Объектно-ориентированное программирование на алгоритмическом языке C++

МИРЭА, Институт Информационных технологий,
кафедра Вычислительной техники

Автор: доцент, канд. физ.-мат. наук,
Путуридзе Зураб Шотаевич

Перегрузка операторов

Перегрузка операторов – это механизм, который позволяет применять привычные знаки операции и операторов к объектам пользовательского класса реализуя специфические алгоритмы их обработки.

Для перегрузки операции или оператора создается оператор-функция.

Оператор-функция является методом класса или дружественной класса.

«тип» [«имя класса» ::] operator# (список параметров)

```
{
   // тело функции реализует специфический алгоритм операции
   // или оператора
}
```

Перегрузка операции

```
class coord {
    int x, y;
public:
    coord  ( ) { x = 0; y = 0; };
    coord  ( int i, int j ) { x = i; y = j; };
    void  get_xy ( int & i, int & j ) { i = x; j = y; };
    coord operator+ ( coord ob2 );
    friend coord operator- ( coord ob1, coord ob2 );
};
coord coord :: operator+ ( coord ob2 ) {
     coord  temp;
     temp.x = x + ob2.x;
     temp.y = y + ob2.y;
     return  temp;
}
```

Перегрузка операции

```
coord operator- ( coord ob1, coord ob2 ) {
    coord  temp;
    temp.x = ob1.x - ob2.x;
    temp.y = ob1.y - ob2.y;
    return  temp;
}
int main ( ) {
    setlocale ( LC_ALL, "Russian" );
    coord  ob1 ( 10, 10 ), ob2 ( 5, 5 );   // Параметризированный конструктор
    coord  ob3;                            // Конструктор по умолчанию
    int    x, y;
    ob3 = ob3 + ob1;
    ob3.get_xy ( x, y );
    cout  << "ob3 x = " << x << "    ob3 y = " << y << "\n";
    ob3 = ob3 - ob2;
    ob3.get_xy ( x, y );
    cout  << "ob3 x = " << x << "    ob3 y = " << y << "\n";
    return 0;
}
```

Перегрузка операции

```
Результат работы программы:
ob3 x = 10      ob3 y = 10
ob3 x = 5       ob3 y = 5
```

Перегрузка операций отношения и логических операций

Оператор-функция перегрузки операции отношения или логической операции должен возвращать целое, интерпретируемое как значение true или false.

Перегрузка операции

```
#include <cmath>
class coord {
    int x, y;
public:
    coord  ( ) { x = 0; y = 0; };
    coord  ( int i, int j ) { x = i; y = j; };
    int   operator== ( coord ob2 );
    int   operator<  ( coord ob2 );
    int   operator&& ( coord ob2 );
};
int coord :: operator== ( coord ob2 ) {
    return  ( x == ob2.x ) && ( y == ob2.y );
}
int coord :: operator< ( coord ob2 ) {
     return  sqrt ( x * x + y * y ) < sqrt ( ob2.x * ob2.x + ob2.y * ob2.y );
}
int coord :: operator&& ( coord ob2 ) {
    return  ( x && ob2.x ) && ( y && ob2.y );
}
```

Перегрузка операции

```
int main ( ) {
    setlocale ( LC_ALL, "Russian" );
    coord  ob1 ( 10, 10 );
    coord  ob2 (  5,  5 );
    cout  << "ob1 == ob2  :  " << ( ob1 == ob2 ) << "\n";
    cout  << "ob2 <  ob1  :  " << ( ob2 <  ob1 ) << "\n";
    cout  << "ob1 && ob2  :  " << ( ob1 && ob2 ) << "\n";
    return 0;
}
```

Результат работы программы:

```
ob1 == ob2  :  0
ob2 <  ob1  :  1
ob1 && ob2  :  1
```

Перегрузка унарных операций

При перегрузке унарной операции посредством оператор-функции метода класса ни один объект не передается явным образом.

Операции инкремента и декремента имеют как префиксную, так и постфиксную формы.

Прототип постфиксной формы имеет следующий вид:

<имя класса> <имя класса> :: operator++ (int notused);

Перегрузка операции

```
class coord {
    int x, y;
public:
    coord  ( ) { x = 0; y = 0; };
    coord  ( int i, int j ) { x = i; y = j; };
    void   get_xy ( int & i, int & j ) { i = x; j = y; };
    coord  operator++ ( );              // префикс
    coord  operator++ ( int notused );  // постфикс
};
coord coord :: operator++ ( ) {
    x ++;
    y ++;
    return  * this;
}
coord coord :: operator++ ( int notused ) {
    coord  temp = * this;
    x ++;
    y ++;
    return  temp;
}
```

Перегрузка операции

```
int main ( ) {
    setlocale ( LC_ALL, "Russian" );
    coord  ob1 ( 10, 10 );
    coord  ob2 (  5,  5 );
    coord  ob3;
    int    x, y;
    ++ ob1;
    ob1.get_xy ( x, y );
    cout  << "ob1 x = " << x << "    ob1 y = " << y << "\n";
    ob3 = ob2 ++;
    ob3.get_xy ( x, y );
    cout  << "ob3 x = " << x << "    ob3 y = " << y << "\n";
    return 0;
}
```

Результат работы программы:

```
ob1 x = 11    ob1 y = 11
ob3 x =  5    ob3 y =  5
```

Аргументы, передаваемые функции по умолчанию

```
void func ( double num = 0.0, char ch = 'X' )
{
     . . . . .
}

void func1 ( int i, char ch = 'X' );
```

Определение адреса перегруженной функции

```
void space ( int count ) {
    for ( ; count; count — ) cout << ' ';
}
void space ( int count, char ch ) {
    for ( ; count; count — ) cout << ch;
}

int main () {
    void ( * fp1 ) ( int );
    void ( * fp2 ) ( int, char );
    fp1 = space; // адреса функции space (int)
    fp2 = space; // адреса функции space (int, char)
    fp1 ( 22 );          // выводит 22 пробела
    cout << " | \n";
    fp2 ( 30, 'x' ); // выводит 30 символов x
    cout << "  \n";
    return 0;
}
```

Шаблонные функции

Описание шаблона функции

template <формальные параметры > «тип» «имя функции» (список параметров)

{

// тело функции

}

формальный параметр ::= class «имя параметра» |

typename «имя параметра» |

«тип» «имя параметра»

Пример шаблонной функции

```
template < class T >  T & inc_value ( T & val )  {
    ++val;   return val;
}

int main ( )  {
    int x = 0;
    x = ( int ) inc_value < int > ( x );
    cout << x << endl;
    char c = 0;
    c = ( char ) inc_value < char > ( c );
    cout << c << endl;
    return 0;
}
```

Пример шаблонной функции

```
template < class T1 >
void PrintArray ( const T1 * array, const int count )
{
    for ( int i = 0; i < count;  i++ )
        cout << array [ i ] << " ";
    cout << endl;
}
```

Пример шаблонной функции

```
int main ( ) {
    const int aCount = 5;
    const int bCount = 7;
    const int cCount = 6;
    int    a [ aCount ] = { 1,2,3,4,5 };
    double b [ bCount ] = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
    char   c [ cCount ] = "HELLO"; //6-я позиция для null
    cout << "Array a:" << endl;
    PrintArray ( a, aCount ); // шаблон для integer
    cout << "Array b:" << endl;
    PrintArray ( b, bCount );  // шаблон для double
    cout << "Array c:" << endl;
    PrintArray ( c, cCount );  // шаблон для character
    return 0;
}
```

Ответ примера

```
Array a:
1 2 3 4 5
Array b:
1.1 2.2 3.3 4.4 5.5 6.6 7.7
Array c:
H E L L O
```

```
void PrintArray ( const int*,    const int );
void PrintArray ( const double*, const int );
void PrintArray ( const char*,   const int );
```

Перегрузка шаблонных функций

```
template < class T1 >
void PrintArray ( const T1 * array, const int count)
{
    for ( int i = 0; i < count; i++ )
        cout << array [ i ] << " ";
    cout << endl;
}

template < class T1 >
void PrintArray ( const T1 * array,
                  const int  lowSubscript,
                  const int  highSubscript )
{
    for ( int i = lowSubscript; i <= highSubscript; i++ )
        cout << array [ i ] << " ";
    cout << endl;
}
```

Перегрузка шаблонных функций

```
int main ( )  {
    const int aCount = 5;
    const int bCount = 7;
    const int cCount = 6;
    int    a [ aCount ] = { 1,2,3,4,5 };
    double b [ bCount ] = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
    char   c [ cCount ] = "HELLO"; //6-я позиция для null
    …
    cout << "Array a from 1 to 3:" << endl;
    PrintArray ( a, 2 );          // шаблон для integer
    cout << "Array b from 4 to 7:" << endl;
    PrintArray ( b, 3, 6 );       // шаблон для double
    cout << "Array c from 3 to 5:" << endl;
    PrintArray ( c, 2, 4 );       // шаблон для character
    return 0;
}
```