

## ТЕМА: Перевантаження спеціальних операторів.



### Префіксна і постфіксна форма операторів інкременту і декременту

Операції інкременту ++ і декременту -- є унарними операторами. Ці операції модифікують свій операнд. Наприклад, ++V, V--. Незалежно від того, маємо префіксну чи постфіксну форми (++V, V++), потрібно, як звичайно, спершу визначити зміст операції для конкретного класу. Для випадку нашого класу Stat визначимо префіксну форму ++V так: збільшити на 1 всі елементи масиву об'єкта. Оскільки це є унарна операція, то операторна функція не буде мати параметрів:

```
Stat & operator++ () // перевантаження префіксної форми ++
{
    for(int i=0; i<size; i++) mas[i]++;
    Calc();
    return *this; // для виконання подальших операцій
}
```




## Перевірка виконання перевантаженої операції:

```
{  
    Stat B = Stat(6,10);  
    B.Print();  
    ++B; // унарна операція – префіксна форма  
    B.Print();  
}
```

```
Current values:  
10  10  10  10  10  10  
ave=10  max=10  
Current values:  
11  11  11  11  11  11  
ave=11  max=11  
Object destroyed.
```

Операторна функція безпосередньо модифікує власний об'єкт. Після цього повертає його як результат – для можливого продовження обчислень, наприклад:

```
Stat B = Stat(6,10);  
Stat C = Stat(4,25);  
B.Print();  C.Print();  
Stat D = ++B + C; // виконання декількох операцій  
B.Print();  
D.Print();
```

 Постфіксна форма. Постфіксну форму перевантаженої операції для випадків типу `B++` можна не визначати, оскільки з сторони задачі це може означати те саме, що й префіксна форма. Якщо не визначати, то вступає в силу правило: компілятор для постфіксної форми підставляє префіксну. Якщо ж все таки є потреба окремо визначити постфіксну форму перевантаженої операції, то формально операторна функція повинна мати один параметр типу `int`:

```
Stat & operator++ (int)  // перевантаження постфіксної форми ++
{
    for(int i=0; i<size; i++)  mas[i]++;
    Calc();
    return *this; // для виконання подальших операцій
}
```

Оскільки параметр операторної функції не використовують, достатньо зазначити лише його присутність типом `int`.

В нашому випадку постфіксна форма нічим не відрізняється від префіксної, проте в інших випадках є можливість визначити дві форми по-різному.



## Перевантаження оператора індексування “[ ]”

Квадратні дужки використовують для індексування елементів масиву. Проте їх можна використати як операцію над об'єктом. Якщо об'єкт називається *М*, тоді можна вживати запис *М[3]* або подібний. Знову ж таки, як раніше, треба надати зміст оператору індексування для випадку об'єкта. Переважно індексування об'єкта означає доступ до відповідного елемента масиву, якщо він присутній в структурі класу. Тобто можна напряду звернутись до окремого елемента масиву.

Форма перевантаженої операції індексування має вигляд:

```
тип_результату ім'я_класу :: operator[] ( int i )  
{  
    // . . . . . повернення елемента масиву  
}
```

Отже, запис *М[3]* буде перетворений в операторну форму *М.operator[] (3)*. Параметром операторної функції буде індекс елемента масиву.

Для коректної реалізації операторна функція має забезпечити контроль за виходом індекса масиву за межі допустимого діапазону. Оскільки операторна функція має повернути якийсь результат у всіх випадках, тому можна допустити, що у випадку некоректного індексу повертаємо число нуль.

```
int operator[] (int i)  // перевантаження оператора індексування
{
    if(i>=0 && i<size) return mas[i]; // контроль індекса
    else return 0;
}
```

Тестування:

```
{
    Stat M = Stat(5,20);
    M.Print();
    int x = M[0]*M[1]-M[14];
    cout << x << endl;
}
```

Буде надруковано значення x рівне 400.



## Перевантаження оператора круглих дужок “( )”

Якщо об'єкт називається *F*, тоді можна передати об'єкту через операторну функцію довільну кількість параметрів *F(par1, par2, ...)*. Зміст такої операторної функції ми визначаємо окремо, як і для інших видів операторних функцій. При цьому функція може повертати результат довільного типу або не повертати результату *void*. Перевантаження круглих дужок називають функціональним застосуванням об'єкта.

Оператор круглих дужок можна перевантажити повторно, використавши іншу сигнатуру.

### Приклади перевантаження круглих дужок

1. Кількість параметрів більша одного. Побудувати перевантаження для задачі: всі елементи масиву відрізка *[a,b]* замінити нулями; обчислити кількість таких елементів:

```
int operator() (int a, int b)
// перевантаження круглих дужок з двома параметрами
{ // всі елементи відрізка [a,b] замінити нулями;
  // обчислити кількість таких елементів
  int count = 0; // лічильник
  for(int i=0; i<size; i++)
    if(mas[i]>=a && mas[i]<=b) { mas[i]=0; count++; }
  Calc();
  return count;
}
```

## Перевірка реалізації:

```
{  
    Stat R = Stat(6,30);  
    R.Repl(2,-4);    R.Repl(3,9);  
    R.Print();  
    int k = R(-1, +10); // перевантаж. круглих дужок з двома парам.  
    R.Print();  
    cout << k << endl;  
}
```

## Результати:

```
Current values:  
30  30  -4  9  30  30  
ave=20  max=30  
Current values:  
30  30  0  0  30  30  
ave=20  max=30  
2  
Object destroyed.
```

2. Перевантаження з одним параметром. В цьому випадку параметр можна використати як код функції і зробити об'єкт подібним до універсальної функції. Параметр може приймати, наприклад, одне значення з допустимого переліку. Сам перелік можна визначити у файлі `stat.h` класу, наприклад, так (до початку визначення класу):

```
enum funclist { firstnegative, firstpositive, avevalue, turnover };
```

Перевантажений оператор може бути таким:

```
int operator() (funclist code)
// перевантаження круглих дужок з одним параметром
{ // параметр означає код потрібної функції
  int result;  int temp;
  switch (code)
  {
  case firstnegative: // знайти перше за порядком від'ємне у мас.
    result=0; // якщо немає - відповідь нуль
    for(int i=0; i<size; i++)
      if(mas[i]<0) { result=mas[i]; break; } // вихід з циклу
    break; // вихід з case
```



```
case firstpositive: // знайти перше за порядком додатне у масиві
    result=0; // якщо немає - відповідь нуль
    for(int i=0; i<size; i++)
        if(mas[i]>0) { result=mas[i]; break; }
    break;
```

```
case avevalue:
    // повернути середнє значення - можна дублювати метод класу
    result=ave;
    break;
```

```
case turnover:
    // переставити елементи масиву в оберненому порядку
    for(int i=0; i<size/2; i++)
        { temp=mas[i]; mas[i]=mas[size-1-i]; mas[size-1-i]=temp; }
    result = size;
    // повернути кількість переставлених елементів - наприклад
    break;
```

```
} // switch
```

```
return result;
```

```
}
```

### Тестування реалізації:

```
{  
    Stat W = Stat(7,25);  
    W.Repl(2,-15);  W.Repl(4,-8);  W.Repl(5,-30);  W.Repl(0,99);  
    W.Print();  
    int anyvalue = W(firstnegative);  cout << anyvalue << endl;  
    cout << W(firstpositive) << '\t' << W(avevalue) << endl;  
    W(turnover);  
    W.Print();  
}
```

### Отримаємо такі результати:

```
Current values:  
99  25  -15  25  -8  -30  25  
ave=17  max=99  
-15  
99      17  
Current values:  
25  -30  -8  25  -15  25  99  
ave=17  max=99  
Object destroyed.
```

## Дружні функції



Припустимо, що приклад додавання

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

розглянутий на попередній лекції, потрібно записувати ще й так:

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

В цьому випадку першим операндом операції додавання не є об'єкт, значить неможливо викликати якийсь метод чи операторну функцію за розглянутими вище правилами. У зв'язку з цим введено поняття «дружня функція» - friend.

Означення. Дружня функція – це такий метод, який визначений в класі, має доступ до всіх елементів класу, але не є членом класу. Дружня функція не отримує неявного вказівника `this`, відповідно, отримує всі параметри явно: два параметри для бінарного оператора і один – для унарного.



## Перевантаження операції додавання.

Запишемо в класі прототип функції, а реалізацію – поза класом. Прототип:

```
friend Stat operator+ (int x, Stat & ob);  
// прототип дружньої функції
```

Реалізація:

```
Stat operator+ (int x, Stat & ob) // реалізація дружньої функції  
{ // сума числа і об'єкта  
    return ob + x; // виклик методу Stat operator+ (int x)  
}
```

За таким способом реалізації слово `friend` повторно не записують, так само не записують посилання на клас `Stat::`

Перевірка:

```
{  
    Stat S1 = Stat(5,14);  
    Stat S2 = 8 + S1;  
    S2.Print();  
}
```

## Перевантаження оператора друкування.

Нехай R3, R4 – деякі об'єкти. Для багатьох задач буває потреба зручного запису операції друкування об'єкта, наприклад:

```
cout << R3 << R4;  
// друкування об'єктів – аналог R3.Print(); R4.Print(); тощо
```

Оператором тут є <<, а операндами – cout, R3, R4. Щоб реалізувати таку операцію, потрібно визначити дружню функцію для оператора << друкування в потік:

```
friend ostream & operator<< (ostream & os, const Stat & ob)  
// друкування об'єкта  
{ // визначення дружньої функції в класі  
  os << "Current values:\n";  
  for(int i=0; i<ob.size; i++) os << ob.mas[i] << "  ";  
  os << endl;  
  os << "ave=" << ob.ave << "  max=" << ob.max << endl;  
  return os; // ! повернути об'єкт друкування  
             //      - для випадку продовження  
}
```

### Зауваження щодо цієї функції.

1) Функція має повертати як результат об'єкт друкування. Тоді можна одним зовнішнім оператором друкувати зразу декілька об'єктів.

Для наведеного вище оператора `cout<<R3<<R4;` спочатку для частини `cout<<R3` буде викликана функція `operator<<(cout, R3)`. Виконавши операції, функція повертає `cout`. Тому наступним кроком буде `cout<<R4` з аналогічним викликом функції.

2) Цю ж функцію можна використати для друкування в текстовий файл. Лише такий файл потрібно відкрити до операції друкування, а закрити – після друкування.

### Загальні зауваження щодо дружніх функцій.

1) Дружня функція не є членом класу. Всі параметри бінарних і унарних перевантажених операторів отримує явно. Має доступ до всіх елементів класу, - але лише через свої параметри, оскільки не має вказівника `this`.

2) Дружні функції можна використати не тільки для перевантаження операторів, але й для будь-яких операцій. Відповідно, може мати довільне ім'я і довільну кількість параметрів, серед яких бодай один має бути параметром-об'єктом, - інакше немає змісту робити її дружньою.

