(int a, int b){}
int sum (int a, int b, int c){}
double sum (d a, d b, d c){}
```

– – – – – – – – –

перегр-ка констр класса:

Мы можем инициализировать класс разн спос-ми (как удобнее). (полиморфизм) – разное поведение в зав-ти от разных ситуаций.

– – – – – – – – – – –

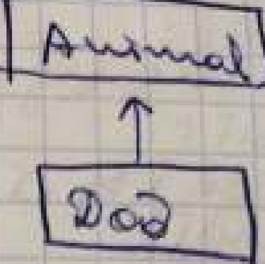
Деструктор класса. (при разрушении объекта класса) (только 1) (правильным образом освоб-ть ресурсы, кот выделялись в классе).

~ Имя класса (){}

Дестр-р не имеет пар-ров!

(вызыв-ся, когда, например, объект исчезает из зоны видимости!

Наследие в ООП

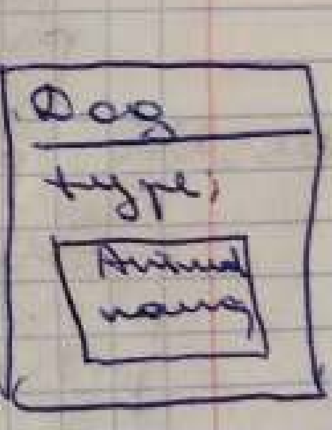


```
class Animal {
public: → private
string name;
public:
Animal(); → Animal (string new_name = ""): n
                                          name(
                                          (new_name)
                                          {};
sound();
}

class Dog : public Animal {
public: string type;
public:
 Dog(); ⟶ Dog(string new_name);
 Sound();   Dog (string new_name = "")

}

int main() {
 Dog bobik ("bobik"); // Dog("bobik");
} Dog noname;        // Dog();
```



! при создании объектов неявно вызывается конструктор родителя.

: констр-ра (но по умолч-ю).

```
Animal();
Dog ("bobik");
```

=> Если есть насл-е, то обяз-но созд-ть констр-р без аргументов.

Но если хочу ~~конкретный~~ вызвать конкретный констр-р, то

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

Огранич доступа.

public - доступна снаружи класса,
protected - ↙
private - доступна только внутри класса

Наследование

```
Class Dog: public Animal {
```

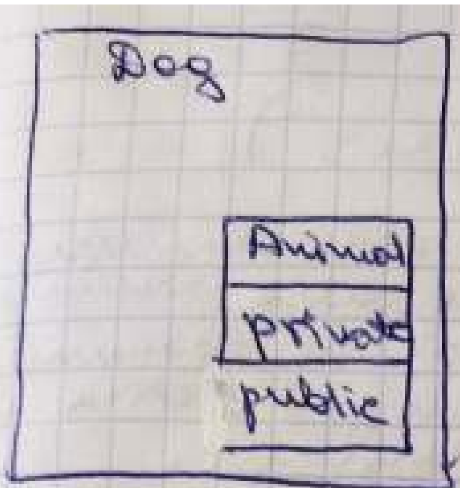
public => public } изменение
protected => protected } доступа
private => недоступен } при насл-ии.

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

public => private }
protected => private }
private => недоступен }

```
class Dog: protected Animal{
```

public => protected }
protected => protected }
private => недоступен }



Можно в наслед. переписать уровень доступа. (если поля нет, то и переписать нельзя, т.е private не переписыв-ся).

если в Animal: sound()
public:
хочу в Dog
protected sound()

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

protected:
  using Animal:: sound; // переписал на protect
}                          ур-нь дост. Dog к мет.
                           sound()
```

Сокрытие или удаление лишнего функц-ла:

```
class Dog: public Animal{
public:
   string type;
public:
   Dog (string new_name = " ");
   sound () = delete; // скрыли метод из класса
}                                          Dog.
```

```
Dog bobik{"bobik"};
Animal creature ("хзоз");
creature = bobik; //в переменную
                    creature скопир-ся
                    часть переменной
                    bobik, кот. относится
                    к Animal.

Dog bobik ("bobik");
Animal *creature01= &bobik; //указ-ль
Animal &creature02= bobik; //ссылка
bobik.sound(); // выз-ся Sound из Dog.
creature01 -> sound(); //выз-ся sound из An
creature02. sound(); //выз sound из Ani

void sound_three_times (Animal&
                              creature){
creature.sound();
creature.sound();
creature sound();
}
```

! Можно предоставить указатель на класс Animal как указатель на класс Dog (если он действительно указывает на объект класса Dog).

```
Dog bobix ("bobix");
Animal *creature = &bobix;
(dynamic_cast <Dog*> (creature)).sound()}
```

вызовется sound() из класса Dog

„понижающее привед. типов"

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

```
class Dog: public Animal{

 public:
   string type;
 public;
   Dog(string new_name = "");
  sound();
}
```

```
void sound_three_times(Animal&creature){

creature.sound();
creature.sound();
creature.sound();
}
```

Dog => sound для Animal.

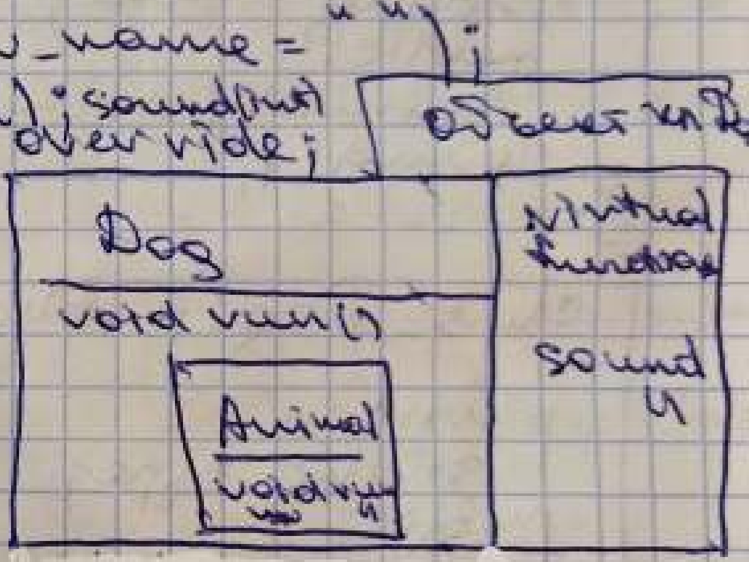
Как сделать так, чтобы вызвалась ф-я из класса Dog?

↓

Нужно объявить эту ф-ю виртуальной.

```
class Animal {
  public:
string name;
  public:
 Animal (string new_name = "") : name
virtual sound();                  (new_name)};
}void run();
}
```

```
class Dog : public Animal {
public:
 string type;
 public:
 Dog (string new_name = "");
 virtual sound() : sound(run)
}void run();       override;
}
```



Вирт ф-я - 1 для копии объекта

! Если определим ф-ю как вирт-ю, то она д.б. виртуальной у всех потомков

Виртуальная ф-я это позднее связ-е

! Дестр-ры д.б. виртуальными, если есть наслед-е

final - запрещает в дальнейшем наследовать класс и переопр. ф-ю;

virtual sound() override final;

дальше переопр. ее нельзя.

```
class Dog final : public Animal {
  ...
}
```
↑ нельзя созд-ть наследников класса Dog.

Абстрактные классы:

Идея интерфейса: пишу что класс должен делать (методы, переменные), но в виде кода - код рабочий (компилир-ся), но без деталей реализации.

```
class IAnimal {

private:
   string name;
   int age;
 public:
   string get_name() {};
   virtual void sound () = 0; // ф-я явл.
                                 абстрактна
}
```

Абстр. класс - класс у кот есть хотя бы 1 абстр. ф-я.

Объекты абстрактно не создаются.
(ошибка компиляц

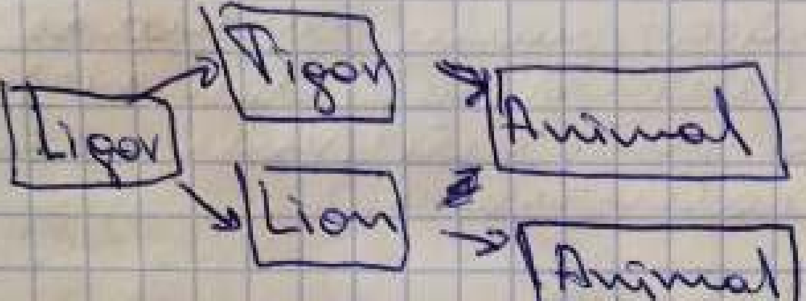
```
class Animal : public IAnimal {
 ...
}
```

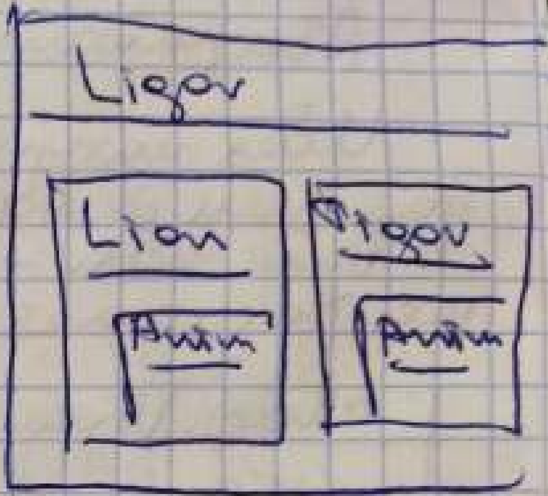
Множественное наслед-е:

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

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

class Ligor: public Lion, public Tiger {
}
```



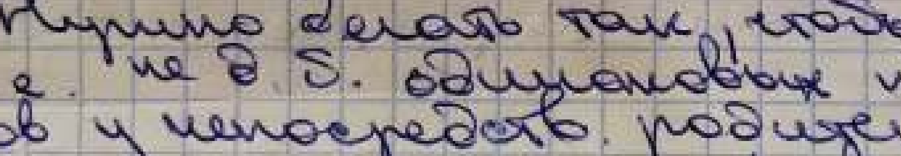


Проблемы:

① Два экземпляра Animal внутри класса.

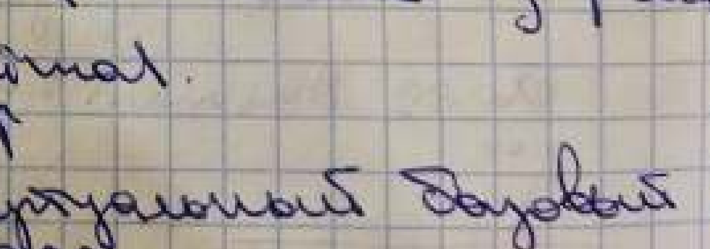
② Два одинаковых поля tail_length.

Нужно делать так, чтобы не было неопр т.е. не д. б. одинаковых полей и методов у непосредств. родителей. (и вообще у родит

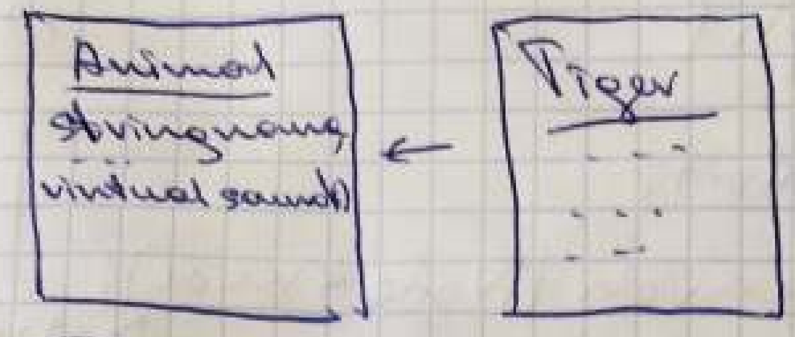


виртуальный базовый класс.

class Lion: virtual public Animal.



UML



Дружественные функции и классы.

```
class Graph {
private:
  Node *root;
public:
  Node *search(Node *node);
}

class Node {
private:
  void *data;
std::list<Note*>neighbors;

   ↑
 friend class Graph; //теперь у class Graph
                       будут видны поля
 }                     и методы class Node.
```

friend - дружба одност.

т.е. это почти как 1 класс.

Node is friend of Graph.

но не наоборот. у Node private методы и поля Graph недост-ны

A - friend of B

B - friend of C

A - не друг C.

```
class Graph {

  private:
    Node *Root;
  public:
    Node* Search (Node *node);
}

class Node {

  private:
    void *data;
  std::list<Node*> neighbors;
  friend Node* Graph::Search(Node*node
```

Метод друг класса Node т.е. ему доступны private поля Node.

Анонимны объекты.

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

Наслед-е:

① Модификат доступа при наслед.
public
private
protected

② Порядок вызова констр-ов при наслед

③ Порядок вызова деструкт при наслед.

④ Повыш привед типов (к наследнику)

⑤ Понижающ. приведтипов (к родителю

⑥ Обрезка объектов (animal = dog);

⑦ virtual методы (virtual дестр-р)

⑧ абстрактный класс (интерфейсы).

⑨ Множеств. наслед-е (виртуальн классы)

⑩ Дружеств. методы и дружеств класс

⑪ Анонимные объекты