ТЕМА: Перевантаження операторів. Операторні функції.

Зміст перевантаження операторів

Якщо перевантаження функцій – це різні способи виконання однакової функції, то перевантаження операторів – це різні способи виконання стандартних операцій, а точніше – налаштування стандартних операцій до конкретного класу.

Перевантаження операторів реалізують за допомогою операторних функцій. Терміни «перевантаження операторів» і «операторні функції» є синонімами.

Припустимо, що маємо деякі два об'єкти

```
Stat T1 = Stat(5,10);
Stat T2 = Stat(6,8);
```

Було б непогано мати можливість записувати і виконувати операції так:

```
Stat T3 = T1 + T2;   // бінарна операція - сума двох об'єктів
```

тобто виконати деяку операцію з об'єктами і отримати новий об'єкт. Оскільки клас `Stat` визначений програмно, то компілятор не знає, як виконати таку операцію. Мова C++ дозволяє побудувати визначення подібних операцій, яке називають перевантаженням. Реалізувати перевантаження можна за допомогою операторної функції, яка має загальний вигляд для випадку бінарних операцій:

```
тип_результату  ім'я_класу :: operator# ( другий_операнд )
{
  //  . . . . . реалізація виконання операції
}
```

Замість символа # записують перевантажену операцію, наприклад **operator+**, **operator/**, **operator=**. Якщо операторна функція визначена, тоді запис **T1+T2** компілятор автоматично перетворює до форми **T1.operator+(T2)**, тобто фактично викликається як звичайний метод. При цьому перший операнд є викликаючим об'єктом, а другий – параметром функції. Якщо ж операторна функція додавання не визначена, компілятор повідомляє про помилку – «не можна виконати такої операції».

З точки зору самої операторної функції для **T1+T2** маємо таке співвідношення:

```
Stat  Stat :: operator+ ( const Stat & ob )
 {  // T1 - this,  T2 - ob
    //  . . . . . реалізація виконання операції
 }
```

Останнє важливе питання щодо змісту перевантаженого оператора – вирішити, що має бути результатом операції. Як правило, операторні функції повертають об'єкт такого самого класу, з яким вони працюють, хоча в принципі – це не обов'язково. Повернений об'єкт може бути новим створеним в операторній функції, а може бути одним з двох, що приймають участь в операції.

Зауваження: зазвичай операторні функції не модифікують свої операнди, як це прийнято традиційно для математичних операцій. Модифікація можлива у випадках, якщо сама операція це передбачає, наприклад, **operator++**.

Перевантажувати можна майже всі операції за деяким винятком.

Приклад перевантаження операції додавання

Щоб можна було додавати об'єкти операцією +, потрібно її перевантажити. Проте спочатку треба вирішити, що ми розуміємо під додаванням об'єктів класу Stat. Визначимо, наприклад, так: це має бути об'єкт меншого з двох розмірів за кількістю елементів, кожен елемент є сумою відповідних пар елементів об'єктів-операндів:

```
Stat operator+ (const Stat & ob) // перевантаження операції +
{ // до власного об'єкта додати об'єкт ob
  Stat temp;  // це буде результат додавання
  temp.size = this->size < ob.size ? this->size : ob.size;
              // менший з двох за розміром
  temp.mas = new int[temp.size]; // місце для нового масиву
  for(int i=0; i<temp.size; i++)
     temp.mas[i]=this->mas[i]+ob.mas[i];
  temp.Calc();
  return temp;  // повернути копію об'єкта temp
}
```

Виконаємо тестування перевантаженої операції + наприклад у функції main():

```
{  // деякий блок
  Stat T1 = Stat(5,10);   Stat T2 = Stat(7,9);
  T1.Print();  T2.Print();
  Stat T3 = T1 + T2;   // бінарна операція - сума двох об'єктів
  T3.Print();
}   // <- тут виконаються деструктори для T3, T2, T1
```

Звернемо увагу, що група операторів записана в окремих фігурних дужках, які означають окремий блок. В цьому разі при виході з блоку можна спостерігати дію деструктора – він має друкувати повідомлення `Object destroyed.`:

```
Current values:
10  10  10  10  10
ave=10  max=10
Current values:
9  9  9  9  9  9  9
ave=9  max=9
Object destroyed.    <- знищення об'єкта temp
Current values:
19  19  19  19  19
ave=19  max=19
Object destroyed.
Object destroyed.
Object destroyed.
```

Аналіз визначення і результатів.

1) Операторна функція `operator+` повертає оператором `return` копію об'єкта `temp`. При цьому неявно виконується конструктор копіювання, визначений нами раніше. За оператором `return` закінчується блок функції, тому сам об'єкт `temp` перестає існувати, про що ми бачимо повідомлення.

2) Оскільки об'єкт `temp` перестає існувати при виході з функції, його не можна повертати як `Stat*` чи `Stat&`. Проте, якщо об'єкт створити динамічно в операторній функції

```
Stat * temp = new Stat(. . .);
. . . . . .
return temp;
```

тоді результат функції може бути `Stat *`, як це було раніше у методі `CreateFromEven()`. В цьому разі повертається копія вказівника temp, але не копія об'єкта.

Ускладнимо процедуру тестування:

```
{
  Stat T1 = Stat(4,10);
  Stat T2 = Stat(5,8);
  Stat T3 = Stat(7,15);
  T1.Print();  T2.Print();  T3.Print();
  Stat T4 = T1 + T2 + T3; // сума трьох об'єктів
  T4.Print();
}
Current values:
33  33  33  33
ave=33  max=33
```

Аналіз результатів показує, що операції додавання можна виконати багатократно одним оператором.

Повторне перевантаження операцій

Нехай, наприклад, є потреба виконувати додавання ще й так:

```
Stat T5 = T1 + 32;   // сума об'єкта та іншого операнда
```

Тепер другий операнд є цілим числом. Треба ще раз перевантажити операцію додавання, але тепер параметром операторної функції буде ціле число. Зміст операції, наприклад такий – кожен елемент масиву збільшити на величину операнда:

```
Stat operator+ (int x)   // ще одне перевантаження операції +
{ // до власного об'єкта додати число x
  Stat temp; // це буде результат додавання
  temp.size = this->size ; // розмір масиву не змінився
  temp.mas = new int[temp.size]; // місце для нового масиву
  for(int i=0; i<temp.size; i++)
      temp.mas[i]=this->mas[i]+x;
  temp.Calc(); return temp; // повернути копію об'єкта temp
}
```

Загальний висновок. Перевантажувати ту саму операцію можна багатократно:

```
 T5 = T1 + (операнд довільного типу);
```

Для кожного іншого типу другого операнда треба будувати операторну функцію:

```
Stat operator+ (тип другого операнда) // повторне перевантаження
{ // до власного об'єкта додати операнд іншого типу
  . . . . . .
  return (об'єкт класу Stat);
}
```

Перевантаження комбінованих операторів присвоєння

Комбіновані оператори присвоєння так само можна перевантажувати. Наприклад, потрібно забезпечити виконання операцій T5 += T1. За змістом така операція співпадає з операцією додавання, лише результатом має бути не новий об'єкт `temp`, а вже існуючий об'єкт першого операнда:

```
Stat operator+= (const Stat & ob) // перевантаження операції +=
{ // власний об'єкт збільшити значеннями об'єкта ob
  int less = size < ob.size ? size : ob.size; // обчислити
менший розмір
  for(int i=0; i<less; i++) mas[i] += ob.mas[i];
  Calc();
  return *this;  // повернути копію власного об'єкта
}
```

Звернемо увагу, що операторна функція повертає як результат перший операнд – для можливості будови складніших виразів, наприклад:

```
T1 += T2 + T5;    // означає T1 = T1 + ( T2 + T5);
Stat T6 = ( T1+=T2 ) + T5;
// приорітет + вищий, ніж комбінованих присвоєнь - потрібні дужки
```

В цьому разі виконуються в одному операторі дві перевантажені операції: додавання окремо і комбіноване присвоєння з додаванням. В другому випадку потрібна копія результату операції +=.

Перевантаження операції присвоєння

Раніше записаний оператор

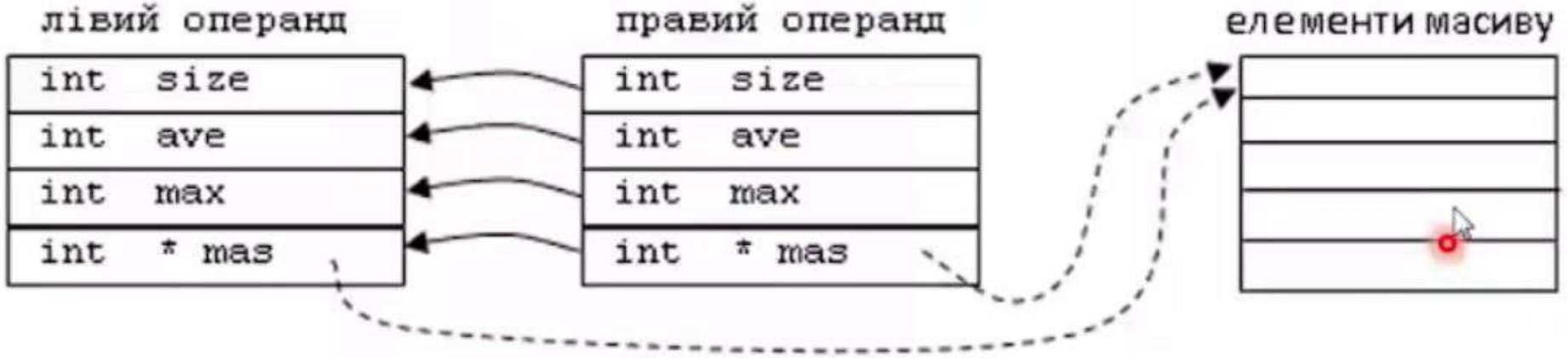
```
Stat T3 = T1 + T2;
```

може бути розведений за текстом

```
Stat T3;
. . . . . .
T3 = T1 + T2;
```

В цьому разі крім операції додавання об'єктів виконується також операція присвоєння об'єктів. Також завжди можна виконати операцію простого присвоєння об'єктів подібно до `T2=T1`. Звернемо увагу, що мова йде про «чисте» присвоєння, а не про комбіновані оператори присвоєння, розглянуті вище.

За замовчуванням оператор присвоєння перевантажується автоматично і виконує побітове копіювання всіх полів. Для випадку статичних полів це дає потрібний результат. Проте, якщо об'єкт має динамічні поля, то вони скопійовані не будуть. Виникає така ж сама ситуація, як у випадку конструктора копіювання:



Отже, потрібно перевантажити оператор присвоєння:

```
Stat & operator= (const Stat & ob) // перевантаження операції =
{
  if(this == &ob) // на випадок присвоєння самому собі
    return *this;
  delete [] mas; // викреслити попередній масив
  size = ob.size;
  mas = new int[size]; // створити масив за новим розміром
  for(int i=0; i<size; i++) mas[i]=ob.mas[i];
        // копіюємо нові значення
  Calc();  // заново переобчислити
  return *this;   //  ! на випадок множинного присвоєння
}
```

Звернемо увагу, що операторна функція повертає `*this` копію об'єкта, який згенерував її виклик. Це дозволяє виконати множинні присвоєння об'єктів:

```
T3 = T2 = T1;
```

Перевантаження операцій логічного типу результату

Шляхом перевантаження можна забезпечити можливість порівняння об'єктів чи виконання логічних операцій над ними, наприклад:

```
if ( об'єкт1 < об'єкт2 )  { блок; }  else  { блок; }
```

В цьому разі результатом операції має бути значення, яке можна привести до логічних `true`, `false`. Так само, як раніше, визначимо зміст операції <, наприклад, порівняти середні значення:

```
bool operator< (const Stat & ob)  // перевантаження операції <
{
  if (this -> ave < ob.ave)  return true;
  else  return false;
}
```

Тепер можна скористатись цією операцією:

```
if (T1 < T2) . . . . . else . . . . . .
```