Перевантаження

З перевантаженням функцій тісно зв'язаний механізм перевантаження операторів. У мові C++ можна перевантажити більшість операторів, настроївши їх на конкретний клас. Наприклад, у класі, що підтримує стек, оператор "+" можна перевантажити для записування елементів у стек, а оператор "-" - для виштовхування елементів з нього. Перевантажений оператор зберігає своє первісне призначення. Просто набір типів, до яких його можна застосовувати, розширюється.

Перевантаження операторів - одна з найефективніших можливостей мови C++. Вона дозволяє повністю інтегрувати нові класи в існуюче програмне середовище. Після перевантаження операції над об'єктами нових класів виглядають точно так само, як операції над змінними вбудованих типів. Крім того, перевантаження операторів лежить в основі системи введення- виведення в мові C++.

Оператори, як функції

Розглянемо наступний фрагмент коду:

```
int a = 5;
	int b = 6;
	std::cout << a + b << '\n';
```

Тут компілятор використовує вбудовану версію оператора плюс (+) для цілочисельних операндів — ця функція додасть два цілочисельних значення (a і b), і поверне цілочисельний результат. Коли ви бачите вираз a + b, то думайте про нього, як про виклик функції `operator+(a, b)` (де operator+ є ім’ям функції).

Тепер розглянемо наступний фрагмент коду:

```
double m = 4.0;
	double p = 5.0;
	std::cout << m + p << '\n';
```

Компілятор також надасть вбудовану версію оператора плюс (+) для операндів типу double. Вираз m + p призведе до виклику функції `operator+(m, p)`, а, завдяки перевантаженню оператора, викличеться версія double (замість версії int).

Тепер розглянемо, що відбудеться, якщо ми спробуємо додати два об’єкти класу:

```
Mystring hello = "Hello, ";
	Mystring world = "World!";
	std::cout << hello + world << '\n';
```

Як ви думаєте, яким буде результат? Напевно, вивід рядка Hello, World!? Ні, результатом буде помилка, тому що клас Mystring є користувацьким типом даних, а компілятор не має вбудованої версії operator+() для використання з операндами Mystring. Для того, щоб зробити те, що ми хочемо, нам доведеться написати свою версію функції operator+() і вказати в ній алгоритм роботи з операндами типу Mystring.

Перевантаження операторів здійснюється за допомогою *операторних функцій* (**operator function**), які визначають дії перевантажених операторів стосовно відповідного класу. Операторні функції створюються за допомогою ключового слова **operator**. Операторні функції можуть бути як членами класу, так і звичайними функціями. Однак звичайні операторні функції, як правило, оголошують дружніми стосовно класу, для якого вони перевантажують оператор. У кожному із цих випадків операторна функція оголошується по-різному. Отже, необхідно розглянути ці варіанти окремо.

Спочатку необхідно відзначити, що не всі символи операцій можна перевантажити. Нижче перераховані
оператори, дозволені до перевантаження.

+	-	*	/	%	^	&	
|	~	!	=	<	>	+	=
=	*=	/=	%=	^=	&=	|=	
<<	>>	>>=	<<=	==	!=	<=	
>=	&&	||	++	--			
->	[]	()	new	new[]	delete	delete[]	

Існують також оператори, заборонені до перевантаження. Зміна їх змісту зруйнувало би логіку програми. До таких операторів належать :: (оператор дозволу області видимості), . (“точка” — оператор доступу до члена класу), ?: (тернарний оператор), .* (доступ до розіменованого вказівника-члена класу), sizeof, typeid, static_cast, dynamic_cast, const_cast і reinterpret_cast. Крім того, не рекомендується перевантажувати логічні оператори && і ||, оскільки на їхні перевантажені версії не поширюється правило скорочених обчислень логічних виразів. (Нагадаємо, що це правило полягає в наступному: якщо на деякому етапі значення усього вираження стає визначеним, подальші обчислення припиняються.)

Створення операторної функції- члена

Операторная функція- член має такий вигляд:

```
тип_повертаємого_значення імя_класу: : operator# ( список-аргументів)
{
      ...   //   Операції
    }
```

Звичайно операторна функція повертає об'єкт класу, з яким вона працює, однак тип значення, що вертається, може бути довільним. Символ # заміняється оператором, що перевантажується. Наприклад, якщо в класі перевантажується оператор ділення "/", операторна функція-член називається **operator/**. При перевантаженні унарного оператора список аргументів залишається порожнім. При перевантаженні бінарного оператора список аргументів містить один параметр.

Є три різних способи перевантаження операторів**:**

через дружні функції;
через звичайні функції;
через методи класу.

Використання дружньої функції для перевантаження оператора зручно тим, що ми маємо прямий доступ до всіх членів класу, з яким працюємо.

Однак, якщо нам не потрібен доступ до членів певного класу, ми можемо перевантажити оператор і через звичайну функцію. Зверніть увагу, в класі Dollars є геттер getDollars(), за допомогою якого ми можемо отримати доступ до m_dollars ззовні класу. Перепишемо перевантаження оператора + через звичайну функцію:

```
#include <iostream>

class Dollars
{
private:
      int m_dollars;

public:
```

```
	Dollars(int dollars) { m_dollars = dollars; }

	int getDollars() const { return m_dollars; }
};

// Примітка: Ця функція не є ані методом класу, ані дружньою класу Dollars!
Dollars operator+(const Dollars& d1, const Dollars& d2)
{
	// Використовуємо конструктор Dollars і operator+(int, int).
	// Тут нам не потрібний прямий доступ до закритих членів класу Dollars
	return Dollars(d1.getDollars() + d2.getDollars());
}

int main()
{
	Dollars dollars1(7);
	Dollars dollars2(9);
	Dollars dollarsSum = dollars1 + dollars2;
	std::cout << "I have " << dollarsSum.getDollars() << " dollars." << std::endl;

	return 0;
}
```

Оскільки принцип перевантаження операторів через звичайні і дружні функції майже ідентичний (вони просто мають різні рівні/умови доступу до закритих членів класу), то єдина відмінність полягає в тому, що у випадку з дружньою функцією, її потрібно обов'язково оголосити в класі, але визначити поза тілом класу (або в класі), в той час як звичайну функцію досить просто визначити поза тілом класу, без вказівки додаткового прототипу функції.

Перевантаження через методи класу.

Розглянемо простий приклад. Ця програма створює клас 1oc, у якому зберігаються географічні координати: широта й довгота. У ній перевантажується оператор "+". Уважно вивчите цю програму, приділяючи особливу увагу визначенню функції operator+ ().

```
#include  <iostream>
using namespace  std;
class loc {
	int longitude, latitude;
public: loc() {}
		loc(int lg, int lt) {
			longitude = lg;
			latitude = lt;
		}
		void show() {
			cout << longitude << " ";
			cout << latitude << "\n";
		}
		 loc operator+(loc op2);
};
// Overload + for loc.
loc loc::operator+(loc op2)
{
	loc temp;
	temp.longitude = longitude + op2.longitude;
	temp.latitude = latitude + op2.latitude;
	return temp;
}

int main() {
	loc ob1(10, 20), ob2(5, 30);
	ob1.show(); // Виводить на екран числа 10 20
	ob2.show(); // Виводить на екран числа 5 30
	ob1 = ob1 + ob2;
	ob1.show(); // Виводить на екран числа 15 50
	system("pause");
```

```
    return 0;
}
```

Як бачимо, функція **operator + ()** має тільки один параметр, незважаючи на те, що вона перевантажує бінарний оператор. (Як відомо, бінарні оператори мають два операнда.) Причина полягає в тому, що операнд, що знаходиться в лівій частині оператора, передається операторній функції неявно за допомогою покажчика **this**. Операнд, що знаходиться у правій частині оператора, передається операторній функції через параметр **op2**. Звідси випливає важливий висновок: при перевантаженні бінарного оператора виклик операторної функції генерується об'єктом, що знаходиться у лівій частині оператора.

Як правило, перевантажені операторні функції повертають об'єкт класу, з яким вони працюють. Отже, перевантажений оператор можна використовувати усередині виразів. Наприклад, якби операторна функція **operator +()** повертала об'єкт іншого типу наступний вираз був би невірним.

```
ob1 = ob1 + ob2;
```

Для того щоб привласнити суму об'єктів *ob1* и *ob2* об'єкту *ob1*, необхідно, щоб результат операції + мав тип **loc**.

Крім того, оскільки операторна функція **operator + ()** повертає об'єкт типу **loc**, допускається наступний вираз:

```
(ob1 + ob2).show();   //   Виводить  на  екран  суму ob1+ob2
```

У цій ситуації операторна функція створює тимчасовий об'єкт, який знищується після повернення з функції **show()**.

Слід розуміти, що операторна функція може повертати об'єкти будь-яких типів, і що ці типи залежать тільки від конкретного додатка. Однак, як правило, операторні функції повертають об'єкти класів, з якими вони працюють.

І ще одне зауваження: операторна функція **operator + ()** не модифікує свої операнди. Оскільки традиційний оператор "+" не змінює свої операнди, не має змісту передбачати для функції **operator + ()** іншої поведінки. (Наприклад, 5+7 рівно 12, але при цьому доданки як і раніше рівні 5 і 7.) Незважаючи на те що зміст операторной функції можна визначати довільно, слід залишатися в рамках здорового глузду.

Наступна програма доповнює клас **loc** трьома перевантаженими операторами: "-", "=" і унарним оператором "++". Зверніть особливу увагу на оголошення цих функцій.

```
#include <iostream>
using namespace std;
class loc {
       int longitude, latitude;
public:
       loc() {}      // Цей конструктор необхідний для створення
                     // тимчасових об'єктів
       loc(int lg, int lt) {
              longitude = lg;
              latitude = lt;
       }
       void show() {
              cout << longitude << " ";
              cout << latitude << "\n";
       }
       loc operator+(loc op2);
       loc operator-(loc op2);
       loc operator=(loc op2);
       loc operator++();
};
// Перевантажений оператор + для класу loc.
loc loc::operator+(loc op2)
```

```
{
	loc temp;
	temp.longitude = op2.longitude + longitude;
	temp.latitude = op2.latitude + latitude;
	return temp;
}
// Перевантажений оператор - для класу loc.
loc loc::operator-(loc op2)
{
	loc temp;
	// Зверніть увагу на порядок операндов
	temp.longitude = longitude - op2.longitude;
	temp.latitude = latitude - op2.latitude;
	return temp;
}

// Перевантажений оператор присвоювання для класу 1oc.
loc loc::operator=(loc op2)
{
	longitude = op2.longitude;
	latitude = op2.latitude;
	return *this; // Повертає об'єкт, що генерує виклик
}
// Перевантажений префиксный оператор инкрементации ++
// для класу loc.
loc loc::operator++() {
	longitude++;
	latitude++;
	return *this;
}

int main() {
	loc ob1(10, 20), ob2(5, 30), ob3(90, 90);
	ob1.show(); ob2.show();
	++ob1;
	ob1.show(); // Виводить на екран числа 11 21
	ob2 = ++obl;
	ob1.show(); // Виводить на екран числа 12 22
	ob2.show(); // Виводить на екран числа 12 22
	ob1 = ob2 = ob3; // Множинне присвоювання
	obl.show(); // Виводить на екран числа 90 90
	ob2.show(); // Виводить на екран числа 90 90
	return 0;
    }
```

Розглянемо, наприклад, операторную функцію **operator- ()**. Зверніть увагу на порядок її операндів. Дія оператора полягає у відніманні значення, що знаходиться в його правій частині, від значення операнда, що знаходиться ліворуч. Оскільки виклик операторної функції **operator-()** генерується об'єктом, що знаходиться ліворуч від знака "мінус", дані об'єкта *ob2* повинні відніматися з даних об'єкта, на який посилається покажчик this. Завжди слід пам'ятати, який об'єкт генерує виклик операторної функції.

Якщо оператор "=" не перевантажений, для класу автоматично створюється оператор присвоювання за замовчуванням, який створює побітові копії об'єктів. Перевантажуючи оператор "=", можна явно визначити оператор присвоювання для конкретного класу. У даному прикладі оператор присвоювання нічим не відрізняється від стандартного, однак в інших ситуаціях він може виконувати інші дії. Зверніть увагу на те, що функція **operator=()** повертає покажчик ***this** на об'єкт, згенерувавший її виклик. Це дозволяє виконувати множинне присвоювання об'єктів, як показано нижче.

ob1 = ob2 = ob3; // Множинне присвоювання

Розглянемо тепер визначення операторної функції **operator++()**. Як бачимо, ця функція не має параметрів. Оскільки оператор інкрементації "++" є унарним, його єдиний операнд неявно

передається йому за допомогою покажчика **this**.

Зверніть увагу на те, що операторні функції **operator=()** і **operator++()** змінюють значення своїх операндів. Наприклад, лівому операнду функції **operator= ()** (генеруючому виклик функції) привласнюється нове значення. Крім того, оператор інкрементації збільшує значення свого операнда на одиницю. Як вказувалося раніше, хоча зміст оператора нічим не регламентується, завжди слід дотримуватися його первісного змісту.

Створення префіксної і постфіксної форм операторів інкрементації й декрементації

У попередній програмі був перевантажений тільки оператор інкрементації в префіксній формі. Однак стандарт мови C++ дозволяє явно створювати окремі версії префіксного й постфіксного операторів інкрементації й декрементаціїи. Для цього слід визначити дві версії функції **operator++()** (і функції **operator- ()** відповідно). Одна із цих версій була визначена в попередній програмі, а іншу ми визначимо нижче.

loc operator++ (int x) ;

Якщо символи ++ передують операндам, викликається операторна функція **operator++ ()**, якщо символи ++ ідуть за операндами, викликається операторна функція **operator++(int x)**.

Попередній приклад можна узагальнити. От як виглядає загальна форма префіксної і постфіксної операторних функцій **operator++()** і **operator--().**

```
//   Префіксний оператор  інкрементації
тип operator++() {
       //   Тіло префіксного оператора
}

//   Постфіксний  оператор  інкрементації
тип  operator++(int   i) {
       //   Тіло постфиксного оператора
}

//   Префіксний  оператор декрементації
тип  operator--() {
       //   Тіло префіксного оператора
}

//   Постфіксний  оператор декрементації
тип  operator--(int   i) {
       //   Тіло постфиксного оператора
    }

loc operator--()
{
       --Re;
       --Im;
       cout >> "Префиксна форма -- \n";
       return *this;
}
loc operator--(int i)
{
       --Re;
       --Im;
       cout << "Постфиксна форма -- n" << i;
       return *this;
    }
```

Якщо символ операції ++ стоїть перед операндом, викликається операторна функція operator++(), якщо після — операторна функція operator++(int i). Змінна і відіграє роль прапора,

що повідомляє компілятору, що дана функція перевантажує постфиксну форму оператора інкремента і декремента.

Перевантаження скорочених операторів присвоювання

Скорочені оператори присвоювання (наприклад, "+=". "-=" і т.п.) також можна перевантажувати. Розглянемо операторну функцію **operator+= ()** для класу **1oc**.

```
loc loc::operator+=(loc op2)
	{
		longitude = op2.longitude + longitude; latitude = op2.latitude + latitude;
		return *this;
	}
```

Обмеження на перевантажені оператори

На застосування перевантажених операторів накладає кілька обмежень. По-перше, не можна змінити пріоритет оператора. По-друге, неможливо змінити кількість операндів оператора. (Однак операнд можна ігнорувати.) По-третє, операторну функцію не можна викликати з аргументами, значення яких задані за замовчуванням (за винятком оператора виклику функції, який ми розглянемо небагато пізніше). І, на закінчення, не можна перевантажувати наступні оператори:

. :: .* ?

Як неодноразово вказувалося, усередині операторної функції можна програмувати будь-які операції. Наприклад, ніхто не забороняє перевантажувати оператор "+" так, щоб він десять раз записував у файл рядок "Я люблю С++". Однак ці дії сильно відрізняються від звичайного змісту оператора "+", і ви ризикуєте зруйнувати свою програму. Якщо хто-небудь стане читати її й побачить оператор *ob1 + ob2*, то, природно, він припустить, що в цім місці складаються два об'єкти, а не виконується операція виведення у файл. Отже, для значного відхилення від звичайного призначення оператора повинні існувати вагомі причини. Одна з них - необхідність перевантажувати оператори "<<" і ">>". Хоча оператори побітового зсуву не мають ніякого відношення до введення- виведення їх зовнішній вигляд настільки добре відповідає цьому призначенню, що неможливо встояти перед спокусою перевантажити їхнім новим змістом. І все-таки , як правило, краще залишатися в рамках традиційного змісту.

За винятком оператора "=", операторні функції успадковуються похідними класами. Однак у похідному класі кожний із цих операторів знову можна перевантажити.

Перевантаження деяких спеціальних операторів

У мові С++ існують оператори індексування елементів масиву "[]", виклику функції "()" і доступу до члена класу "->". Ці оператори також можна перевантажувати, що породжує масу цікавих можливостей.

На перевантаження цих операторів поширюється одне загальне обмеження: вони повинні бути нестатичними функціями-членами. Дружні функції застосовувати не можна.

Перевантаження оператора"[]"

При перевантаженні оператор "**[]**" уважається бінарним. Отже, він повинен мати такий вид:

```
тип імя_класу:: operator [ ] (int  i)
	{
		//…
	}
```

З технічної точки зору параметр не зобов'язаний мати тип **int**, однак функція **operator [] ()** звичайно застосовується для індексування елементів масиву, тому найчастіше використовують саме цілочисленний параметр.

Допустимо, що об'єкт називається **o**. Тоді вираження**0[3]**перетвориться в наступний виклик функції operator[] ():**O.operator[](3)**

Інакше кажучи, значення виразу, вкладеного у квадратні дужки, передається функції **operator [ ] ()** у якості явного параметра. Покажчик **this** посилається на об'єкт **()**, який генерує виклик функції.

У наступній програмі клас **atype** містить масив, що складається із трьох елементів. Його конструктор ініціалізує кожний елемент масиву окремим значенням. Перевантажена операторна функція **operator()[]** повертає значення елемента масиву по заданому індексу.

```
#include   <iostream>
using namespace  std;
class atype {
       int a[3];
public:
       atype(int i, int j, int k) {
              a[0] = i;
              a[1] = j;
              a[2] = k;
       }
       int operator[](int i) { return a[i]; }
};
int main() {
       atype ob(1, 2, 3);
       cout << ob[1]; // Виводить на екран число 2
       return 0;
    }
```

Функцію **operator() []** можна визначити таким чином, щоб оператор "**[]**" був припустимим і в лівій, і в правій частині оператора присвоювання. Для цього потрібно, щоб функція **operator[] ()** повертала посилання. Продемонструємо це за допомогою наступної програми.

```
#include <iostream>
using namespace std;
class atype {
       int a[3];
public:
       atype(int i, int j, int k) { a[0] = i; a[1] = j; a[2] = k; }
       int &operator [ ] (int i) { return a[i]; }
};
int main() {
       atype ob(1, 2, 3);

       cout << ob[1]; // Виводить на екран число 2
       cout << " ";
       ob[1] = 25; // Оператор [] розташований ліворуч
       cout << ob[1]; // Тепер на екран виводиться число 25
       return 0;
    }
```

Оскільки тепер операторна функція **operator [] ()** повертає посилання на елемент масиву з індексом i, її можна використовувати в лівій частині оператора присвоювання для модифікації елементів масиву. (Зрозуміло, оператор "**[]**" як і раніше можна використовувати в правій частині оператора присвоювання.)

Перевантаження оператора "**[]**" дозволяє забезпечити контроль над виходом індексу масиву за межі припустимого діапазону, який за замовчуванням не передбачений у мові C++. Створивши клас, що містить масив, і передбачивши перевантажений оператор "**[]**", можна перехопити виникаючу виняткову ситуацію. Розглянемо приклад, у якому програма передбачає перевірку діапазону індексів масиву.

// Приклад безпечного масиву.

```
#include <iostream>

using namespace std;
class atype {
       int a[3];
public:
       atype(int i, int j, int k) { a[0] = i; a[1] = j; a[2] = k; }
       int &operator[](int i);
};
// Перевірка діапазону зміни індексу для класу atype.
int &atype::operator[](int i)
{
       if (i<0 || i> 2)
       {
              cout << "Вихід за межі допустимого діапазону\n"; exit(1);
       }
       return a[i];
}
int main() {
       atype ob(1, 2, 3);
       cout << ob[1]; // Виводить на екран число 2
       cout << " ";
       ob[1] = 25; // Оператор [] ліворуч
       cout << ob[1]; // Виводить на екран число 25
       ob[3] = 44;   // Генерується помилка, значення 3
                                  // лежить поза припустимим діапазоном
       return 0;
    }
```

При виконанні оператора

ob[3] = 44;

функція **operator[] ()** перехоплює виниклу виняткову ситуацію й припиняє виконання програми. (На практиці виконання програми звичайно не припиняється, а замість цього викликається оброблювач виняткової ситуації, пов'язаної з виходом індексу за межі припустимого діапазону.)

Перевантаження оператора"()"

При перевантаженні оператора "()" створюється операторна функція, якої можна передавати довільну кількість параметрів. При цьому новий спосіб виклику функції па самій справі не створюється. Розглянемо приклад. Допустимо, що оператор виклику функції перевантажений у деякому класі в такий спосіб:

```
double  operator()(int  a, float   f, char   *s);
```

Крім того, припустимо, що змінна **o** є об'єктом цього класу. Тоді оператор

```
0(10, 23.34, "Привіт");
```

перетвориться в наступний виклик операторної функції **operator()**:

```
O.operator()(10, 23.34, "Привіт");
```

Як правило, при перевантаженні оператора " **()**" визначаються параметри, передані функції. При виклику функції ці параметри копіюються й привласнюються відповідним аргументам. Як завжди, покажчик **this** посилається на об'єкт, що генерує виклик операторної функції (у даному прикладі на об'єкт **o**)

Розглянемо приклад перевантаженого оператора "**()**" у класі **1oc**. Він привласнює значення двох своїх аргументів членам об'єкта класу **loc**, у яких зберігається значення широти й довготи.

```
#include <iostream>
using namespace std;
class loc {
	int longitude, latitude;
public:
	loc() {}
	loc(int lg, int lt) { longitude = lg; latitude = lt; }
	void show() {
		cout << longitude << " ";
		cout << latitude << "\n";
	}
	loc operator+(loc op2);
	loc operator()(int i, int j);
};
// Перевантажений оператор ( ) для класу loc.
loc loc::operator()(int i, int j)
{
	longitude = i;
	latitude = j;
	return *this;
}
// Перевантажений оператор + для класу loc.
loc loc::operator+(loc op2)
{
	loc temp;
	temp.longitude = op2.longitude + longitude; temp.latitude = op2.latitude + latitude;
	return temp;
}
int main() {
	loc ob1(10, 20), ob2(1, 1);
	ob1.show();
	ob1(7, 8); // Оператор застосовується самостійно
	ob1.show();
	ob1 = ob2 + ob1(10, 10);    // Оператор можна використовувати
                                                    // усередині виразів
	ob1.show();
	return 0;
    }
```

Ця програма виводить на екран такі рядки.

10 20

7 8

11 11

Слід пам’ятати, що, перевантажуючи оператор " **()**", ви можете використовувати будь-який тип параметрів і будь-який тип значення, що вертається. Ці типи можна вибирати залежно від конкретного додатка. Крім того, можна передбачати значення аргументів за замовчуванням.

Перевантаження оператора "->"

Оператор посилання на член об'єкта (class member access operator) при перевантаженні вважається унарним. Його загальний вид такий.

об'єкт -> елемент

Операторна функція, що перевантажує оператор "->", викликається з об'єкта. Функція **operator->()** повинна повертати покажчик на об'єкт класу, для якого він визначений. Елемент повинен бути членом, доступним усередині об'єкта.

Наступна програма ілюструє перевантаження оператора "->" і демонструє еквівалентність виразів**ob.i** й **ob->i**, коли операторна функція **operator-> ()** повертає покажчик this.

```
#include <iostream>
using namespace std;
```

```
class myclass {
public:
	int i;
	myclass *operator->() { return this; }
};
int main() {
	myclass ob;
	ob->i = 10; // Еквівалентно виразу ob.i
	cout << ob.i << " " << ob->i;
	return 0;
    }
```

Операторная функція **operator->()** повинна бути членом класу, для якого вона визначається.

Перевантаження оператора ","

Мова С++ допускає перевантаження оператора ",". Цей оператор є бінарним. Його зміст при перевантаженні може бути довільним. Однак, якщо метою перевантаження є виконання операції, аналогічної стандартної, слід ігнорувати всі аргументи, крім крайнього праворуч. Саме цей параметр уважається результатом стандартного оператора ",".

Розглянемо програму, що демонструє перевантаження оператора ",".

```
#include  <iostream>
using namespace  std;
class loc {
	int longitude, latitude;
public: loc() {}
		loc(int lg, int lt) { longitude = lg; latitude = lt; }
		void show() {
			cout << longitude << " ";
			cout << latitude << "\n";
		}
		loc operator+(loc op2);
		loc operator, (loc op2);
};
// Перевантаження оператора "кома" для класу loc
loc loc::operator,(loc op2)
{
	loc temp;
	temp.longitude = op2.longitude;
	temp.latitude = op2.latitude;
	cout << op2.longitude << " " << op2.latitude << "\n";
	return temp;
}
// Перевантаження оператора + для класу loc
loc loc::operator+(loc op2)
{
	loc temp;
	temp.longitude = op2.longitude + longitude; temp.latitude = op2.latitude + latitude;
	return temp;
}
int main() {
	loc ob1(10, 20), ob2(5, 30), ob3(1, 1);
	ob1.show(); ob2.show(); ob3.show(); cout << "\n";
	ob1 = (ob1, ob2 + ob2, ob3);
	ob1.show();   // Виводить на екран числа 1 1,
				// тобто значення об'єкта ob3
	return 0;
    }
```

Ця програма виводить на екран наступні рядки.

10 20
5 30
1 1

```
10   60
1   1
1   1
```

Незважаючи на те що всі операнди, що розташовані у лівій частині, ігноруються, кожний вираз як і раніше обчислюється компілятором, тому можливі всі передбачені побічні ефекти.

Слід пам'ятати, що операнд, що знаходиться в лівій частині, передається за допомогою покажчика this, а його значення ігнорується функцією **operator, ()**. Функція повертає значення операнда, що знаходиться в правій частині. Таким чином, дії перевантаженого оператора "," нагадують стандартні. Якщо ви прагнете, щоб оператор виконував інші операції, слід змінити ці властивості.