Объектно-ориентированное программирование

Конструкторы в языке C++

Вопросы

- Назовите основные отличия синтаксиса C от синтаксиса C++:
  - Методы ввода-вывода?
  - Описание переменных?
  - Области видимости?
- Как реализуется принцип инкапсуляции при описании классов в C++?
- Что означает ключевое слово static в C++?
- Как получить доступ к статическому методу в C++?
- Что такое указатель this? Как его можно использовать?

Модификатор CONST

Ключевое слово const

- Ключевое слово `const`, как и ключевое слово `static`, обладает множеством различных смыслов в зависимости от того места, где оно употребляется.
- Самый простой случай, обычная переменная. Переменная объявляется, тут же инициализируется, менять ее значение больше нельзя.

```
const int p=4;
p=5; //ошибка
```

Ключевое слово const

- Про использование `const` с указателями есть известный C++ паззл, который любят давать на собеседованиях при приеме на работу.
- Чем отличаются:

```
int *const p1;
int const* p2;
const int* p3;
```

Ключевое слово const

- Необходимо провести мысленно вертикальную черту по звездочке. То, что находится справа относится к переменной. То, что слева - к типу, на который она указывает. Вот например:

```
int *const p1;
```

- Справа находится p1, и это p1 константа. Тип, на который p1 указывает, это `int`. Значит получился константный указатель на `int`. Его можно инициализировать лишь однажды и больше менять нельзя.
  Нужно так:

```
int q=1;
int *const p1 = &q; //инициализация в момент объявления
*p1 = 5;  //само число можно менять
```

Ключевое слово const

- Объявления

```
int const* p2;
const int* p3;
```

это по разному записанное одно и то же объявление. Указатель на целое, которое нельзя менять.

```
int q=1;
const int *p;
p = &q; //на что указывает p можно менять
*p = 5; //ошибка, число менять уже нельзя
```

Ключевое слово const

- Ссылка на объект, который нельзя менять

```
int p = 4;
const int& x=p; //нельзя через x поменять значение p
x=5; //ошибка
```

- Константная ссылка

```
int& const x; //не имеет смысла
```

CONST: ПЕРЕДАЧА ПАРАМЕТРОВ

- `const` удобен, если нужно передать параметры в функцию, но при этом надо обязательно знать, что переданный параметр не будет изменен

```
void f1(const std::string& s);
void f2(const std::string* sptr);
void f3(std::string s);
```

CONST: КЛАССЫ

- Значения `const` данных класса задаются один раз и навсегда в конструкторе.

```
class CFoo {
  const int num;
public:
  CFoo(int anum);
};
CFoo::CFoo(int anum):num(anum)
{
...
}
```

const: классы

- Функция класса, объявленная `const`, трактует `this` как указатель на константу.

```
class CFoo
{
public:
  int inspect() const; // Эта функция обещает не менять *this
  int mutate(); // Эта функция может менять *this
};
```

- В таких методах не может быть ничего присвоено переменным класса, которые не объявлены как `static` или как `mutable`. Также `const`-функции не могут возвращать не `const` ссылки и указатели на данные класса и не могут вызывать не `const` функции класса.

КОНСТРУКТОРЫ


ПОМНИМ.
скорбим.
demotivation.ru

Конструктор объектов

- Конструктор – это метод, предназначенный для инициализации объекта и вызывающийся автоматически при его создании.
  - *не возвращает значения*
  - Класс может иметь *несколько конструкторов*
  - *Параметры конструктора* могут иметь любой тип, кроме этого же класса
  - компилятор может создать его *автоматически*
  - *Конструкторы не наследуются*.
  - *Конструктор не может быть константным, статическим и виртуальным*
  - Конструкторы глобальных объектов вызываются до вызова функции `main`.

Конструкторы по умолчанию

- Конструкторы часто вызываются неявно для создания временных объектов. Обычно это происходит в следующих случаях:
  - при инициализации;
  - при выполнении операции присваивания;
  - для задания значений параметров по умолчанию;
  - при создании и инициализации массива;
  - при создании динамических объектов;
  - при передаче параметров в функцию и возврате результатов по значению.
- В этих случаях вызывается тот конструктор, который может быть вызван без передачи в него значений параметров

```
class monster
{
      int health, ammo;
      color skin;
      char *name;
public:
      monster(int he = 100, int am  = 10);
      monster(color sk);
      monster(char * nam);
      ...
};
```

ОН

Примеры реализации конструкторов

```
class monster
{
    int health, ammo;
    color skin;
    char *name;
public:
    monster
       (int he = 100, int am  = 10);
    monster
       (color sk);
    monster
       (char * nam);
...
};
```

Выделение памяти под динамический объект

```
monster::monster(int he, int am)
{ health = he; ammo = am; skin = red; name = 0;}
//--------------------------------

monster::monster(color sk)
{
  switch (sk)
  {
     case red: health = 100; ammo = 10; skin = red; name = 0; break;
     case green: health = 100;ammo = 20;skin = green; name = 0; break;
     case blue: health = 100; ammo = 40; skin = blue; name = 0; break;
  }
}

//--------------------------------
monster::monster(char * nam)
{
   /* К длине строки добавляется 1 для хранения нуль-символа */
   name = new char [strlen (nam) + 1];
   strcpy (name, nam);
   health = 100; ammo = 10; skin = red;
}

//--------------------------------
monster * m = new monster ("Ork");
monster Green (green);
```

Конструктор копирования

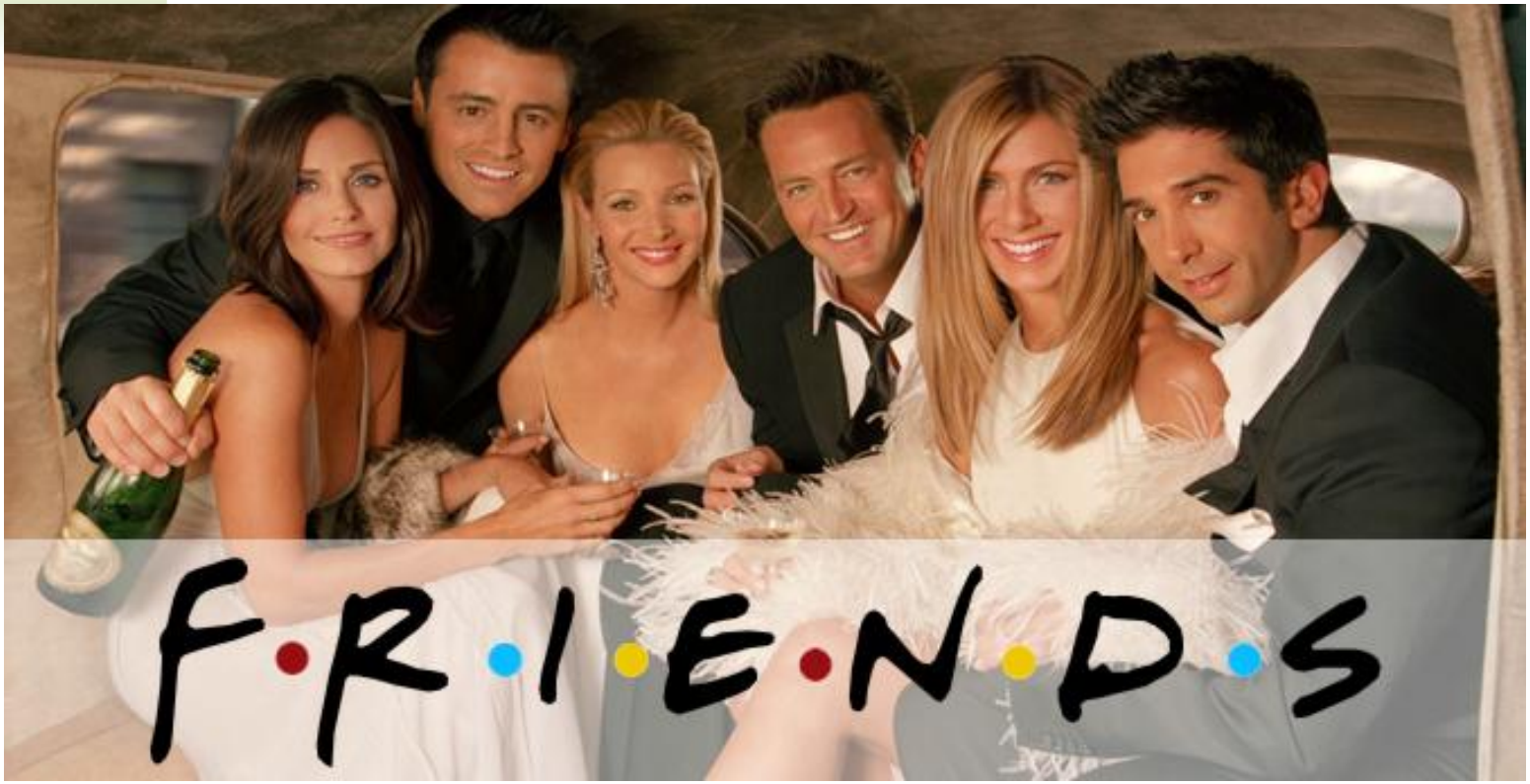
- Конструктор копирования - это специальный вид конструктора, получающий в качестве единственного параметра указатель на объект этого же класса. Этот конструктор вызывается в тех случаях, когда новый объект создается путем копирования существующего:
  - при описании нового объекта с инициализацией другим объектом;
  - при передаче объекта в функцию по значению;
  - при возврате объекта из функции.

`T::T(const T&)`

Копирование динамического объекта

```
monster::monster(const monster &M)
{
if (M.name)
  {
    name = new char [strlen(M.name) + 1];
    strcpy(name, M.name);
  }
  else name = 0;
  health = M.health; ammo = M.ammo; skin = M.skin;
}
...
monster Vasia (blue);
monster Super = Vasia;// Работает конструктор копирования
monster *m = new monster ("Ork");
monster Green = *m; // Работает конструктор копирования
```

Дружественные функции и классы


F·R·I·E·N·D·S

Дружественная функция

- Дружественная функция объявляется *внутри класса*, к элементам которого ей нужен доступ, с ключевым словом `friend`.
- В качестве параметра ей должен передаваться объект или ссылка на объект класса, поскольку указатель `this` ей не передается.
- Одна функция может "дружить" сразу с несколькими классами

Пример дружественной функции

```
class monster;// Предварительное объявление класса
class hero
{
   ...
   void kill(monster &);
};

class monster
{
   ...
   friend int steal_ammo(monster &);
   /* Класс hero должен быть определен ранее */
   friend void hero::kill(monster &);
};

int steal_ammo(monster &M){return --M.ammo;}
void hero::kill(monster &M){M.health = 0; M.ammo = 0;}
```

Дружественный класс

- Если все методы какого-либо класса должны иметь доступ к скрытым полям другого, весь класс объявляется дружественным с помощью ключевого слова `friend`.
- Объявление `friend` не является спецификатором доступа и не наследуется. Обратите внимание на то, что класс сам определяет, какие функции и классы являются дружественными, а какие нет.

Деструктор

Деструкторы

- Деструктор - это особый вид метода, применяющийся для освобождения памяти, занимаемой объектом. Деструктор вызывается автоматически, когда объект выходит из области видимости:
  - для *локальных* переменных - при выходе из блока, в котором они объявлены;
  - для *глобальных* - как часть процедуры выхода из main;
  - для *объектов, заданных через указатели*, деструктор вызывается неявно при использовании операции delete (автоматический вызов деструктора при выходе указателя из области действия не производится).

Деструкторы

- При уничтожении массива деструктор вызывается для каждого элемента удаляемого массива. Для динамических объектов деструктор вызывается при уничтожении объекта операцией `delete`. При выполнении операции `delete`[] деструктор вызывается для каждого элемента удаляемого массива.

Деструкторы

- Имя деструктора начинается с тильды (~), непосредственно за которой следует имя класса. Деструктор:
  - не имеет аргументов и возвращаемого значения;
  - не может быть объявлен как `const` или `static`;
  - не наследуется;
  - может быть виртуальным;
  - может вызываться явным образом путем указания полностью уточненного имени; это необходимо для объектов, которым с помощью `new` выделялся конкретный адрес.

Итого

- Const
- Конструкторы
- Друзья
- Деструкторы