Лекція 8

Тема лекції: Перевантаження оператора індексування масиву, оператора виклику функції та оператора доступу до члена класу. Перевантаження унарних операторів інкременту та декременту. Правила перевантаження операторів

Оператор індексування масиву

Можливе перевантаження оператора індексування масиву *[]* для реалізації доступу до даних-членів класу, на зразок доступу до елементів масиву. Ці дані-члени можуть мати вигляд окремих членів або списку. Приклад перевантаження оператора *[]* для класу, що запам'ятовує чотири цілих значення у вигляді окремих даних-членів.

```
class PseudoArray
{
 private:
  int val0,val11,val2,val3;
 public:
  PseudoArray(int v0, int v1, int v2, int v3)
   {
    val0 = v0;
    val1 = v1;
    val2 = v2;
    val3 = v3;
   }
  int GetInt(unsigned i);
  int operator[](unsigned i);
};
main()
{
 PseudoArray pa(10,20,30,40);
 for(int i=0;i<=3;i++) cout << pa[i] << '\n';
 return 0;
```

```
}
int PseudoArray::GetInt(unsigned i)
 {
switch(i)
{
    case 0: return val0;
    case 1: return val1;
    case 2: return val2;
    case 3: return val3;
    default:return val0;
}
  }
int PseudoArray::operator[](unsigned i)
{
 return GetInt(i);
}
```

Перевантаження оператора індексування дає можливість використовувати в масиві індексацію замість виклику функції-члена для об’єкту класу. У цьому прикладі замість

```
cout << pa[i] << '\n';
```

можна було б написати:

```
cout << pa.GetInt(i);
```

Вираз *pa[i]* забезпечує доступ до масиву, незважаючи на те, що *pa* у дійсності посилається на об'єкт класу. Індекси перевантажених масивів можуть бути довільними типами даних. Наприклад, можна використовувати функцію:

```
int operator[](char c);
```

і потім оперувати символьними індексами для індексації масиву з початковим індексом *'A'*. У звичайних масивах C та C++ використовують цілі індексні значення, що починаються з нуля. Перевантажуючи *[]* і використовуючи символи в якості індексів, ми встановлюємо відповідність між множиною символів і множиною значень, створюючи асоціативний масив.

Можлива така реалізація функції-члена перевантаження оператора *[]*:

```
int PseudoArray::operator[](char c)
 {
  return GetInt(c-'A');
```

```
 }
```

Після цього можна використовувати символьну змінну в якості індексу масиву:

```
for(char c= 'A';c<='D';c++)
cout<< pa[c]<<'\n';
```

Оператор виклику функції

Перевантаження оператора виклику функції *operator()* робить об'єкт класу схожим на функцію, яку можна викликати. Перевантажена функція () може повертати значення заданих типів або зовсім нічого не повертати. В ній також можуть оголошуватись параметри. Вона не може бути статичним членом класу.

Приклад: клас із перевантаженим оператором виклику функції, що повертає ціле значення:

```
class TAnyClass
{
 int x;
 public:
  int operator()(void);
  TAnyClass(int n)
   {
    x=n;
   }
 };
 int TAnyClass::operator()(void)
 {
  return x;
 }
main()
{
 TAnyClass object(100);
 int g = object();
 cout << g;
 return 0;
```

```
}
```

У класі *TAnyClass* перевантажується оператор виклику функції за допомогою оголошення

```
int operator()(void);
```

Можна замінити *void* списком параметрів. Ця функція повертає ціле значення члена *x* об'єкта *TAnyClass*. У програмі оператор

```
int g = object();
```

схожий на виклик функції з ім’ям *object()*, але насправді виконується оператор :

```
object.operator()();
```

Оператор доступу до члена класу

Унарний оператор доступу до члена класу перевантажується як *operator ->()*. Він не може бути статичною функцією-членом класу.

```
class TAnyClass
{
 int x,y;
 public:
 TAnyClass(int xx,int yy)
  {
   x=xx;
   y=yy;
  }
  TAnyClass* operator->();
  int GetX(void)
  {
   return x;
  }
  int GetY(void)
  {
   return y;
   }
};
TAnyClass* TAnyClass::operator->()
```

```
{
 cout << "Доступний член:";
 return this;
}
main()
{
 TAnyClass t(100,200);
 cout << t->GetX() << '\n';
 cout << t->GetY() << '\n';
 return 0;
}
```

Функція-член *operator ->()* повинна повертати об'єкт, посилку або покажчик на об'єкт класу. У даному випадку вона повертає покажчик на клас *TAnyClass*. У цьому прикладі використовується перевантаження оператора -> для налагодження програми.

Перевантажений оператор виводить повідомлення перед тим, як повернути покажчик *this*, що посилається на об'єкт, для якого була викликана функція-член. У програмі в двох операторах виведення в потік використовується перевантажений оператор -> для доступу до функцій-членів *GetX()* і *GetY()* об'єкту *t* класу *TAnyClass*. Насправді ці оператори виконуються так:

```
cout << (t.operator->())->GetX() << '\n';
out << (t.operator->())->GetY() << '\n';
```

Програма відобразить помітки перед значеннями, що повертаються функціями *GetX() і GetY()*:

Доступний член: 100

Доступний член: 200

Оператори інкременту та декременту

Приклад перевантаження операторів інкременту та декременту:

```
class TAnyClass
{
 int x;
```

```
 public:
  TAnyClass(int xx)
  {
   x=xx;
  }
  int operator++()
  {
   return ++x;
  }
  int operator++(int)
  {
   return x++;
  }
  int GetX(void)
  {
   return x;
  }
};
```

Функція-член *operator++()* визначає префіксний оператор інкременту для об'єкту типу *TAnyClass*. Ця функція не має параметрів. Функція-член *operator++(int)* визначає постфіксний оператор інкременту для об'єкту типу *TAnyClass*. C++ привласнює 0 єдиному цілому параметру. Для об'єкту *v* типу *TAnyClass* вираз *++v* викличе перевантажений префіксний оператор ++. Насправді виконується оператор

```
v.operator++().
```

Вираз *v++* викликає перевантажений постфіксний оператор *v++* виконується як

```
v.operator++(int);
```

Оператори декременту працюють аналогічно.

Правила перевантаження операторів

• Необхідно забезпечити обмірковані функції перевантаження операторів, тому що C++ не розуміє семантики перевантаженого оператора.

• C++ не може виводити складні оператори з простих. У класі, де оголошені функції перевантаження операторів *operator*()* і *operator=()*, C++ не зможе обчислити вираз *a*=b* доти, доки не буде перевантажений оператор operator*=*().*

• Не можна змінити синтаксис перевантажених операторів. Бінарні оператори повинні залишатися бінарними, унарні - унарними. Наприклад, не можна створити унарне ділення, тому що такої вбудованої можливості в C++ не існує.

• Не можна винаходити нові оператори.

• Не можна перевантажувати символи препроцесора # *і* ##.