Конструкторы классов

Конструкторы классов

Конструктор - это особый тип метода класса, который автоматически вызывается при создании объекта этого же класса. Предназначен для инициализации объектов класса.

```
class Point {
public:
    Point();
    Point(int, int);

    int x;
    int y;
};

Point::Point() {
    x = 0;
    y = 0;
}
Point::Point(int _x, int _y) {
    x = _x;
    y = _y;
}
```

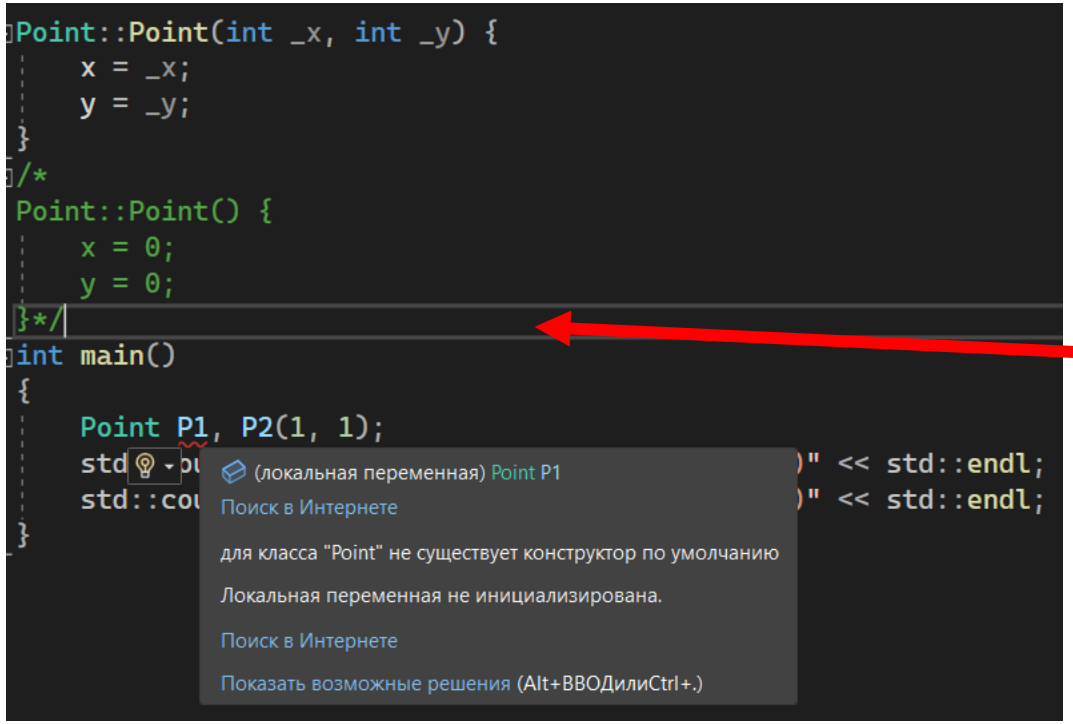
Конструктор по умолчанию

Конструктор с параметрами

Конструкторы классов

```
int main()
{
    Point P1, P2(1,1);
    std::cout << "(" << P1.x << ":" << P1.y << ")" << std::endl;
    std::cout << "(" << P2.x << ":" << P2.y << ")" << std::endl;
}
```

```
(0:0)
(1:1)
```



Если в классе не создать ни одного конструктора, то конструктор по умолчанию создастся автоматически.
Но если в классе есть хотя бы один конструктор, то КПУ не создается

Конструктор копирования

```
Point::Point(const Point& T) {
    x = T.x;
    y = T.y;
}
```

Передаем значение по **константной ссылке**.

1. Почему по ссылке?
Если передадим само значение, то компилятор создаст копию объекта, что приведет к рекурсии…
2. А зачем константная ссылка?
Тогда не сможем копировать константные объекты, т.к. привести не константный объект к константному мы можем, но не наоборот.

noconst = const // NO OK

const = noconst // OK

Конструктор копирования

```
Point::Point(const Point& T) {
    x = T.x;
    y = T.y;
}
```

```
Point P1(1, 1), P3;
Point P2 = P1; //так скопируется
P3 = P1;       //а так нет
```

При первом вызове «=» создается новый объект. Знак «=» обозначает вызов конструктора, не присваивание.

Во втором случае знак = обозначает присваивание. И т.к. вокруг него неизвестные типы, то его необходимо перегрузить.

Конструктор копирования: перегрузка =

```
Point& Point::operator=(const Point& other)
{
    if (this == &other)
        return *this;
    this->x = other.x;
    this->y = other.y;
}
```

Возвращаемое значение оператора присваивания – **ссылка** на наш тип.

На вход оператор принимает константную ссылку, аналогично с самим конструктором

В случае, если мы попробуем выполнить присваивание самому себе, то получится неопределённое поведение -> нужно сделать проверку

Конструктор перемещения

```
Point::Point(Point&& T) noexcept {
    x = T.x;
    y = T.y;
    T.x = T.y = 0;
}
```

```
Point& Point::operator=(Point&& other) noexcept
{
    if (this == &other)
        return *this;
    this->x = other.x;
    this->y = other.y;
    other.x = other.y = 0;
    return *this;
}
```

```
Point P1(1, 1), P2,  P3;
P2 = std::move(P1);
Point P4 = std::move(P2);
```

Деструкторы

Деструктор - это особый тип метода класса, который автоматически вызывается при уничтожении объекта этого же класса. Предназначен для корректного удаления объектов класса.

```
Point::~Point() {
    std::cout << "~Point" << std::endl;
}
```