C++: Перегрузка операций

В языке C++ определены множества операций над переменными стандартных типов, такие как +, -, *, / и т.д.

Каждую операцию можно применить к операндам определенного типа.

Лишь ограниченное число типов непосредственно поддерживается любым языком программирования.

C и C++ не позволяют выполнять операции с комплексными числами, матрицами, строками, множествами.

Перегрузка операторов (operator overloading) позволяет определить для объектов классов встроенные операторы, такие как +, -, * и т.д.

Пусть заданы множества A и B:
A = { a1, a2, a3 };
B = { a3, a4, a5 };
нужно выполнить операции объединения (+) и пересечения (*) множеств.
A + B = { a1, a2, a3, a4, a5 }
A * B = { a3 }.

Можно определить класс Set - "множество" и определить операции над объектами этого класса, выразив их с помощью знаков операций, которые уже есть в языке C++, например, + и *.

В результате операции + и * можно будет использовать как и раньше, а также снабдить их дополнительными функциями (объединения и пересечения).

C++: Перегрузка операций

Как определить, какую функцию должен выполнить оператор: старую или новую?
По типу операндов.

А как быть с приоритетом операций?
Сохраняется определенный ранее приоритет операций.

Для распространения действия операции на новые типы данных надо определить специальную функцию, 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.

Нельзя изменить количество операндов, их ассоциативность, приоритет.

C++: Перегрузка операций

Если функция оператора определена как отдельная функция и не является членом класса, то количество параметров такой функции совпадает с количеством операндов оператора.

У функции, которая представляет унарный оператор (унарный минус), 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.

Определение операторов в виде функций-членов класса:

```
1 // бинарный оператор
2 ReturnType operator Op(Type right_operand);
3 // унарный оператор
4 ClassType& operator Op();
```

ClassType – класс, для которого определяется оператор;
Type – тип другого операнда;
ReturnType – тип возвращаемого результата, который может совпадать с типом операнда, а может и отличаться;
Op – операция.

C++: Перегрузка операций

Определение операторов в виде функций, которые не являются членами класса:

```
// бинарный оператор
ReturnType operator Op(const ClassType& left_operand, Type right_operand);
// альтернативное определение, где класс, для которого создается оператор, представляет правый операнд
ReturnType operator Op(Type left_operand, const ClassType& right_operand);
// унарный оператор
ClassType& operator Op(ClassType& obj);
```

пример функции члена класса Counter, который хранит некоторое число:

```
class Counter
{
public:
    Counter(int val)
    {
        value =val;
    }
    void print()
    {
        std::cout << "Value: " << value << std::endl;
    }
    Counter operator + (const Counter& counter) const
    {
        return Counter{value + counter.value};
    }
private:
    int value;
};
```

Результатом оператора сложения является новый объект Counter, в котором значение value равно сумме значений value обоих операндов.

```
int main()
{
    Counter c1{20};
    Counter c2{10};
    Counter c3 = c1 + c2;
    c3.print();   // Value: 30
}
```

После определения оператора можно складывать два объекта Counter

оператор сложения, цель которого сложить два объекта Counter

объект, который передается в функцию через параметр counter, будет представлять правый операнд операции.

текущий объект будет представлять левый операнд операции

C++: Перегрузка операций

пример функции вне класса:

```
3  class Counter
4  {
5  public:
6      Counter(int val)
7      {
8          value =val;
9      }
10     void print()
11     {
12         std::cout << "Value: " << value << std::endl;
13     }
14     int value; // к приватным переменным внешняя функция оператора не может обращаться
15 };
16 // определяем оператор сложения вне класса
17 Counter operator + (const Counter& c1, const Counter& c2)
18 {
19     return Counter{c1.value + c2.value};
20 }
21
22 int main()
23 {
24     Counter c1{20};
25     Counter c2{10};
26     Counter c3 {c1 + c2};
27     c3.print();   // Value: 30
28 }
```

внешняя функция не может обращаться к приватным полям класса, поэтому переменная value сделана публичной

бинарный оператор определяется в виде внешней функции, поэтому передаётся два параметра

первый параметр будет представлять левый операнд операции

второй параметр - правый операнд

C++: Перегрузка операций

Функция оператора необязательно должна возвращать объект класса.

Это может быть любой объект.

Также можно определять дополнительные перегруженные функции операторов.

```
3  class Counter
4  {
5  public:
6      Counter(int val)
7      {
8          value =val;
9      }
10     void print()
11     {
12         std::cout << "Value: " << value << std::endl;
13     }
14     Counter operator + (const Counter& counter) const
15     {
16         return Counter{value + counter.value};
17     }
18     int operator + (int number) const
19     {
20         return value + number;
21     }
22 private:
23     int value;
24 };
```

```
27 int main()
28 {
29     Counter counter{20};
30     int number = counter + 30;
31     std::cout << number << std::endl;   // 50
32 }
```

левый операнд операции должен представлять тип Counter

правый операнд - тип int

определена вторая версия оператора сложения, которая складывает объект Counter с числом и возвращает также число

C++: Перегрузка операций

Какие операторы где определять?

Операторы присвоения, индексирования [], вызова (), доступа к указателю ->, инкремент ++, декремент -- определяются в виде функций-членов класса.

Операторы выделения и удаления памяти new, delete определяются в виде функций, которые не являются членами класса.

Остальные операторы можно определять в виде функций, которые не являются членами класса.

C++: Перегрузка операций

Операторы сравнения ==, !=, <, >

Результатом операторов сравнения, как правило, является значение типа **bool**.

```
class Counter
{
public:
    Counter(int val)
    {
        value =val;
    }
    void print()
    {
        std::cout << "Value: " << value << std::endl;
    }
    bool operator == (const Counter& counter) const
    {
        return value == counter.value;
    }
    bool operator != (const Counter& counter) const
    {
        return value != counter.value;
    }
    bool operator > (const Counter& counter) const
    {
        return value > counter.value;
    }
    bool operator < (const Counter& counter) const
    {
        return value < counter.value;
    }
private:
    int value;
};
```

Если используется простое сравнение полей класса, то для операторов == и != можно использовать специальный оператор default.

По умолчанию будут сравниваться все поля класса, для которых определен оператор ==.

```
class Counter
{
public:
    Counter(int val)
    {
        value =val;
    }
    void print()
    {
        std::cout << "Value: " << value << std::endl;
    }
    bool operator == (const Counter& counter) const = default;
    bool operator != (const Counter& counter) const = default;
private:
    int value;
};
```

```
int main()
{
    Counter c1(20);
    Counter c2(10);
    bool b1 = c1 == c2;     // false
    bool b2 = c1 > c2;   // true

    std::cout << "c1 == c2 = " << std::boolalpha << b1 << std::endl;    // c1 == c2 = false
    std::cout << "c1 > c2 = " << std::boolalpha << b2 << std::endl;     // c1 > c2 = true
}
```

C++: Перегрузка операций

Оператор присвоения +=

Оператор присвоения возвращает ссылку на левый операнд

```
class Counter
{
public:
    Counter(int val)
    {
        value =val;
    }
    void print()
    {
        std::cout << "Value: " << value << std::endl;
    }
    // оператор присвоения
    Counter& operator += (const Counter& counter)
    {
        value += counter.value;
        return *this;   // возвращаем ссылку на текущий объект
    }
private:
    int value;
};

int main()
{
    Counter c1{20};
    Counter c2{50};
    c1 += c2;
    c1.print();     // Value: 70
}
```

Унарные операции -

Унарные операции обычно возвращают новый объект, созданный на основе имеющегося.

```
class Counter
{
public:
    Counter(int val)
    {
        value =val;
    }
    void print()
    {
        std::cout << "Value: " << value << std::endl;
    }
    // оператор унарного минуса
    Counter operator - () const
    {
        return Counter{-value};
    }
private:
    int value;
};

int main()
{
    Counter c1{20};
    Counter c2 = -c1;   // применяем оператор унарного минуса
    c2.print();     // Value: -20
}
```

Операция унарного минуса возвращает новый объект Counter, значение value в котором равно значению value текущего объекта, умноженного на -1.

C++: Перегрузка операций

Операции инкремента ++ и декремента --

Нужно определить и префиксную, и постфиксную форму для этих операторов.

```
3  class Counter
4  {
5  public:
6      Counter(int val)
7      {
8          value =val;
9      }
10     void print()
11     {
12         std::cout << "Value: " << value << std::endl;
13     }
14     // префиксные операторы
15     Counter& operator++ ()
16     {
17         value += 1;
18         return *this;
19     }
20     Counter& operator-- ()
21     {
22         value -= 1;
23         return *this;
24     }
25     // постфиксные операторы
26     Counter operator++ (int)
27     {
28         Counter copy {*this};
29         ++(*this);
30         return copy;
31     }
```

```
32     Counter operator-- (int)
33     {
34         Counter copy {*this};
35         --(*this);
36         return copy;
37     }
38 private:
39     int value;
40 };
41
42 int main()
43 {
44     Counter c1{20};
45     Counter c2 = c1++;
46     c2.print();       // Value: 20
47     c1.print();       // Value: 21
48     --c1;
49     c1.print();       // Value: 20
50 }
```

Постфиксные операторы должны возвращать значение объекта до инкремента, то есть предыдущее состояние объекта.
Поэтому постфиксная форма возвращает копию объекта до инкремента.

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

Чтобы постфиксная форма отличалась от префиксной, постфиксные версии получают дополнительный параметр типа int, который не используется.

C++: Перегрузка операций

Переопределение оператора <<

Оператор << принимает два аргумента: ссылку на объект потока (левый операнд) и фактическое значение для вывода (правый операнд).
Затем он возвращает новую ссылку на поток, которую можно передать при следующем вызове оператора << в цепочке.

```
3  class Counter
4  {
5  public:
6      Counter(int val)
7      {
8          value =val;
9      }
10     int getValue()const {return value;}
11 private:
12     int value;
13 };
14
15 std::ostream& operator<<(std::ostream& stream, const Counter& counter)
16 {
17     stream << "Value: ";
18     stream << counter.getValue();
19     return stream;
20 }
21
22 int main()
23 {
24     Counter counter1{20};
25     Counter counter2{50};
26     std::cout << counter1 << std::endl;     // Value: 20
27     std::cout << counter2 << std::endl;     // Value: 50
28 }
```

Стандартный выходной поток cout имеет тип std::ostream.
Поэтому первый параметр (левый операнд) представляет объект **ostream**,
а второй (правый операнд) - выводимый объект Counter.

Поскольку мы не можем изменить стандартное определение std::ostream, то определяем функцию оператора, которая не является членом класса.

C++: Перегрузка операций

Выражение одних операторов через другие

```
3  class Counter
4  {
5  public:
6      Counter(int n)
7      {
8          value = n;
9      }
10     void print() const
11     {
12         std::cout << "value: " << value << std::endl;
13     }
14     Counter& operator+=(const Counter& counter)
15     {
16         value += counter.value;
17         return *this;
18     };
19     Counter& operator+(const Counter& counter)
20     {
21         Counter copy{ value };      // копируем данные текущего объекта
22         return copy += counter;
23     };
24 private:
25     int value;
26 };
```

оператор сложения с присвоением +=

```
28 int main()
29 {
30     Counter counter1{20};
31     Counter counter2{10};
32
33     counter1 += counter2;
34     counter1.print();   // value: 30
35     Counter counter3 {counter1 + counter2};
36     counter3.print();   // value: 40
37 }
```

В функции оператора сложения создаётся копия текущего объекта и к этой копии и аргументу применяется оператор +=