Перегрузка операций
(продолжение)

Перегрузка операторов

"Операторный" метод имеет следующий формат:

```
<Тип результата> operator<Название оператора>(
      [<Тип> <Название параметра1>[, ..., <Тип> <Название параметра N>]])
{
   // Тело метода
}
```

Перегрузка операторов

Название "операторного" метода состоит из слова "operator", после которого указывается перегружаемый оператор, например, `operator+`, `operator=` и т. д. Внутри круглых скобок указывается тип, объект которого может быть расположен в выражении справа от перегружаемого оператора. Следовательно для каждого типа, с которым может взаимодействовать экземпляр класса внутри выражения, необходимо создать отдельный "операторный" метод. Например, чтобы сложить объект с целым числом следует создать метод `operator+(int x)`, а для сложения с вещественным числом — метод `operator+(double x)`. Внутри метода доступен указатель `this` на объект, который расположен слева от оператора. Таким образом, внутри метода можно напрямую обращаться к атрибутам и методам класса. Пример перегрузки оператора + для сложения объекта с целым числом и возвратом нового объекта:

Перегрузка операторов

```
C C::operator+(int x) {      // Объект класса C + целое число
   C newobj;                 // Создаем временный объект


   newobj.x_ = x_ + x;       // x_ - закрытый член класса C
   return newobj;            // Возвращаем временный объект
}
```

Метод будет вызван в следующей ситуации:

```
obj2 = obj1 + 20;            // C obj1, obj2;
```

Перегрузка операторов

Однако, если будет следующая ситуация, то метод вызван не будет, так как объект должен быть расположен слева от оператора, а не справа:

```
obj2 = 20 + obj1;            // Метод C::operator+(int x) не вызывается!
```

Перегрузка операторов

Перегрузить оператор + в этом случае позволяют дружественные функции. Параметры дружественной функции передаются явным образом. Объект, расположенный слева от оператора, доступен через первый параметр, а объект, расположенный справа от оператора, доступен через второй параметр. Объявление дружественной функции внутри объявления класса выглядит так:

```
friend C operator+(int x, C &obj);
```

Перегрузка операторов

Пример определения дружественной функции вне класса:

```
C operator+(int x, C &obj) {        // Целое число + объект класса C
    C newobj;
    newobj.x_ = obj.x_ + x;         // Обращение к x_ через параметр
    return newobj;
}
```

Перегрузка операторов

С помощью дружественной функции можно также заменить "операторный" метод. Тем не менее, стоит отдать предпочтение "операторным" методам, а дружественные функции применять в случаях, когда обойтись методом нельзя. В качестве примера перегрузим оператор + для сложения объекта с целым числом:

```
C operator+(C &obj, int x) {       // Объект класса C + целое число
    C newobj;
    newobj.x_ = obj.x_ + x;
    return newobj;
}
```

Перегрузка бинарных операторов

При использовании бинарных операторов в выражении участвуют два операнда. Примеры бинарных операторов: + (плюс), – (минус), * (умножение), / (деление), % (остаток от деления), & (двоичное И), | (двоичное ИЛИ), ^ (двоичное исключающее ИЛИ), << (сдвиг влево), >> (сдвиг вправо) и оператор "запятая". К бинарным операторам также относятся операторы сравнения: == (равно), != (неравно), < (меньше), > (больше), <= (меньше или равно), >= (больше или равно), && (логическое И) и || (логическое ИЛИ).

Перегрузка бинарных операторов

При перегрузке бинарных операторов через единственный параметр в "операторном" методе доступен операнд, расположенный от оператора справа, а указатель на сам объект, расположенный от оператора слева, передается неявным образом. Таким образом внутри метода ко всем членам класса можно обращаться напрямую или через указатель `this`. Дружественной функции передаются два параметра. Через первый параметр доступен операнд, расположенный от оператора слева, а через второй параметр доступен операнд, расположенный от оператора справа. Тип возвращаемого значения, а также внутренняя реализация зависит от предпочтений программиста. Обратите внимание на то, что в обычных выражениях при использовании бинарных операторов возвращается новое значение, а значения операндов внутри выражения не изменяются. Хотя это не правило, а лишь рекомендация. В качестве примера перегрузим операторы + и == (листинг 10.1).

Листинг 10.1. Перегрузка бинарных операторов

```
#include <iostream>

class C {
   int x_;
public:
   C() { x_ = 0; }                    // Конструктор по умолчанию


   explicit C(int x) { x_ = x; }     // Обычный конструктор
   C(const C &c) { x_ = c.x_; }      // Конструктор копирования
   int get_x() { return x_; }
   C operator+(const C &obj);                      // obj1 + obj2
   C operator+(int x);                             // obj + целое число
   friend C operator+(int x, const C &obj);        // целое число + obj
   bool operator==(const C &obj);                  // obj1 == obj2
   bool operator==(int x);                         // obj == целое число
   friend bool operator==(int x, const C &obj); // целое число == obj
};
```

Листинг 10.1. Перегрузка бинарных операторов

```
int main() {
    C obj1(10), obj2(20), obj3;
    obj3 = obj1 + obj2;
    std::cout << obj3.get_x() << std::endl;     // 30
    obj3 = obj3 + 50;
    std::cout << obj3.get_x() << std::endl;     // 80
    obj2 = 20 + obj3;
    std::cout << obj2.get_x() << std::endl;     // 100
    std::cout << (obj1 == obj2) << std::endl;   // 0
    std::cout << (obj1 == 10) << std::endl;     // 1
    std::cout << (100 == obj2) << std::endl;    // 1
    std::cin.get();
    return 0;
}
```

Листинг 10.1. Перегрузка бинарных операторов

```
C C::operator+(const C &obj) {          // obj1 + obj2
   C newobj;
   newobj.x_ = x_ + obj.x_;
   return newobj;
}
C C::operator+(int x) {                 // obj + целое число
   C newobj;
   newobj.x_ = x_ + x;
   return newobj;
}
```

Листинг 10.1. Перегрузка бинарных операторов

```
C operator+(int x, const C &obj) {      // целое число + obj
   C newobj;
   newobj.x_ = obj.x_ + x;
   return newobj;
}
bool C::operator==(const C &obj) {      // obj1 == obj2
   return (x_ == obj.x_);
}
bool C::operator==(int x) {             // obj == целое число
   return (x_ == x);
}
bool operator==(int x, const C &obj) {  // целое число == obj
   return (x == obj.x_);
}
```

Перегрузка унарных операторов

При использовании унарных операторов в выражении участвует только один операнд — сам объект. Примеры унарных операторов: + (унарный плюс), - (унарный минус), * (разыменование), & (взятие адреса), ~ (двоичная инверсия) и ! (логическое отрицание).

Перегрузка унарных операторов

При перегрузке унарных операторов "операторный" метод не принимает параметров. Внутри метода ко всем членам объекта можно обращаться напрямую или через указатель this. Дружественной функции передается один параметр — объект класса. В качестве примера перегрузим унарный минус с помощью "операторного" метода, а оператор ! с помощью дружественной функции (листинг 10.2).

Листинг 10.2. Перегрузка унарных операторов

```
#include <iostream>

class C {
   int x_;
public:
   C() { x_ = 0; }                    // Конструктор по умолчанию
   explicit C(int x) { x_ = x; }      // Обычный конструктор
   C(const C &c) { x_ = c.x_; }       // Конструктор копирования

   int get_x() { return x_; }
   C operator-();                     // -obj
   friend bool operator!(const C &obj);    // !obj
};
```

Листинг 10.2. Перегрузка унарных операторов

```
int main() {
    C obj1(10), obj2;
    obj2 = -obj1;
    std::cout << obj2.get_x() << std::endl; // -10
    std::cout << (!obj1) << std::endl;      // 0
    std::cin.get();
    return 0;
}

C C::operator-() {                  // -obj
    C newobj;
    newobj.x_ = -x_;
    return newobj;
}
bool operator!(const C &obj) {      // !obj
    return !(obj.x_);
}
```

Перегрузка операторов инкремента и декремента

Операторы инкремента (++) и декремента (--) также являются унарными операторами, но они изменяют сам объект. Кроме того, операторы могут использоваться в постфиксной или префиксной формах. При постфиксной форме (x++) возвращается значение переменной перед операцией, а при префиксной форме (++x) — вначале производится операция и только потом возвращается значение.

Перегрузка операторов инкремента и декремента

Прототипы "операторных" методов:

```
<Название класса> &operator++();       // ++obj
<Название класса> &operator--();       // --obj
<Название класса> operator++(int);     // obj++
<Название класса> operator--(int);     // obj--
```

Перегрузка операторов инкремента и декремента

Прототипы дружественных функций:

```
friend <Название класса> &operator++(<Название класса> &);      // ++obj
friend <Название класса> &operator--(<Название класса> &);      // --obj
friend <Название класса> operator++(<Название класса> &, int); // obj++
friend <Название класса> operator--(<Название класса> &, int); // obj--
```

Перегрузка операторов инкремента и декремента

Целочисленный параметр, используемый в прототипах при постфиксной форме, можно проигнорировать. По умолчанию этот параметр имеет значение 0. Пример перегрузки оператора инкремента приведен в листинге 10.3.

Листинг 10.3. Перегрузка оператора инкремента

```
#include <iostream>

class C {
   int x_;
public:
   C() { x_ = 0; }                          // Конструктор по умолчанию
   explicit C(int x) { x_ = x; }            // Обычный конструктор
   C(const C &c) { x_ = c.x_; }             // Конструктор копирования
   int get_x() { return x_; }
   C &operator++();                         // ++obj
   C operator++(int x);                     // obj++
};
```

Листинг 10.3. Перегрузка оператора инкремента

```
int main() {
    C obj1(10), obj2;
    obj2 = ++obj1;
    std::cout << obj1.get_x() << std::endl; // 11
    std::cout << obj2.get_x() << std::endl; // 11
    obj2 = obj1++;
    std::cout << obj1.get_x() << std::endl; // 12
    std::cout << obj2.get_x() << std::endl; // 11
    ++obj2;
    std::cout << obj1.get_x() << std::endl; // 12
    std::cout << obj2.get_x() << std::endl; // 12
    std::cin.get();
    return 0;
```

Листинг 10.3. Перегрузка оператора инкремента

```
}

C &C::operator++() {                          // ++obj
    ++x_;
    return *this;
}
C C::operator++(int x) {                      // obj++
    C tmp = *this;
    ++*this;
    return tmp;
}
```

Перегрузка операторов присваивания

Перегрузка оператора = и сокращенных операторов присваивания (например, +=, -=, *=, /= и др.) осуществляется также как и перегрузка бинарных операторов. Главное отличие заключается в том, что операторы присваивания изменяют сам объект, а не возвращают новый. При использовании "операторных" методов ссылка на объект, расположенный слева от оператора присваивания, передается неявным образом, а объект, расположенный справа от оператора, доступен через параметр. Дружественной функции передаются два параметра. Через первый параметр доступен объект, расположенный от оператора слева, а через второй параметр доступен объект, расположенный от оператора справа. В качестве результата обычно возвращается указатель на исходный объект.

Перегрузка операторов присваивания

При перегрузке оператора = следует учитывать:

- если оператор не перегружен, то автоматически создается оператор приваивания по умолчанию, который создает побитовую копию объекта. Это важно учитывать, если внутри класса производится динамическое выделение памяти;
- при использовании оператора присваивания при инициализации объекта вызывается конструктор копирования, а не перегруженная версия оператора =;
- перегрузить оператор = с помощью дружественной функции нельзя.

Пример перегрузки операторов присваивания приведен в листинге 10.4.

Листинг 10.4. Перегрузка операторов присваивания

```
#include <iostream>

class C {
   int x_;
public:
   C() { x_ = 0; }                    // Конструктор по умолчанию
   explicit C(int x) { x_ = x; }      // Обычный конструктор
   C(const C &c) { x_ = c.x_; }       // Конструктор копирования
   int get_x() { return x_; }
   C &operator=(C &obj);              // obj2 = obj1
   C &operator=(int x);               // obj  = целое число
   C &operator+=(int x);              // obj += целое число
};
```

Листинг 10.4. Перегрузка операторов присваивания

```
int main() {
    C obj1, obj2;
    obj1 = 10;                                  // obj  = целое число
    obj2 = obj1;                                // obj2 = obj1
    obj1 = obj1;                                // obj = obj
    obj2 += 20;                                 // obj += целое число
    std::cout << obj1.get_x() << std::endl; // 10
    std::cout << obj2.get_x() << std::endl; // 30
     // Множественное присваивание
    obj2 = obj1 = 5;
    std::cout << obj1.get_x() << std::endl; // 5
    std::cout << obj2.get_x() << std::endl; // 5
    // Вызывается конструктор копирования,
    // а не метод C &operator=(C &obj)!!!
    C obj3 = obj1;
    std::cout << obj3.get_x() << std::endl; // 5
    std::cin.get();
    return 0;
}
```

Листинг 10.4. Перегрузка операторов присваивания

```
C &C::operator=(C &obj) {                    // obj2 = obj1
   if (this == &obj) return *this;           // Случай obj = obj
   x_ = obj.x_;
   return *this;
}
C &C::operator=(int x) {                     // obj = целое число
   x_ = x;
   return *this;
}
C &C::operator+=(int x) {                    // obj += целое число
   x_ += x;
   return *this;
}
```

Перегрузка оператора ( )

Перегрузка оператора () позволяет обработать вызов экземпляра класса как вызов функции. Количество параметров и тип возвращаемого значения в "операторном" методе могут быть произвольными. Внутри метода ко всем членам объекта можно обращаться напрямую или через указатель this. Обратите внимание на то, что перегрузить оператор () с помощью дружественной функции нельзя. Пример перегрузки оператора () приведен в листинге 10.5.

Листинг 10.5. Перегрузка оператора ( )

```
#include <iostream>

class C {
   int x_;
public:
   C() { x_ = 0; }                    // Конструктор по умолчанию
   explicit C(int x) { x_ = x; }      // Обычный конструктор
   int get_x() { return x_; }
   void operator()(int x) { x_ = x; }
   int operator()() { return x_; }
};
```

Листинг 10.5. Перегрузка оператора ( )

```
int main() {
    C obj;
    obj(10);
    std::cout << obj() << std::endl;          // 10
    std::cout << (20 + obj()) << std::endl; // 30
    std::cin.get();
    return 0;
}
```

Перегрузка оператора [ ]

Перегрузка оператора [] позволяет обработать доступ к элементу по индексу, причем индекс может быть не только целочисленным. При перегрузке оператора [] через единственный параметр в "операторном" методе доступен индекс, расположенный внутри квадратных скобок. Внутри метода ко всем членам класса можно обращаться напрямую или через указатель this. Чтобы можно было присвоить значение элементу, расположенному по указанному индексу, необходимо внутри метода возвратить ссылку на элемент. Обратите внимание на то, что перегрузить оператор [] с помощью дружественной функции нельзя. Пример перегрузки оператора [] приведен в листинге 10.6.

Листинг 10.6. Перегрузка оператора [ ]

```
#include <iostream>
#include <cstdlib>

class C {
   static const int size_ = 2;
   int arr_[size_];
public:
   C() { arr_[0] = 0; arr_[1] = 0; }
   C(int x, int y) { arr_[0] = x; arr_[1] = y; }
   int &operator[](int i);
};

int &C::operator[](int i) {
   if (i < 0 || i >= size_) { // Проверка выхода за границы массива


      std::cout << "Error" << std::endl;
      std::exit(1);
   }
   return arr_[i];
}
```

Листинг 10.6. Перегрузка оператора [ ]

```
int main() {
    C obj1, obj2(30, 40);
    obj1[0] = 10;
    obj1[1] = 20;
    std::cout << obj1[0] << std::endl;              // 10
    std::cout << (obj1[1] + obj2[0]) << std::endl; // 50
    std::cin.get();
    return 0;
}
```

Перегрузка оператора доступа к члену класса

Перегрузка оператора -> существляется также как и перегрузка унарных операторов. В качестве значения "операторный" метод должен возвращать указатель this. Обратите внимание на то, что перегрузить оператор -> с помощью дружественной функции нельзя. Пример перегрузки оператора -> приведен в листинге 10.7.

Листинг 10.7. Перегрузка оператора ->

```
#include <iostream>

class C {
   int x_;
public:
   int y;
   C(int a, int b) { x_ = a; y = b; }
   C *operator->() { return this; }
};
```

Листинг 10.7. Перегрузка оператора ->

```
int main() {
    C obj(30, 40);

    // obj->x_ = 10; // Ошибка. x_ - закрытый член класса
    obj->y = 20;     // OK. y - открытый член класса
    std::cout << obj->y << std::endl; // 20
    std::cin.get();
    return 0;
}
```

Перегрузка операторов << и >>

В языке C++ операторы << и >> перегружены для вывода и ввода данных соответственно. Чтобы объекты пользовательского класса имели возможность взамодействовать с потоками ввода/вывода следует перегрузить эти операторы с помощью дружественных функций. Использовать в этом случае "операторные" методы нельзя. Перегрузка оператора << осуществляется следующим образом:

```
std::ostream &operator<<(std::ostream &stream, <Класс> &obj) {
    // Выводим данные в поток используя переменную stream
    return stream;                    // Возвращаем ссылку на поток
}
```

Перегрузка операторов << и >>

Перегрузка оператора >> выглядит так:

```
std::istream &operator>>(std::istream &stream, <Класс> &obj) {

    // Вводим данные из потока используя переменную stream
    return stream;                    // Возвращаем ссылку на поток
}
```

В качестве параметров функции принимают ссылку на соответствующий поток и ссылку на объект. Внутри функции необходимо вернуть ссылку на поток, чтобы иметь возможность составлять цепочки из вызовов операторов. Пример перегрузки операторов << и >> показан в листинге 10.9.

Листинг 10.9. Перегрузка операторов << и >>

```
#include <iostream>

class C {
   int x_;
public:
   int y;
   C(int a, int b) { x_ = a; y = b; }
   friend std::ostream &operator<<(std::ostream &stream, C &obj);
   friend std::istream &operator>>(std::istream &stream, C &obj);
};

std::ostream &operator<<(std::ostream &stream, C &obj) {
   stream << "Class C { x_ = " << obj.x_ << ", y = "
          << obj.y << " }";
   return stream;                          // Возвращаем ссылку на поток
}
```

Листинг 10.9. Перегрузка операторов << и >>

```
std::istream &operator>>(std::istream &stream, C &obj) {
    std::cout << "y = ";
    stream >> obj.y;
    return stream;                          // Возвращаем ссылку на поток
}
int main() {
    C obj(10, 20);
    std::cin >> obj;                        // Например, ввели число 500
    std::cout << obj << std::endl;          // Результат:
                                            // Class C { x_ = 10, y = 500 }


  std::cin.ignore(255, '\n').get();
  return 0;
}
```

Листинг 10.9. Перегрузка операторов << и >>

```
std::istream &operator>>(std::istream &stream, C &obj) {
    std::cout << "y = ";
    stream >> obj.y;
    return stream;                          // Возвращаем ссылку на поток
}
int main() {
    C obj(10, 20);
    std::cin >> obj;                        // Например, ввели число 500
    std::cout << obj << std::endl;          // Результат:
                                            // Class C { x_ = 10, y = 500 }


  std::cin.ignore(255, '\n').get();
  return 0;
}
```