Статичні члени класів

Статичні дані-члени (поля) класів використовуються для збереження даних, спільних для всіх екземплярів класу, наприклад, кількість екземплярів класу, вказівник на вершину динамічного списку, деяку глобальну для всього класу константу, тощо. Статичний член класу має бути продекларованим у класі з допомогою службового слова **`static`**, а процес виділення під нього пам'яті та його ініціалізація відбувається поза класом.

Звертання до статичних членів можливе через ім'я класу або через ідентифікатор екземпляру (В C# - тільки через ім'я класу).

На статичні члени розповсюджуються звичайні правила доступу.

Слід зауважити, що в класі присутня лише декларація статичного члену, для його створення необхідно виділити пам’ять під нього та в разі необхідності проініціалізувати – це відбувається ***поза межами класу***, ***навіть якщо статичний член задекларований як закритий***.

Більше того, якщо статичний член класу (скалярного типу) помічений службовим словом `const`, то він може бути проініціалізований в класі, але пам’ять під нього все рівно має бути виділена поза класом!

Операція `sizeof` не враховує пам'ять, виділену під статичні поля.

Приклад.

```
class Example
{
 public :
   static int num;  // декларація статичного члену класу
   int key;
   Example (int key_) : key (key_) {}
};
//////////////////////////////////////////////////////////////
int Example :: num; // виділення пам'яті під статичний член
                    // в разі відсутності ініціалізації він = 0
//////////////////////////////////////////////////////////////
int main(int argc, char *argv[])
{
  Example e (1), f (10);
  cout << Example :: num << endl; // звертання через ім'я класу
  Example :: num = 100;
  cout << e.num << endl;       // звертання через ім'я екземпляру
  cout << f.num << endl;    // звертання через ім'я екземпляру
  e.num = 1000;
  cout << e.num << endl;
  cout << f.num << endl;
  system("PAUSE");
  return 0;
}
```

Статичні функції-члени класів використовуються <u>тільки</u> для звертання до статичних даних-членів і не можуть використовувати звичайні дані та методи класу, адже вони не прив'язані до екземпляру, їм не передається вказівник **`this`**.

Службове слово **`static`** вказується лише у декларації статичної функції, при її визначенні воно <u>не повторюється</u>.

Звертання до статичних методів так само може відбуватись і через ім'я класу, і через ідентифікатор екземпляру.

Слід зауважити, що звичайні функції-члени класу мають право працювати із статичними членами класу.

Конструктор та деструктор в C++ не можуть бути статичними!

Спадкування, похідні класи.

Спадкування – це один з основних принципів об'єктно-орієнтованого програмування, який дозволяє створювати об'єкти, що спадкують свої властивості від існуючих об'єктів, додаючи власної функціональності.

Похідний та базовий клас пов’язані співвідношенням «Є» («is-a») : якщо **SubBase** – похідний клас від класу **Base**, то **SubBase** «є» **Base**. Синтаксис визначення похідного класу:

```
class SubBase : <вид_спадкування> Base
{
  // тіло похідного класу
};
```

Деякі правила спадкування.

1.Похідний клас спадкує (містить) всі відкриті (`public`) та захищені (`protected`) члени базового класу. Закриті (`private`) члени базового класу недоступні у похідному.

2.Конструктори та деструктор **не спадкуються** похідним класом.

3.При створенні екземпляру похідного класу автоматично викликається конструктор базового класу і лише потім конструктор похідного. При знищенні екземпляру деструктори викликаються у зворотному порядку.

4.Якщо у похідному класі відсутній безпосередній виклик конструктора базового класу, то викликається конструктор за умовчанням (без параметрів) базового класу. В разі його відсутності виникає помилка.

5.Не спадкується також операція присвоєння, якщо вона була перевантажена у базовому класі.

Доступ до членів класу при різних видах спадкування

Види спадкування (не використовувались в мові C#) та доступ до членів базового класу у похідному класі зазначені у наступній таблиці:

Вид спадкування	**Доступ у базовому класі**	**Доступ у похідному класі**

`public`	`private`
	`protected`
	`public`
		не доступний
		`protected`
		`public`

`protected`	`private`
	`protected`
	`public`
		не доступний
		`protected`
		`protected`

`private` (діє за умовчанням)	`private`
	`protected`
	`public`
		не доступний
		`private`
		`private`

Приклад.

```
class Base
{
protected :          // закриті члени класу
   double money;
   int key;
public :             // відкриті члени класу
   Base (double money_ = 1000, int key_ = 1):
        money (money_), key (key_)
   {cout << "Create Base" << endl;}
   double show_money () {return money;}
   int show_key () {return key;}
};
class SubBase : public Base
{
public :
 // Тут автоматично викликається конструктор базового класу
SubBase (double money_, int key_)
{
    cout << "Create SubBase" << endl;
    money = money_; key = key_;
}
};
```

Виклик конструктора базового класу.

```
class Base
{
protected :          // закриті члени класу
   double money;
   int key;
public :             // відкриті члени класу
   Base (double money_ = 1000, int key_ = 1):
        money (money_), key (key_)
   { }
   double show_money () {return money;}
   int show_key () {return key;}
};
class SubBase : public Base
{
public :
 // Тут явно викликається конструктор базового класу
SubBase (double money_, int key_): Base(money_, key_)
   { }
};
```

Зміна доступу до членів базового класу у похідному класі.

Оголошення доступу у похідних класах дають можливість:

- зробити знову відкритими або захищеними відповідно захищені або відкриті члени базового класу у похідному класі;
- зробити знову відкритими відкриті члени базового класу у закритому або захищеному похідному класі;

Приклад.

```
class Base
{
 private :              // закриті члени базового класу
   double money;
 protected :            // захищені члени базового класу
   int key;
 public:
   char symb;
};
class SubBase : public Base
{
 public :
   Base :: key;     // захищений член базового класу –
                    // тепер відкритий у похідному
   Base :: money;   // Помилка! В похідному класі немає
            // доступу до закритого члена базового класу
 protected :
   Base :: symb; // у похідному класі він захищений
};
```

Зауваження про ініціалізацію екземплярів класу.

При створенні масиву екземплярів класу виникає необхідність у конструкторі за замовчуванням (без параметрів):

```
class Student
{
 private :
// закриті члени класу
 public :
  Student (double ball_, double exam_,
            char * name_ = "NoName");
// конструктор за замовчуванням
  Student (char * name_ = "NoName");
  ~Student ();
};
int main ()
{
  Student grup_1 [25]; // виклик констр. за замовч.
  return 0;
}
```

А якщо необхідний масив з проініціалізованими полями екземплярів? В такому разі можливо передавати конструкторам аргументи наступним чином:

```
int main ()
{
   Student bad [] = {
   Student (30,25, “Ivan"), // явний виклик
   Student (32,27, “Oleg"),// явний виклик
   Student (25,30, "Maria") // явний виклик
   };
   return 0;
}
```

Тут створений масив із трьох елементів, кожний з яких ініціалізується завдяки явному виклику конструктора з параметрами.

Список ініціалізації.

В багатьох випадках, особливо, коли членом класу є екземпляр деякого іншого класу, зручно ініціалізувати члени класу так званим <u>списком ініціалізації конструктора</u>. Він вказується прямо після сигнатури конструктора (тобто після круглої дужки, що закриває його список параметрів) і відокремлюється від неї двокрапкою. В списку вказується ідентифікатор члену класу, а в дужках – початкове значення для нього. Елементи списку відокремлюються комами:

```
class Special_Student
{
 Student s;
 int key;
 public :
 Special_Student (Student s_, k) :
   s (s_), key (k)  // Це список ініціалізації
 { // тіло конструктора порожнє!
 }
};
int main ()
{
  Student s;
  Special_Student c (s);
  return 0;
}
```

В більшості випадків такий спосіб не є обов'язковим, і може бути замінений на явне присвоєння в тілі конструктора. Проте є ситуації, коли присвоєння неможливе:

```
class X
{
  int i;
public:
  X (int i_) {i = i_;}
};
//////////////////////////////
class Y
{
   const int val; // членом класу є константа
   X xx;  // членом класу є екземпляр деякого класу
   X & ref_x;  // членом класу є ref-змінна
public:
  Y (int v, X x, X& rx):
     val (v), xx (x),
     ref_x (rx)    // ref_x не допускає присвоєння
  {}
};
```