C++
Лекция 2
Механизм наследования

Конструкторы

```
class Triangle {
private:
  double a;
  double b;
  double c;

public:

  Triangle(double a_in, double b_in, double c_in)
    : a(a_in), b(b_in), c(c_in) {

    // Тело функции пустое
  }

  Triangle()
    : a(1), b(1), c(1) { }
};
```

Имя конструктора совпадает с именем класса

Специальный синтаксис для инициализации полей класса в конструкторе

Конструктор по умолчанию не принимает аргументов

- Порядок инициализации полей класса зависит от порядка объявления их в классе!

```
class Triangle {
private:
  double a;
  double b;
  double c;

public:
  Triangle(double a_in, double b_in, double c_in)
    : c(c_in), b(b_in), a(a_in) {

  }
};
```

сначала инициализируется a, затем b и c.

Этот порядок игнорируется

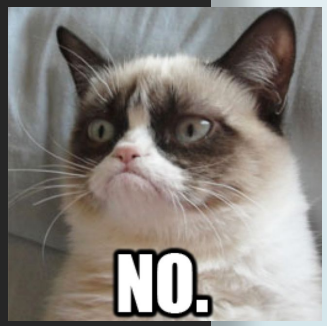
Делегирующие конструкторы

```
class Triangle {
  ...
  Triangle(double a_in, double b_in, double c_in)
    : a(a_in), b(b_in), c(c_in) {
    // Be assertive!
    assert(0 < a && 0 < b && 0 < c);
    assert(a + b > c  &&  a + c > b  &&  b + c > a);
  }

  Triangle(double side_in)
    : a(side_in), b(side_in), c(side_in) {
    // Be assertive!
    assert(0 < a && 0 < b && 0 < c);
    assert(a + b > c  &&  a + c > b  &&  b + c > a);
  }

  Triangle(double side_in)
    : Triangle(side_in, side_in, side_in) { }

};
```

Валидация Triangle.

Дублирование кода!

Уже лучше)



Делегирующие конструкторы

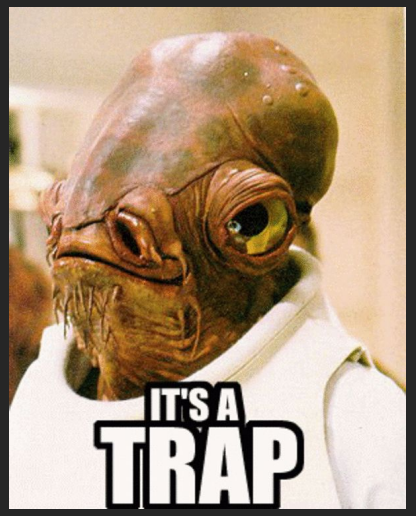
```
class Triangle {
  ...
  Triangle(double a_in, double b_in, double c_in)
    : a(a_in), b(b_in), c(c_in) {
    // Be assertive!
    assert(0 < a && 0 < b && 0 < c);
    assert(a + b > c  &&  a + c > b  &&  b + c > a);
  }


  Triangle(double side_in)
    : Triangle(side_in , side_in, side_in) {
  }

  Triangle(double side_in) {
    Triangle(side_in , side_in, side_in);
  }
};
```

DO

DON'T

Делегирование происходит в списке инициализации

Что видит компилятор:
`Triangle temp(side_in, side_in, side_in);`



Exercise: Прямоугольник

- Написать класс Прямоугольник
- Который содержит:
  - Конструктор по умолчанию создающий прямоугольник размером 1x1.
  - Конструктор принимающий один параметр s (длина стороны) и создающий прямоугольник размером s x s.
  - Конструктор принимающий два параметра - длина и ширина.
  - Методы для подсчета площади и периметра (методы не принимают аргументов).

Solution: Прямоугольник

```
class Rectangle {

private:
  double w, h;

public:
  Rectangle()
    : Rectangle(1, 1) { }

  Rectangle(double s)
    : Rectangle(s, s) { }

  Rectangle(double w_in,
            double h_in)
    : w(w_in), h(h_in) { }

  ...
```

Поля класса

Делегирующие конструкторы

Основной конструктор

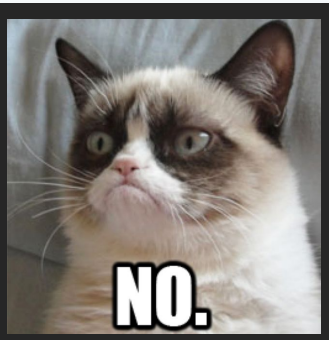
```
  ...

  double area() const {
    return w * h;
  }

  double perimeter() const {
    return 2 * (w + h);
  }


};
```

Птички

Дублирование кода!

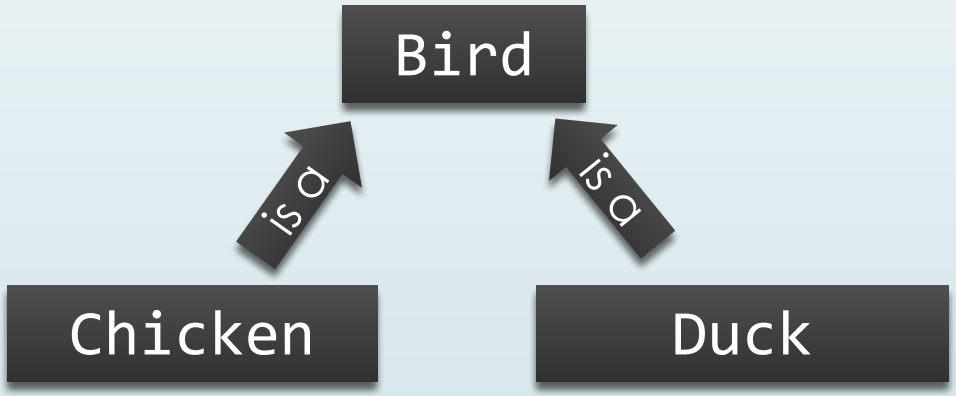
```
class Chicken {
private:
  int age;
  string name;
  int roadsCrossed;

public:
  Chicken(string name_in)
    : age(0), name(name_in),
      roadsCrossed(0) {
    cout << "Chicken ctor" << endl;
  }
  string getName() const
  { return name; }
  int getAge() const
  { return age; }
  void crossRoad() {
    ++roadsCrossed;
  }
  void talk() const {
    cout << “bawwk" << endl;
  }
};
```

```
class Duck {
private:
  int age;
  string name;
  int numDucklings;

public:
  Duck(string name_in)
    : age(0), name(name_in),
      numDucklings(0) {
    cout << "Duck ctor" << endl;
  }
  string getName() const
  { return name; }
  int getAge() const
  { return age; }
  void babyDucklings() {
    numDucklings += 7;
  }
  void talk() const {
    cout << "quack" << endl;
  }
};
```



Наследование

- Рассмотрим абстрактный тип данных для представления птиц
- Курицы и Утки - Птицы)



- Мы можем использовать базовый тип Птица, а Куриц и Уток наследовать от базового типа.

Птицы и наследование

Базовый класс

```
class Bird {
private:
  int age;
  string name;

public:
  Bird(string name_in)
    : age(0), name(name_in) {
    cout << "Bird ctor" << endl;
  }
  string getName() const {return name;}
  int getAge() const { return age; }
  void haveBirthday() { ++age; }

  void talk() const {
    cout << "tweet" << endl;
  }
};
```

Поля общие для всех птиц

Класс наследник

```
class Chicken : public Bird {
private:
  int roadsCrossed;

public:
  Chicken(string name_in)
    : Bird(name_in), roadsCrossed(0) {
    cout << "Chicken ctor" << endl;
  }

  void crossRoad() { ++roadsCrossed; }

  void talk() const {
    cout << "bawwk" << endl;
  }
};
```

Специальное поле, только для Куриц

Передача параметра в конструктор базового класса

Птицы и наследование

Базовый класс

```
class Bird {
private:
  int age;
  string name;

public:
  Bird(string name_in)
    : age(0), name(name_in) {
    cout << "Bird ctor" << endl;
  }
  string getName() const {return name;}
  int getAge() const { return age; }
  void haveBirthday() { ++age; }

  void talk() const {
    cout << "tweet" << endl;
  }
};
```

Поля общие для всех птиц

Ещё один класс наследник

```
class Duck : public Bird {
private:
  int numDucklings;

public:
  Duck(string name_in)
    : Bird(name_in), numDucklings(0) {
    cout << "Duck ctor" << endl;
  }

  void babyDucklings() {
    numDucklings += 7;
  }
  void talk() const {
    cout << "quack" << endl;
  }
};
```

Специальное поле, только для Уток

Передача параметра в конструктор базового класса

Типы наследования

- В C++ существует 3 типа наследования
  - `public`
  - `protected`
  - `private`

```
class Duck : public Bird {
   ...
   ...
   ...
};
```

Если здесь не указан тип, то по умолчанию считается private

Типы наследования

- При объявлении члена класса открытым (***public***) к нему можно получить доступ из любой другой части программы.
- Если член класса объявляется закрытым (***private***), к нему могут получаться доступ только члены того же класса. К закрытым членам базового класса не имеют доступа даже производные классы.
- Если член класса объявляется защищенным (***protected***), к нему могут получать доступ только члены тоже же или производных классов. Спецификатор ***protected*** позволяет наследовать члены, но оставляет их закрытыми в рамках иерархии классов.

Типы наследования

- Если базовый класс наследуется как ***public***:
  - ***public*** -> ***public***
  - ***protected*** -> ***protected***
- Если базовый класс наследуется как ***protected***:
  - ***public*** -> ***protected***
  - ***protected*** -> ***protected***
- Если базовый класс наследуется как ***private***:
  - ***public*** -> ***private***
  - ***protected -> private***

Время жизни объекта

- Конструктор вызывается в момент создания объекта, один раз.

```
Triangle()
  : a(1), b(1), c(1) {
  cout << "Triangle ctor" << endl;
}
```

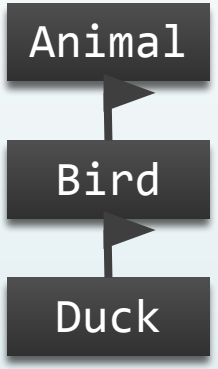
- Деструктор вызывается в момент окончания жизни объекта (зависит от типа продолжительности хранения)

```
~Triangle() {
  cout << "Triangle dtor" << endl;
}
```

- Для локальных переменных время жизни заканчивается когда они покидают область видимости.

Конструкторы и деструкторы в класса-наследниках

- Деструкторы вызываются в момент уничтожения объекта
- Конструкторы вызываются в порядке происхождения классов.
- Деструкторы в обратном порядке.




```
int main() {
  Duck d("Scrooge");  // Animal ctor, Bird ctor, Duck ctor
  Bird b("Big Bird"); // Animal ctor, Bird ctor
  ...
```

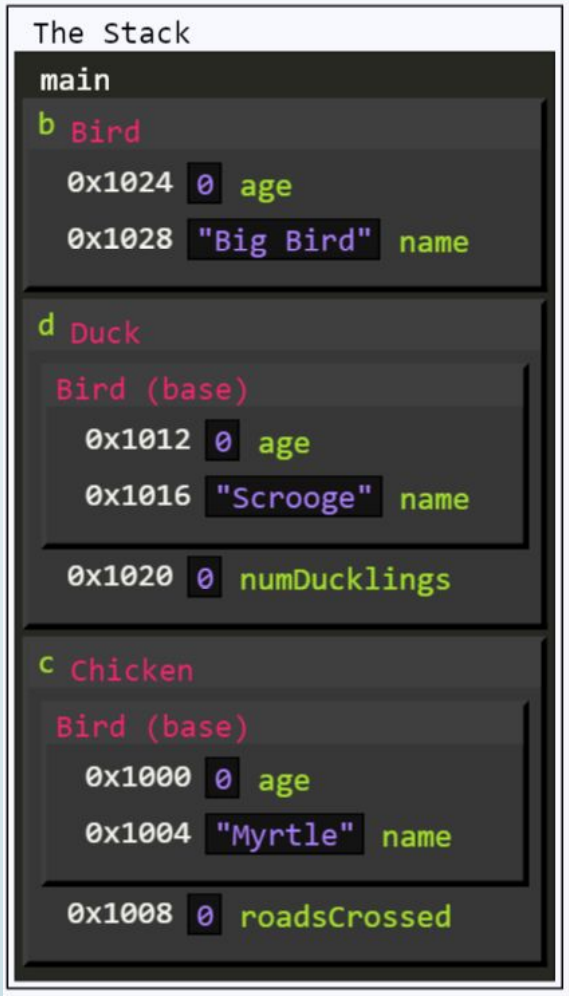
```
  ...
  // b dies: Bird dtor, Animal dtor
  // d dies: Duck dtor, Bird dtor, Animal dtor
};
```

Наследование и память

- В памяти класс-наследник содержит в себе часть базового класса.
- Чтобы получить доступ к полям базового класса используется оператор . или ->

```
int main() {
  Chicken c("Myrtle");
  Duck d("Scrooge");
  Bird b("Big Bird");
  c.age;
}
```

Если **c** сделать private, то работать не будет



Порядок поиска членов класса

- Сначала компилятор ищет среди полей и методов класса-наследника
- Если не находит, то ищет в базовом классе

```
int main() {
  Chicken c("Myrtle");
  Duck d("Scrooge");
  Bird b("Big Bird");
  c.getAge();
}
```

Поиск метода getAge.

```
class Bird {
  int age;
  string name;
  Bird(string name_in);
  void talk() const;
  string getName() const;
  string getAge() const;
};
```

```
class Chicken : public Bird {
  int roadsCrossed;
  Chicken(string name_in);
  void talk() const;
};
```

Tip: Уровень доступа проверяется после поиска

Скрытие

- Сначала компилятор ищет среди полей и методов класса-наследника
- Если не находит, то ищет в базовом классе
- **Останавливается если не находит совпадений**

Скрытый

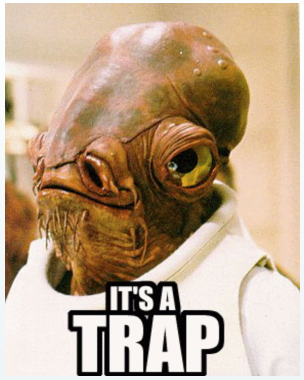
```
class Bird {
  int age;
  string name;
  Bird(string name_in);
  void talk() const;
  string getName() const;
  string getAge() const;
};
```

Поиск метода talk.

```
int main() {
  Chicken c("Myrtle");
  Duck d("Scrooge");
  Bird b("Big Bird");
  c.talk();
}
```

```
class Chicken : public Bird {
  int roadsCrossed;
  Chicken(string name_in);
  void talk() const;
};
```

Exercise: Name Hiding



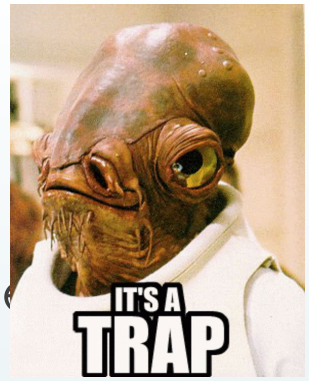
- Что случится в каждом из случаев?
  - Какой член класса будет найдет?
  - Скомпилируется?
  - Произойдет то чего ожидали?

```
int main() {
  Derived der;
  Duck q;

1  int y = der.x;
2  der.foo("test");
3  der.bar(&q);

}
```

```
class Base {
public:
  int x;
  void foo(int a);
  int foo(string b);
  void bar(Duck *c);
};
```

```
class Derived : public Base
{
public:
  void x(int a);
  int foo(int b);
  void bar(bool c);
};
```

Solution: Name Hiding



Все поля и методы будут найдены в классе Derived

1. Не скомпилируется.
2. Не скомпилируется.
3. Скомпилируется! Указатель преобразуется к bool.

```
int main() {
  Derived der;
  Duck q;

1  int y = der.x;
2  der.foo("test");
3  der.bar(&q);

}
```

```
class Base {
public:
  int x;
  void foo(int a);
  int foo(string b);
  void bar(Duck *c);
};
```

```
class Derived : public Base
{
public:
  void x(int a);
  int foo(int b);
  void bar(bool c);
};
```

Доступ к скрытым полям

•

```
class Bird {
private:
  int age;
  string name;

public:

  ...

  void talk() const {
    cout << "tweet" << endl;
  }
};
```

```
class Chicken : public Bird {
private:
  int roadsCrossed;

public:

  ...

  void talk() const {
    if (age >= 1) {
      cout << "bawwk" << endl;
    } else {
      // baby chicks tweet
      Bird::talk();
    }
  }
};
```

Problem: **age** is private in **Bird**

Call **Bird**'s version of **talk()**

Protected

- Для того чтобы сделать поля класса доступными для классов наследников уровень доступа необходимо изменить с **private** на **protected**.

```
class Bird {
protected:
  int age;
  string name;

public:

  ...

  void talk() const {
    cout << "tweet" << endl;
  }
};
```

Проблема: раскрывает реализацию Bird

```
class Chicken : public Bird {
private:
  int roadsCrossed;

public:

  ...

  void talk() const {
    if (age >= 1) {
      cout << "bawwk" << endl;
    } else {
      // baby chicks tweet
      Bird::talk();
    }
  }
};
```

Что делать?

- Использовать **get** функцию :)

```
class Bird {
private:
  int age;
  string name;

public:

  ...

  int getAge() const {
    return age;
  }

  void talk() const {
    cout << "tweet" << endl;
  }
};
```

```
class Chicken : public Bird {
private:
  int roadsCrossed;

public:

  ...

  void talk() const {
    if (getAge() >= 1) {
      cout << "bawwk" << endl;
    } else {
      // baby chicks tweet
      Bird::talk();
    }
  }
};
```