

# Объектно-ориентированное программирование

Инкапсуляция в C++. Классы

Гришмановский Павел Валерьевич,  
кафедра автоматики и компьютерных систем, Политехнический институт, СурГУ

# Описание класса

```
class <имя_кл> [: <СПИСОК_кл_предков>]
{
    <описания>
}
[<СПИСОК_экземпляров>];
```

## Описание полей данных

```
<тип> <имя> [, <имя> ...];
```

## Описание методов (прототип)

```
<т.в.з.> <имя_м>(<СПИСОК_форм_парам>) [const];
```

## Реализация методов (вне класса)

```
<т.в.з.> <имя_кл>::<имя_м>(<СПИСОК_форм_парам>) [const]
{
    <тело_метода>
}
```

Содержит описания:

1. Вложенные (внутренние) типы данных
2. Поля (данные, переменные)
3. Методы (функции)
4. Модификаторы (директивы) доступа
5. Отношения дружественности

- Конструкторы и деструкторы
- Виртуальные и абстрактные методы
- Встраиваемые методы
- Константные методы
- Перегрузка операции
- Статические члены класса

# Модификаторы доступа

`private`: доступ только внутри класса

`protected`: доступ внутри класса и в его потомках

`public`: доступ вне класса

Действие распространяется до следующего модификатора доступа или до конца описания класса

Все члены класса доступны в его пространстве, т.е. в теле любого метода этого класса

# Конструкторы и деструкторы

## Описание конструкторов (прототип)

[**void**] <имя\_кл>(<список\_форм\_парам>) ;

## Описание деструкторов (прототип)

[**void**] ~<имя\_кл>([**void**]) ;

# Конструкторы и деструкторы

**Прототип конструктора (внутри описания класса)**

[**void**] <имя\_кл>(<список\_форм\_парам>) ;

**Прототип деструктора (внутри описания класса)**

[**void**] ~<имя\_кл>([**void**]) ;

**Реализация конструктора (вне описания класса)**

[**void**] <имя\_кл>::<имя\_кл>(<список\_форм\_парам>  
[ : <имя>(<список\_факт\_парам\_конс>), ... ] -  
{ } )

<имя> – имя поля данных или класса-предка

**Реализация деструктора (вне описания класса)**

[**void**] <имя\_кл>::~<имя\_кл>([**void**])  
{ }

# Создание объектов

Глобальные, статические и локальные (стековые) объекты:

`<имя_кл> <имя>(<список_факт_парам_конс>);`

- Локальные: область действия и время жизни – до конца блока
- Глобальные: область действия – глобальная, время жизни – до конца выполнения программы
- Статические: область действия – в пределах блока, время жизни – до конца выполнения программы

Динамические объекты:

`new <имя_кл>(<список_факт_парам_конс>)`

- Доступны по указателю, время жизни – бесконечно, до явного уничтожения

Анонимные стековые экземпляры

`<имя_кл>(<список_факт_парам_конс>)`

- Доступен непосредственно, время жизни – текущее выражение

# Конструктор по умолчанию

Не имеет параметров:

```
<имя_кл> ( [void] ) ;
```

Вызывается в случаях:

- Список фактических параметров конструктора пуст или не указан
- При создании массива объектов – для каждого объекта (элемента массива)
- Для поля или класса-предка, если в конструкторе другого класса не приведен вызов конструктора для этого поля или класса-предка

# Конструктор копирования

Предназначен для создания объекта как копии существующего объекта того же класса.

Имеет единственный параметр – ссылку на объект-оригинал (обычно константный):

```
<имя_кл> (const <имя_кл> & <имя>) ;
```

Вызывается в случаях:

- Явно – единственным параметром конструктора является объект того же класса
- При передаче объекта через стек в качестве параметра или возвращаемого значения
- Для поля или класса-предка, если в конструкторе копирования другого класса не приведен вызов конструктора для этого поля или класса-предка

# Конструктор преобразования

Предназначен для создания объекта, инициализированного значением другого типа.

Имеет единственный параметр произвольного типа:

`<имя_кл> (<тип> <имя>) ;`

Вызывается в случаях:

- Явно
- При передаче через стек в качестве параметра или возвращаемого значения вместо объекта значения другого типа
- При присваивании объекту значения другого типа (если отсутствуют соответствующие перегруженные операции присваивания и приведения типа)

Неявный вызов может быть запрещен:

`explicit <имя_кл> (<тип> <имя>) ;`

# Указатель **this**

Специальный скрытый параметр любого метода:

- является указателем на тот объект, для которого вызван метод
- всегда имеет тип <класс>\*, соответствующий классу, в котором этот метод реализован

Вызов любого метода выполняется только для конкретного объекта, следовательно, известен адрес объекта

Объект представлен структурой его данных (как обычная структура в памяти), указатель **this** – указатель на эту структуру

Метод является обычной функцией, которая получает указатель **this** в качестве параметра

# Статические элементы класса

Данные и методы, объявленные с использованием **static**:

- являются общими для всех экземпляров класса (как глобальные переменные и функции, но «спрятанные» в пространстве класса)
- распространяется действие модификаторов доступа
- доступны:
  - в методах класса (статических и нестатических) – непосредственно
  - посредством имени экземпляра или ссылки (извне) – прямой селектор .
  - посредством указателя (может не указывать на конкретный объект) – косвенный селектор ->
  - посредством имени класса – операция видимости ::
- статические поля не могут быть инициализированы в конструкторе
- в статических методах нет указателя **this**
- статические методы не могут обращаться к нестатическим полям и методам без явного указания экземпляра

# Встраиваемые функции и методы

Тело встраиваемого метода или функции непосредственно встраивается в код вместо вызова – это ускоряет выполнение и может уменьшить размер программы, т.к. сокращается код и время, требуемые для:

- передачи параметров и возвращаемого значения через стек
- настройку и восстановление стека
- вызова подпрограммы

Встраиваемые функции и методы не должны содержать:

- операторы ветвлений и циклов
- операторы передачи управления (перехода), кроме `return`
- рекурсию

Реализация встраиваемых функций и методов выполняется с использованием ключевого слова `inline`:

`inline <т.в.з.> <имя_м>(<список_форм_парам>) . . .`

или (методов) непосредственно в описании класса

# Константные объекты и константные методы

# Отношения дружественности

Отношения дружественности необходимы для предоставления доступа к закрытым (частным и защищенным) членам класса некоторому другому классу, отдельному методу класса или отдельной функции

- Описываются только внутри описания класса, предоставляющего дружественность
- Место описания не имеет значения, модификаторы доступа не влияют
- Односторонние
- Не передаются по другим отношениям дружественности
- Не передаются при наследовании (ни с одной стороны)

## Описание дружественности:

```
class ...
{
    friend class <имя_кл>;
    friend <т.в.з.> <имя_кл>::<имя_мет>(<сп.форм.п.>) [const];
    friend <т.в.з.> <имя_мет>(<сп.форм.п.>);
};
```