Лекція 4. Конструктори і деструктори

Конструктор – це функція, яка викликається при створенні об'єкта.

Як правило, деяку частину об'єкта, перш ніж його можна буде використовувати, необхідно ініціалізувати. Наприклад, розглянемо клас Cat (він представлений вище в цьому розділі). Перш ніж клас Cat можна буде застосовувати, змінним ім’я name і tail потрібно присвоїти початкові значення. В даному конкретному випадку це виконується за допомогою функції init(). Крім того, для визначення змінних name і tail використовуються функції SetName і SetTail. Але, оскільки вимога ініціалізації членів класу вельми поширена, в C++ передбачена реалізація цієї можливості при створенні об'єктів класу. Така автоматична ініціалізація виконується завдяки використанню конструктора.

Конструктор – це спеціальна функція, яка є членом класу і ім'я якої збігається з ім'ям класу. Ось, наприклад, як став виглядати клас Cat після переробки, пов'язаної із застосуванням конструктора для його ініціалізації:

```
// Створення класу Cat
class Cat
{
 string name; // ім'я кота
 int tail; // довжина хвоста кота
public:
 Cat();  // конструктор
 void SetName(string name);
 string GetName();
 void SetTail(int tail);
 int GetTail();
};
```

Зверніть увагу на те, що в оголошенні конструктора Cat() відсутній тип значення, що повертається. Конструктори ніколи не повертають значень і, отже, немає сенсу у вказуванні їх типу. (При цьому не можна вказувати навіть тип void.)

Тепер наведемо код функції Cat ().

```
// Визначення конструктора
Cat::Cat()
{
 name = "Томас";
 tail = 25;
 cout << "З'явився об'єкт класу Cat на ім'я " << name << endl;
}
```

Конструктор об'єкту викликається при створенні об'єкта. Це означає, що він викликається при виконанні інструкції оголошення об'єкта. Конструктори глобальних об'єктів викликаються на самому початку виконання програми, ще до звернення до функції main(). Що стосується локальних об'єктів, то їх конструктори викликаються кожен раз, коли зустрічається оголошення такого об'єкта.

Якщо клас немає конструкторів, то компілятор C++ автоматично згенерує для цього класу відкритий конструктор за замовчуванням (default constructor). Його іноді називають неявним конструктором (або "неявно згенерованим конструктором"). Так, у прикладі, де клас Cat не має конструктора, компілятор згенерує наступний конструктор:

```
public:
Cat()
{
}
```

Цей конструктор дозволяє створювати об'єкти класу, але не виконує їх ініціалізацію та не надає значення членам класу. Якщо клас має інші конструктори, то конструктор за замовчуванням створюватися не буде.

Деструктор – це функція, яка викликається при руйнуванні об'єкта.

У багатьох випадках при руйнуванні об'єкта необхідно виконати деяку дію або навіть деяку послідовність дій. Локальні об'єкти створюються при вході в блок, в якому вони визначені, і руйнуються при виході з нього. Глобальні об'єкти руйнуються при завершенні програми. Існує безліч факторів, що обумовлюють необхідність деструкції. Наприклад, об'єкт повинен звільнити раніше виділену для нього пам'ять. У C++ саме деструкторам доручається

обробка процесу дезактівізация об'єкта. Ім'я деструктора збігається з ім'ям конструктора, але передує символом "~" Подібно конструкторам деструктори не повертають значень, отже, в їх оголошеннях відсутня тип значення. Розглянемо вже знайомий нам клас Cat, але тепер він містить конструктор і деструктор.

```
// Створення класу Cat
class Cat
{
 string name; // ім'я кота
 int tail; // довжина хвоста кота
public:
 Cat();  // конструктор
 ~Cat(); // деструктор
 void SetName(string name);
 string GetName();
 void SetTail(int tail);
 int GetTail();
};
```

Ось як виглядає нова версія програми, в якій демонструється використання конструктора і деструктора.

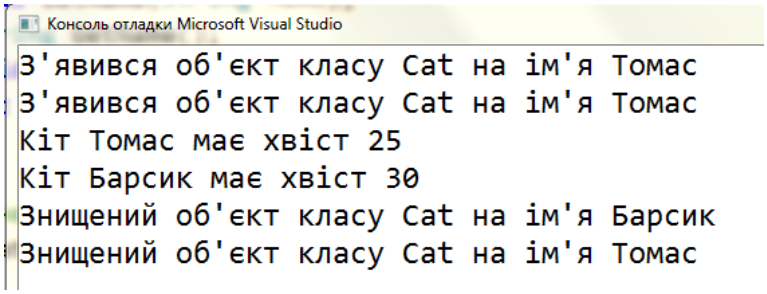
```
// Визначення конструктора
Cat::Cat()
{
 name = "Томас";
 tail = 25;
 cout << "З'явився об'єкт класу Cat на ім'я " << name << endl;
}
// Визначення деструктора
Cat::~Cat()
{
 cout << "Знищений об'єкт класу Cat на ім'я " << name << endl;
}

void Cat::SetName(string name)
```

```
{
 this->name = name;
}
string Cat::GetName()
{
 return name;
}
void Cat::SetTail(int tail)
{
 this->tail = tail;
}
int Cat::GetTail()
{
 return tail;
}

int main()
{
 setlocale(0, "ukr");
 Cat Tomas, Barsik;
 Tomas.init();
 cout << "Кіт " << Tomas.GetName() << " має хвіст ";
 cout << Tomas.GetTail() << endl;
 Barsik.SetName("Барсик");
 Barsik.SetTail(30);
 cout << "Кіт " << Barsik.GetName() << " має хвіст ";
 cout << Barsik.GetTail() << endl;
}
```

При виконанні цієї програми виходять такі результати:



Рекомендується завжди створювати принаймні один конструктор у класі. Це дозволить вам контролювати процес створення об'єктів вашого класу, і запобігатиме виникненню потенційних проблем після додавання інших конструкторів.

Параметризовані конструктори

Конструктор може мати параметри. З їх допомогою при створенні об'єкта членам даних (змінним класу) можна привласнити деякі початкові значення, які визначаються в програмі. Це реалізується шляхом передачі аргументів конструктору об'єкта. У наступному прикладі ми вдосконалюємо клас Cat так, щоб він приймав аргументи, які будуть визначати значення змінних name і tail. Перш за все необхідно внести зміни в визначення класу Cat. Тепер воно виглядає так:

```
// Створення класу Cat
class Cat
{
 string name; // ім'я кота
 int tail; // довжина хвоста кота
public:
 Cat(string name, int tail); // параметризований конструктор
 ~Cat(); // деструктор (завжди без параметрів)
 void SetName(string name);
 string GetName();
 void SetTail(int tail);
 int GetTail();
};
```

Запам’ятайте, що деструктор завжди немає жодних параметрів.

Конструктор класу Cat виглядає тепер таким чином:

```
// Визначення конструктора
Cat::Cat(string name, int tail)
{
 this->name = name;
 this->tail = tail;
 cout << "З'явився об'єкт класу Cat на ім'я " << name << endl;
}
```

Щоб передати аргумент конструктору, необхідно зв’язати цей аргумент з об'єктом при оголошенні об'єкту. C++ підтримує два способи реалізації такого зв'язування. Ось як виглядає перший спосіб:

```
Cat Tomas = Cat("Томас", 25);
```

Але ця форма (в такому контексті) використовується рідко, оскільки другий спосіб має більш короткий запис і зручніше для використання.

У другому способі аргумент повинен слідувати за ім'ям об'єкта і укладатися у круглі дужки. Наприклад, така інструкція еквівалентна попередньому оголошенню:

```
Cat Tomas("Томас", 25);
```

Це найпоширеніший спосіб оголошення параметризованих об'єктів. Спираючись на цей метод, наведемо загальний формат передачі аргументів конструкторам:

тіп_класу ім'я_змінної (список_аргументів);

Тут список_аргументів – це список розділених комами параметрів, переданих конструктору.

У наступній версії програми демонструється використання параметризованого конструктора.

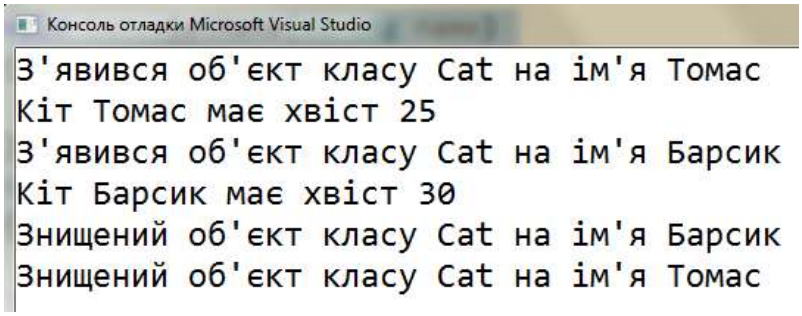
```
// Визначення конструктора
Cat::Cat(string name, int tail) // параметризований конструктор
{
 this->name = name;
 this->tail = tail;
 cout << "З'явився об'єкт класу Cat на ім'я " << name << endl;
}

// Визначення деструктора (завжди без параметрів)
Cat::~Cat()
{
 cout << "Знищений об'єкт класу Cat на ім'я " << name << endl;
}
void Cat::SetName(string name)
{
 this->name = name;
```

```
}
string Cat::GetName()
{
 return name;
}
void Cat::SetTail(int tail)
{
 this->tail = tail;
}
int Cat::GetTail()
{
 return tail;
}

int main()
{
 setlocale(0, "ukr");
 Cat Tomas("Томас", 25);
 cout << "Кіт " << Tomas.GetName() << " має хвіст ";
 cout << Tomas.GetTail() << endl;
 Cat Barsik("Барсик", 30);
 cout << "Кіт " << Barsik.GetName() << " має хвіст ";
 cout << Barsik.GetTail() << endl;
}
```

При виконанні цієї програми виходять такі результати:



Альтернативний варіант ініціалізації об'єкта.

Якщо конструктор приймає тільки один параметр, можна використовувати альтернативний спосіб ініціалізації членів об'єкта. Розглянемо наступну програму.

```
class Pryklad {
 int data;
public:
 Pryklad(int param);
 int GetData();
};
Pryklad::Pryklad(int x)
{
 data = x;
}
int Pryklad::GetData()
{
 return data;
}
int main()
{
 Pryklad ob = 4; //виклик функції Pryklad(4)
 cout << ob.GetData();
}
```

Тут конструктор для об'єктів класу Pryklad приймає тільки один параметр. Зверніть увагу на те, як у функції main() оголошується об'єкт ob. Для цього використовується такий формат оголошення:

```
Pryklad ob = 4;
```

У цій формі ініціалізації об'єкта число 4 автоматично передається параметру data при виклику конструктора Pryklad(4). Іншими словами, ця інструкція оголошення обробляється компілятором так, якщо б вона була записана в такий спосіб.

```
Pryklad ob = Pryklad(4);
```