

# **Тема 6. Объектно-ориентированное программирование. Перегрузка и шаблоны**

## Перегрузка функций

Перегрузка функций - одна из форм полиморфизма, которая заключается в возможности использования в одном пространстве имен нескольких функций с одинаковым именем, но с разными параметрами.

<b>Язык C</b>	<code>int abs(int x)</code>	<code>double fabs(double x)</code>
<b>Язык C++</b>	<code>int abs(int x)</code>	<code>double abs(double x)</code>

## Перегрузка функций в C++

```
char abs(char x);  
short abs(short x);  
long abs(long x);  
float abs(float x);
```

*Язык C++ разрешает определение нескольких функций с одним и тем же именем, если функции отличаются числом или типом параметров.*

```
char abs(char x)  
{  
    return(x < 0 ? -x : x);  
}
```

```
short abs(short x)  
{  
    return(x < 0 ? -x : x);  
}
```

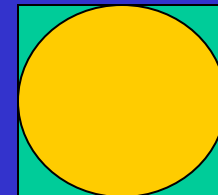
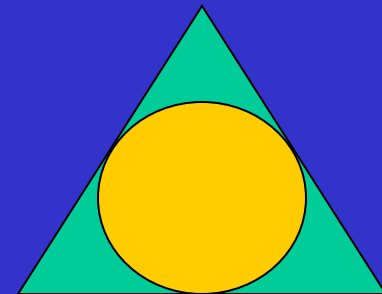
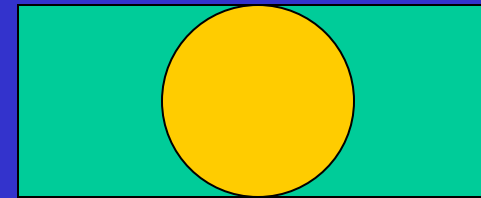
```
long abs(long x)  
{  
    return(x < 0 ? -x : x);  
}
```

```
float abs(float x)  
{  
    return(x < 0 ? -x : x);  
}
```

## Перегрузка методов в C++

```
class Circle
{
private:
    double r;
    ...
public:
    void    Fit(Rectangle &R);
    void    Fit(Triangle &T);
    void    Fit(Square &S);
};

void Circle::Fit(Square &S)
{
    r = S.Side() / 2;
}
```



## Перегрузка, переопределение, сокрытие

**Перегружаемые функции** должны находиться в одной области определения.

**Перегружаемые функции** различаются компилятором по их параметрам.

Посредством **перегрузки** вызываемая версия функции соответствует конкретным типам параметров.

**Переопределение** функции осуществляется новым описанием функции в другой области определения в иерархиях классов.

**Переопределяемые** функции имеют идентичные имена и типы параметров.

**Соккрытие** или "затенение" функций осуществляется для функций, находящихся в различных областях определения, причем функция, описанная во внутренней области, скрывает описание, данное во внешней области.

**Скрываемые и скрывающие** функции имеют идентичные имена.

Доступ к **скрытым** функциям осуществляется через оператор расширения области видимости.

# Пример перегрузки, переопределения, сокрытия

```
class Figure
{
public:
    double    Area();
    void     Draw();
};

class Circle : public Figure
{
public:
    double    Area();
    void     Draw(int x, int y);
    void     Fit(Rectangle &R);
    void     Fit(Triangle &T);
    void     Fit(Square &S);
};
```

```
Circle c(100);
c.Draw(120, 50);    // правильно
c.Draw();           // ошибка
c.Figure::Draw();  // правильно
```

Переопределение

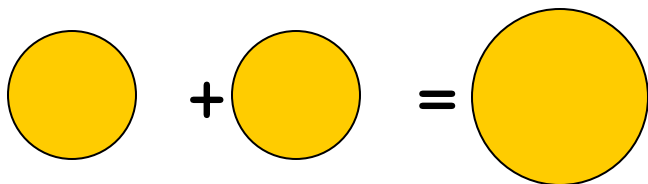
Сокрытие

Перегрузка

## Перегрузка операторов в C++

`2 + 2 = 4`

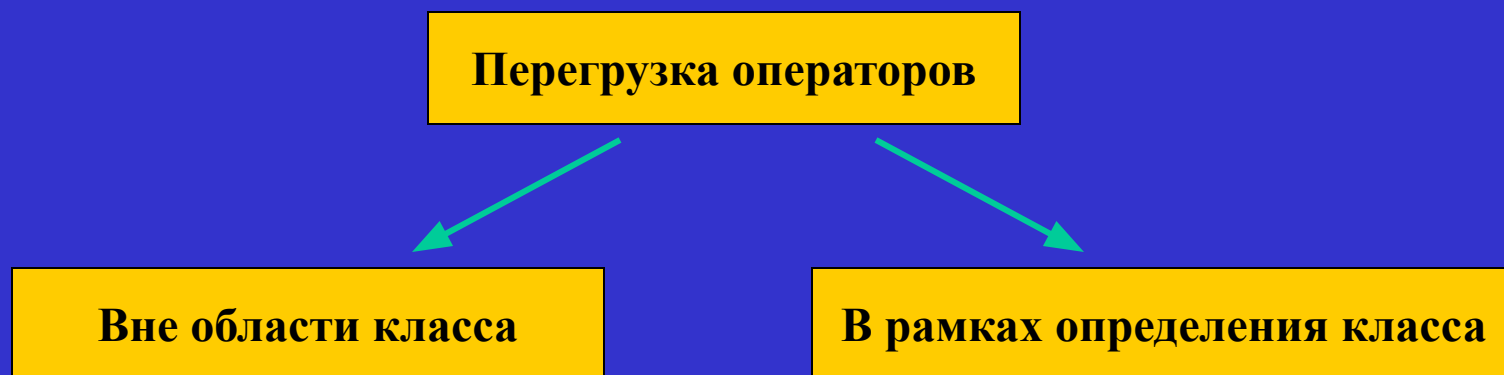
```
int a = 2;  
int b = 2;  
int c = a+b;
```



```
Circle a(2);  
Circle b(2);  
Circle c = a+b;  
  
Circle c = a.operator+(b);
```

## Правила перегрузки операторов

- Нельзя определять новые операторы
- Нельзя перегружать операторы `::` , `?:` , `.` , `.*` , `#` , `##`
- По крайней мере один из операндов перегруженного оператора должен быть объектом класса или ссылкой на объект класса
- Нельзя изменять общий синтаксис оператора (число операндов, приоритет, задаваемые аргументы)





## Пример перегрузки оператора «+» в области класса

```
class Circle
{
private:
    double r;
public:
    Circle(double R) { r = R; }

    Circle &operator+(Circle &C);
};

Circle &Circle::operator+(Circle &C)
{
    static Circle tmp(0);
    tmp.r = sqrt(r*r+C.r*C.r);
    return(tmp);
}
```

## Пример перегрузки оператора «+» вне области класса

```
Circle &operator+(Circle &C1, Circle &C2)
{
    static Circle C();
    C.r = sqrt(C1.r*C1.r+C2.r*C2.r);
    return C;
}
```

## Пример перегрузки оператора присваивания

```
class Circle
{
private:
    double r;
public:
    Circle(double R) { r = R; }

    Circle &operator=(Circle &C);
};

Circle &Circle::operator=(Circle &C)
{
    r = C.r;
    return(*this);
}
```

## Перегруженные операторы и цепочные вычисления

```
Circle a, b, c, d, e;
```

```
e = a+b+c+d;
```



```
e.operator=(a.operator+(b).operator+(c).operator+(d));
```

```
e = (a+b) + (c+d) ;
```



```
e.operator=(a.operator+(b).operator+(c.operator+(d)));
```

```
e = a+(b+c)+d;
```



```
e.operator=(a.operator+(b.operator+(c)).operator+(d));
```

## Множественная перегрузка операторов

```
class Circle
{
private:
    double r;
public:
    Circle(double R) { r = R; }

    Circle &operator=(Circle &C);
    Circle &operator=(Rectangle &R);
    Circle &operator=(Triangle &T);
    Circle &operator=(Square &S);
};

Circle &Circle::operator=(Square &S)
{
    r = sqrt(S.Area() / M_PI);
    return (*this);
}
```

## Пример перегрузки оператора >

```
class Circle
{
public:
    double    Area() { return(M_PI*r*r); }
    bool    operator>(Circle &C);
    bool    operator>(Rectangle &R);
    bool    operator>(Triangle &T);
    bool    operator>(Square &S);
};

bool Circle::operator>(Circle &C)
{
    return(r > C.r);
}

bool Circle::operator>(Rectangle &R)
{
    return(Area() > R.Area());
}
```

## Перегрузка операторов ++ и --

```
class Circle
{
private:
    double r;
public:
    Circle(double R) { r = R; }
    Circle& operator++();           // префиксная форма
    Circle& operator++(int);       // постфиксная форма
    Circle& operator--();          // префиксная форма
    Circle& operator--(int);       // постфиксная форма
};
```

```
Circle &operator++()
{
    r *= 2;
    return(*this);
}
```

```
Circle &operator++(int)
{
    static Circle tmp(0);
    tmp.r = r;
    r *= 2;
    return(tmp);
}
```

```
Circle a(10);
Circle b = a++;
Circle c = ++a;
```

## Шаблоны функций

```
char   min(char x, char y);  
short  min(short x, short y);  
long   min(long x, long y);  
float  min(float x, float y);  
double min(double x, double x);
```



```
template<class T>  
T min(T x, T y)  
{  
    return(x < y ? x : y);  
}  
  
long double min(long double x, long double y); // инсталляция
```



## Замещение шаблонов функций

```
template<class T>
T min(T x, T y)
{
    return(x < y ? x : y);
}
```



```
char* min(char* x, char* y)
{
    return(strcmp(x, y) < 0 ? x : y);
}
```

## Шаблоны классов

```
template<class T>
class Stack
{
private:
    T      *data;
    int    count;
public:
    Stack(int Number);
    ~Stack();

    void   Input(T Value);
    T      Output();
};
```

## Реализация методов шаблонов классов

```
template<class T>
Stack<T>::Stack(int Number)
{
    data = new T[Number];
}

template<class T>
Stack<T>::~~Stack()
{
    delete [] data;
}
```

```
template<class T>
void Stack<T>::Input(T
Value)
{
    data[count++] = Value;
}

template<class T>
T Stack<T>::Output()
{
    return (data[--count]);
}
```

## Использование шаблонов в программе

```
void main()
{
    Stack<int> istack(20);
    istack.Input(14);
    istack.Input(11);
    istack.Input(25);
    int i = istack.Output();

    Stack<float> fstack(10);
    fstack.Input(230.21);
    fstack.Input(18.513);
    fstack.Input(83.234);
    float f = fstack.Output();

    Stack<char*> cstack(24);
    cstack.Input("First");
    cstack.Input("Second");
    char* c = cstack.Output();
}
```

## Полное и частичное замещение шаблонов классов

```
template<class T1, class T2>
class Converter
{
    T1 obj1;
    T2 obj2;
    ...
};
```

**Полное замещение**

```
void main()
{
    class Converter<Rectangle, Circle> {...};

    template<class T>
    class Converter<T, Circle> {...};
}
```

**Частичное замещение**

## Создание новых типов на основе шаблонов классов

```
template<class T>
class Stack
{
private:
    T      *data;
    int    count;
public:
    Stack(int Number);
    ~Stack();
    ...
};

typedef Stack<Circle> StackOfCircles;
typedef Stack<Square> StackOfSquares;

StackOfCircles circles(100);
StackOfSquares squares(120);
```