**Перегрузка операторов**

Перегрузка операторов позволяет определить действия, которые будет выполнять оператор. Перегрузка подразумевает создание функции, название которой содержит слово operator и символ перегружаемого оператора. Функция оператора может быть определена как член класса, либо вне класса.

Перегрузить можно только те операторы, которые уже определены в C++. Создать новые операторы нельзя.

Если функция оператора определена как отдельная функция и не является членом класса, то количество параметров такой функции совпадает с количеством операндов оператора. Например, у функции, которая представляет унарный оператор, будет один параметр, а у функции, которая представляет бинарный оператор, - два параметра. Если оператор принимает два операнда, то первый операнд передается первому параметру функции, а второй операнд - второму параметру. При этом как минимум один из параметров должен представлять тип класса

Рассмотрим пример с классом Counter, который представляет секундомер и хранит количество секунд:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | #include <iostream>    class Counter  {  public:      Counter(int sec)      {          seconds = sec;      }      void display()      {          std::cout << seconds << " seconds" << std::endl;      }  int seconds;  };    Counter operator + (Counter c1, Counter c2)  {      return Counter(c1.seconds + c2.seconds);  }    int main()  {      Counter c1(20);      Counter c2(10);      Counter c3 = c1 + c2;      c3.display();   // 30 seconds        return 0;  } |

Здесь функция оператора не является частью класса Counter и определена вне его. Данная функция перегружает оператор сложения для типа Counter. Она является бинарной, поэтому принимает два параметра. В данном случае мы складываем два объекта Counter. Возвращает функция также объект Counter, который хранит общее количесто секунд. То есть по сути здесь операция сложения сводится к сложению секунд обоих объектов:

|  |  |
| --- | --- |
| 1  2  3  4 | Counter operator + (Counter c1, Counter c2)  {      return Counter(c1.seconds + c2.seconds);  } |

При этом необязательно возвращать объект класса. Это может быть и объект встроенного примитивного типа. И также мы можем определять дополнительные перегруженные функции операторов:

|  |  |
| --- | --- |
| 1  2  3  4 | int operator + (Counter c1, int s)  {      return c1.seconds + s;  } |

Данная версия складывает объект Counter с числом и возвращает также число. Поэтому левый операнд операции должен представлять тип Counter, а правый операнд - тип int. И, к примеру, мы можем применить данную версию оператора следующим образом:

|  |  |
| --- | --- |
| 1  2  3 | Counter c1(20);  int seconds = c1 + 25;  // 45  std::cout << seconds << std::endl; |

Также функции операторов могут быть определены как члены классов. Если функция оператора определена как член класса, то левый операнд доступен через указатель this и представляет текущий объект, а правый операнд передается в подобную функцию в качестве единственного параметра:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | #include <iostream>    class Counter  {  public:      Counter(int sec)      {          seconds = sec;      }      void display()      {          std::cout << seconds << " seconds" << std::endl;      }      Counter operator + (Counter c2)      {          return Counter(this->seconds + c2.seconds);      }      int operator + (int s)      {          return this->seconds + s;      }  int seconds;  };    int main()  {      Counter c1(20);      Counter c2(10);      Counter c3 = c1 + c2;      c3.display();           // 30 seconds      int seconds = c1 + 25;  // 45        return 0;  } |

В данном случае к левому операнду в функциях операторов мы обращаемся через указатель this.

Какие операторы где переопределять? Операторы присвоения, индексирования ([]), вызова (()), доступа к члену класса по указателю (->) следует определять в виде функций-членов класса. Операторы, которые изменяют состояние объекта или непосредственно связаны с объектом (инкремент, декремент,), обычно также определяются в виде функций-членов класса. Все остальные операторы чаще определяются как отдельные функции, а не члены класса.

**Операторы сравнения**

Ряд операторов перегружаются парами. Например, если мы определяем оператор ==, то необходимо также определить и оператор !=. А при определении оператора < надо также определять функцию для оператора >. Например, перегрузим данные операторы:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | bool operator == (Counter c1, Counter c2)  {      return c1.seconds == c2.seconds;  }  bool operator != (Counter c1, Counter c2)  {      return c1.seconds != c2.seconds;  }  bool operator > (Counter c1, Counter c2)  {      return c1.seconds > c2.seconds;  }  bool operator < (Counter c1, Counter c2)  {      return c1.seconds < c2.seconds;  }  int main()  {      Counter c1(20);      Counter c2(10);      bool b1 = c1 == c2;     // false      bool b2 = c1 > c2;       // true        std::cout << b1 << std::endl;      std::cout << b2 << std::endl;        return 0;  } |

**Операторы присвоения**

Операторы присвоения обычно возвращает ссылку на свой левый операнд:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | #include <iostream>    class Counter  {  public:      Counter(int sec)      {          seconds = sec;      }      void display()      {          std::cout << seconds << " seconds" << std::endl;      }      Counter& operator += (Counter c2)      {          seconds += c2.seconds;          return \*this;      }  int seconds;  };  int main()  {      Counter c1(20);      Counter c2(10);      c1 += c2;      c1.display();   // 30 seconds      return 0;  } |

Операции инкремента и декремента

Особую сложность может представлять переопределение операций инкремента и декремента, поскольку нам надо определить и префиксную, и постфиксную форму для этих операторов. Определим подобные операторы для типа Counter:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | #include <iostream>    class Counter  {  public:      Counter(int sec)      {          seconds = sec;      }      void display()      {          std::cout << seconds << " seconds" << std::endl;      }      // префиксные операторы      Counter& operator++ ()      {          seconds += 5;          return \*this;      }      Counter& operator-- ()      {          seconds -= 5;          return \*this;      }      // постфиксные операторы      Counter operator++ (int)      {          Counter prev = \*this;          ++\*this;          return prev;      }      Counter operator-- (int)      {          Counter prev = \*this;          --\*this;          return prev;      }      int seconds;  };  int main()  {      Counter c1(20);      Counter c2 = c1++;      c2.display();       // 20 seconds      c1.display();       // 25 seconds      --c1;      c1.display();       // 20 seconds      return 0;  } |

Префиксные операторы должны возвращать ссылку на текущий объект, который можно получить с помощью указателя this:

|  |  |
| --- | --- |
| 1  2  3  4  5 | Counter& operator++ ()  {      seconds += 5;      return \*this;  } |

В самой функции можно определить некоторую логику по инкременту значения. В данном случае количество секунд увеличивается на 5.

Постфиксные операторы должны возвращать значение объекта до инкремента, то есть предыдущее состояние объекта. Чтобы постфиксная форма отличалась от префиксной постфиксные версии получают дополнительный параметр типа int, который не используется. Хотя в принципе мы можем его использовать.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Counter operator++ (int)  {      Counter prev = \*this;      ++\*this;      return prev;  } |