Перевантаження операцій в мові C++

При використанні змінних будь-якого стандартного типу, для них припустимий певний набір операцій. Мова C++ надає можливість створювати класи так, щоб вони також реалізовували необхідний набір операцій. Зокрема можливо перевизначити (перевантажити) звичні знаки операцій таким чином, щоб вони стосувались екземплярів класів. Наприклад, нехай визначений клас **`complex`**, що реалізує комплексне число. Звісно, можна реалізувати дії з екземплярами цього класу як звертання до певних функцій класу, проте значно зручніше було б оперувати звичними для сприйняття виразами на кшталт:

```
complex a, b;
complex c = a + b; c = a*b;
```

Вивченню подібних можливостей і присвячений подальший виклад.

Дружні функції.

Як відомо, доступ до закритих (і захищених) членів класу неможливий із зовнішнього коду. Проте виняток становлять так звані ***“дружні”*** функції. Достатньо помістити в класі декларацію зовнішньої функції із службовим словом **friend**, і вона матиме доступ до всіх без винятку членів класу.

```
class Student {
private :
    char name [20]; // ім’я студента
    double av_mark; // середній бал
    double ex_mark; // бал за екзамен
   // дружня функція
    friend void excellent (Student &s);
public :
   …
};
// Ця функція робить відмінником будь-якого студента
void excellent (Student &s)
{
   s.av_mark = 60;
   s.ex_mark = 40;
}
```

Зауваження про дружні функції.

1. Слід пам'ятати, що функція із модифікатором **`friend`** не є членом класу, хоча її декларація і фігурує в класі.
2. Дружня функція має такі самі права доступу до всіх членів класу як і функція-член класу.

Вказівник this.

Кожна функція-член класу (в тому числі конструктори та деструктор) мають вказівник **this**. Його особливість в тому, що він вказує на екземпляр, для якого здійснюється виклик даної функції.

Приклад (продовження).

```
class Student
{   …
};
int main ()
{
  Student st1 (30, 30);
  Student st2 (60, 40, "Ivanov");
  cout << "Student " << st1.get_mark () << endl;
  cout << "Student " << st2.get_exam () << endl;
  return 0;
}
```

В даному прикладі функція **get_mark ()** класу одержує вказівник **this**, який вказує на екземпляр **st1**, а функція **get_exam () -** вказівник **this**, який вказує на екземпляр **st2**.

Правила перевантаження операцій в класі

Для перевантаження операцій використовуються так звані ***операторні функції*** – функції із ідентифікатором operator@, де замість символу @ стоїть знак операції, що перевантажується. При визначенні операторної функції дотримуються спеціальних синтаксичних вимог.

По-перше, операторна функція може бути визначена як ***функція-член класу*** або як ***зовнішня дружня функція***.

По-друге, по-різному визначаються унарні та бінарні операції.

Декларація функції-члену класу для перевантаження бінарної операції:

```
<тип_результату> operator@(< тип_операнду_2>);
```

При цьому перший операнд операції передається неявно у вигляді вказівника **this**, тобто обов'язково має тип классу. Таким чином, для реалізації операції виду **a@b**, де **a** та **b** – екземпляри відповідного класу, відбувається виклик операторної функції:

```
a.operator@ (b)
```

Декларація дружньої функції для перевантаження бінарної операції:

```
friend <тип_результату> operator@
   (<тип_операнду_1>, < тип_операнду_2>);
```

При цьому для реалізації операції виду **a@b** відбувається виклик операторної функції:

```
operator@ (a,b)
```

Декларація функції-члену класу для перевантаження унарної операції:

```
<тип_результату> operator@ ();
```

При цьому операнд операції передається неявно у вигляді вказівника **this**, тобто обов'язково має тип классу. Для реалізації операції виду **@a** відбувається виклик операторної функції:

```
a.operator@ ()
```

Декларація дружньої функції для перевантаження унарної операції:

```
friend <тип_результату> operator@
(<тип_операнду>);
```

В цьому випадку для реалізації операції виду **@a** відбувається виклик операторної функції:

```
operator@ (a)
```

Особливості використання операції присвоєння.

1. Слід знати, що за умовчанням операція = для двох екземплярів класу здійснює поелементне копіювання даних-членів цих екземплярів. Якщо це саме те, що потрібно для вашого класу, немає необхідності у перевантаженні операції = .
2. Якщо членом класу є вказівник, то за умовчанням відбуватиметься копіювання відповідних вказівників, а не об'єктів, на які вони посилаються (чого, скоріше за все, ви очікуєте при присвоєнні). В такому разі необхідно коректно перевантажити операцію = .
3. Використання параметром такої операторної функції посилання на об'єкт позбавить від створення та знищення у стеку його копії і зекономить ресурси.

Ще один нюанс визначення унарних операцій пов'язаний із операціямии інкременту та декременту. Як відомо, і інкремент, і декремент можуть префіксними або постфіксними (в мові C# вони не розрізнялись). Мова C++ надає можливість реалізувати префіксні та постфіксні операії по-різному. Для цього в операторній функції, що реалізує постфіксні операції, використовується фіктивний параметр. Його наявність дозволяє перевантажити відповідним чином операції:

```
// префіксний інкремент:
<тип_результату> operator++ ();
// постфіксний інкремент:
<тип_результату> operator++ (int unused);
```

Або, якщо операції визначаються з допомогою дружніх функцій:

```
// префіксний інкремент:
friend <тип_результату> operator++ (<тип_операнду>);
// постфіксний інкремент:
friend <тип_результату> operator++
                    (<тип_операнду>, int unused);
```

Абсолютно аналогічні правила діють і для операції декременту.

Зауваження про обмеження при перевантаженні операцій.

1. Перевантажена операція класу повинна мати принаймні один операнд з типом даного класу – таким чином забезпечується цілісність операцій зі стандартними типами даних: ви, наприклад, не можете перевантажити операцію `+` для цілих змінних так, щоб вона реалізовувала, наприклад, віднімання.
2. Перевантажена операція не може змінювати синтаксис існуючих операцій, наприклад операція `%` не може бути унарною. Так само неможливо змінити пріоритети існуючих операцій.
3. Неможливо створювати нові символи для операцій, наприклад, **`x**y`** неможливо реалізувати як піднесення **`x`** до степеня **`y`**.
4. Не перевантажуються наступні операції:

`sizeof` `.` (доступ до елементу) `*` (операція вказівник на елемент) `::` (оператор області видимості) `?:` (тернарна операція) та деякі інші, пов'язані з приведенням та перетворенням типів.

Зауваження про використання дружніх функцій при перевантаженні операцій.

1. В багатьох випадках не має різниці, яку саме форму операторної функції (функція-член класу чи дружня функція) ви використовуєте.
2. Якщо ж ви намагаєтесь перевантажити деяку операцію, яка буде використовуватись у виразах, де перший операнд не є екземпляром даного класу (скажімо, при множенні скаляра на екземпляр класу, що реалізує вектор або матрицю), то операторна функція обов'язково має бути реалізована як дружня.
3. Більшість операцій можуть бути перевантажені як функції-члени класу, так і з допомогою дружніх функцій. Проте наступні операції можуть перевантажуватись лише з допомогою функцій-членів класу: `=` (присвоєння) `()` (оператор виклику функції) `[]` (оператор індексації)

`->` (доступ до членів класу через вказівник).