Перегрузка операторов (продолжение)

5 июня 2017 г.

Повторение

1. Что такое бинарные и унарные операторы?
2. Зачем нужна перегрузка операторов?
3. Какие есть способы перегрузки операторов?
4. Какие операторы нельзя перегрузить?
5. Сколько аргументов имеют перегруженные бинарные операторы – методы класса?
6. Сколько аргументов имеют перегруженные унарные операторы?

Есть ли ошибки в коде?

```
class Point {
      double x_;
      double y_;
public:
      Point( ) : x_(0.0), y_(0.0) { }
      Point(const double x, const double y) : x_(x), y_(y) { }
      explicit Point(const int x, const int y = 0) : Point(x, y) { }
      ~Point( ) { }
 };

void f ( ){
      Point myPoint(2, 10);
}
```

Есть ли ошибки в коде?

```
class Point {
    double x_;
    double y_;
public:
    …
    Point& operator+(const Point& p, const int step){
            p.x_ += step;
            p.y_ += step;
            return *this;
    }
};
```

Есть ли ошибки в коде?

```
class Point {
    double x_;
    double y_;
public:
    …
    Point& operator-(const int step){
            x_ = x_ - step;
            y_ = y_ - step;
            return this;
    }
};
```

Есть ли ошибки в коде?

```
class Point {
        double x_;
        double y_;
public:
        …
        bool operator==(const Point& p){
                if (this == &p) { return true; }
                else if ( x_ == p.x_ && y_ == p.y_ ) { return true; }
                else { return false; }
        }
 };
```

Преобразования типов (операторы приведения)

```
operator type( );
        operator int( );
        operator Square( );
```

- преобразуют пользовательский тип к стандартному (MyClass => int и т.п.) или другому пользовательскому (Rectangle => Square)
- не имеют аргументов
- не указывается тип возвращаемого значения
- всегда член класса
- наследуется дочерними классами

Преобразование типов: пример

```
class Digit{
      int x_;
public:
      Digit( ){ x_ = 10; }
      ~Digit() { }
      operator int( ) const {
                  return x_;
      }
 };

void f ( ){
      Digit myDig;
      int  a = myDig;            // a = 10
}
```

Оператор индекса [ ]

- принимает один аргумент
- возвращает ссылочный тип
- реализуется как нестатический метод класса

```
double& Point::operator[ ](int index){
    if (index < 0 || index > 1){
            std::cout << "Error" << std::endl;
            exit(1);
    }
    if (index == 0){ return x_; }
    else { return y_; }
}
```

Оператор вызова функции ( )

- может принимать любое число аргументов
- может не возвращать значение
- реализуется как нестатический метод класса

```
void Point::operator( )(double x, double y){
    x_ = x;
    y_ = y;
}
```

Ключевое слово default

- указывает, что используется метод,
  по умолчанию генерируемый компилятором
- добавляется через = в объявление
- используется, если есть вариант по умолчанию

```
class Rectangle {
public:
    Rectangle() = default;
    Rectangle(double x, double y);
    ~Rectangle() = default;
    Rectangle(const Rectangle& r) = default;
    Rectangle& operator=(const Rectangle& r) = default;
};
```

Ключевое слово delete

- указывает компилятору, что не нужно генерировать объявленный метод
- добавляется через = в объявление
- может быть использовано с любой функцией

```
class Rectangle {
public:
    Rectangle() = delete;
    Rectangle(double x, double y);
    explicit Rectangle(double x) = delete;
    ~Rectangle();
    Rectangle(const Rectangle& r) = delete;
    Rectangle& operator=(const Rectangle& r) = delete;
};
```

Вопросы?