Лекція 2

Тема: Конструктори та деструктори, виклик конструкторів.

ПЛАН

1. **Конструктори й деструктори**
2. **Конструктори з параметрами**
3. **Конструктори з одним параметром: особливий випадок**
4. **Виклик конструкторів і деструкторів**
5. **Конструктор копіювання**

1. Конструктори й деструктори

Дуже часто деяка частина об'єкта перед його першим використанням повинна бути ініціалізована. Наприклад, повернемося до опису класу stack. Перед першим використанням об'єкта цього класу змінної tos слід привласнити число 0. Для цього призначена функція init(). Оскільки ініціалізація об'єктів - дуже розповсюджена вимога, у мові C++ передбачений особливий механізм, що дозволяє ініціалізувати об'єкти в момент їх створення. Ця автоматична ініціалізація здійснюється конструктором.

Конструктор - це особлива функція, що є членом класу. Її ім'я повинне збігатися з іменем класу. Наприклад, клас stack можна модифікувати, передбачивши в ньому конструктор.

```
// Клас stack з конструктором.
class stack {
       int stck[SIZE];
       int tos;
public:
       stack();  // Конструктор
       void push(int i);
       int pop();
    };
```

Зверніть увагу на те, що в оголошенні конструктора stack () не зазначений тип значення, що вертається, оскільки в мові C++ конструктори в принципі не можуть повертати значення.

Код конструктора stack () може виглядати так.

```
// Конструктор класу
stack stack::stack() {
       tos = 0; cout << "Стек ініціалізований \n";
    }
```

Урахуйте, що повідомлення "Стек ініціалізований" просто ілюструє роботу конструктора. На практиці конструктори найчастіше нічого не вводять і не виводять: вони просто виконують різні види ініціалізації.

Конструктор автоматично викликається в момент створення об'єкта, тобто при його оголошенні. Отже, оголошення об'єкта в мові C++ - це не пасивний запис, а активний процес. У мові C++ оголошення є виконуваним оператором. Ця відмінність носить зовсім не академічний характер. Код, за допомогою якого конструюється об'єкт, може бути досить важливим. Для глобальних і статичних локальних об'єктів конструктори викликаються лише одного разу. При оголошенні локальних об'єктів конструктори викликаються щораз при вході у відповідний блок.

Антиподом конструктора є деструктор. У багатьох ситуаціях об'єкт повинен виконати деяка дія або дії, які знищать його. Локальні об'єкти створюються при вході у відповідний блок, а при виході з нього вони знищуються. При руйнуванні об'єкта автоматично викликається його деструктор. Для цього існує кілька причин. Наприклад, об'єкт повинен звільнити займану їм пам'ять або закрити файл, відкритий їм раніше. У мові C++ ці дії виконує деструктор. Ім'я деструктора повинне збігатися з іменем конструктора, але перед ним ставиться знак ~ (тильда). Розглянемо клас stack конструктор, що містить, і деструктор. (Урахуйте, що насправді клас stack не має потреби в деструкторі, тут він наведений як ілюстрація.)

```
//  Оголошення класу  stack,
class  stack {
       int  stck[SIZE];
       int   tos; public:
              stack();     //  Конструктор
              ~stack();   //  Деструктор
              void push(int   i);
              int  pop();
};
// Конструктор класу stack
stack::stack()
{
       tos = 0;
       cout << " Стек ініціалізований ";
}
// Деструктор класу stack
stack::~stack()
{
       cout << " Стек знищений\n";
    }
```

Врахуйте, що конструктори й деструктори не повертають ніяких значень.

Проілюструємо роботу конструктора й деструктора за допомогою нової версії програми stack. Зверніть увагу на те, що тепер функція init() не потрібна.

```
//  Використання конструктора й деструктора.
#include <iostream>
using namespace std;
#define SIZE  100

// Створюємо клас stack,
class stack {
       int stck[SIZE];
       int tos;
public:
       stack();  // Конструктор
       ~stack(); // Деструктор
       void push(int i);
```

```
	int pop();
};
// Конструктор класу stack
stack::stack()
{
	tos = 0;
	cout << " Стек ініціалізований \n";
}
// Деструктор класу stack
stack::~stack()
{
	cout << " Стек знищений\n";
}
void stack::push(int i) {
	if (tos == SIZE) {
		cout << " Стек повний.\n";
		return;
	}
	stck[tos] = i; tos++;
}
int stack::pop() {
	if (tos == 0) {
		cout << " Стек порожній.\n"; return 0;
	}
	tos--;
	return stck[tos];
}
int main() {
	stack a, b;  // Створюємо два об'єкти класу stack
	a.push(1); b.push(2);
	a.push(3); b.push(4);
	cout << a.popo << "   ";
	cout << a.popo << "   ";
	cout << b.popo << "   ";
	cout << b.popo << "\n";
	return 0;
}
```

Результати роботи програми:
Стек ініціалізований
Стек ініціалізований
3 1 4 2
Стек знищений
Стек знищений

2. Конструктори з параметрами

Конструкторам можна передавати аргументи, призначені для ініціалізації об'єкта. Параметри конструктора задаються так само, як і для будь-якої іншої функції. Розглянемо клас, конструктор якого має параметри.

```
#include <iostream>
using namespace std;
class myclass {
	int a, b; public:
		myclass(int i, int j) { a = i; b = j; }
		void show() { cout << a << " " << b; }
};
int main() {
	myclass ob(3, 5);
	ob.show();
```

```
    return 0;
}
```

Зверніть увагу на те, що у визначенні конструктора myclass () параметри і й j використовуються для ініціалізації членів a й b.

Ця програма ілюструє найпоширеніший спосіб завдання аргументів при оголошенні об'єкта, що використовує конструктор з параметрами. Наприклад, оператор

```
myclass   ob(3, 4);
```

створює об'єкт ob і передає аргументам і й j конструктора myclass () значення 3 і 4. Існує ще один спосіб передачі аргументів:

```
myclass  ob = myclass(3, 4);
```

Однак перший спосіб використовується набагато частіше. Насправді різниця між цими двома оголошеннями невелика.

Розглянемо приклад, що ілюструє застосування конструктора з параметрами. У ньому оголошений клас, об'єкти якого зберігають інформацію про бібліотечні книги.

```
#include <iostream>
#include <cstring>
using namespace std;
const int IN = 1;
consc int CHECKED_OUT = 0;
class book {
       char author[40];
       char title[40];
       int status;
public:
       book(char *n, char *t, int s);
       int get_status() { return status; }
       void set_status(int s) { status = s; }
       void show();
};
book::book(char *n, char *t, int s) {
       strcpy(author, n);
       strcpy(title, t), -
              status = s;
}
void book::show() {
       cout << title << "." << author;
       cout << " перебуває ";
       if (status == IN) cout << "у бібліотеці.\n";
       else cout << "у читача.\n";
}
int main() {
       book b1("Твен", "ТОМСОЙЕР", IN);
       book b2("мелвилл", "МОБИДИК", CHECKED_OUT);
       b1.show(); b2.show();
       return  0;
    }
```

Конструктори з параметрами дозволяють уникнути застосування окремих функцій для ініціалізації однієї або декількох змінних в об'єкті. Чим менше функцій викликається в програмі, тем вище її ефективність. Крім того, зверніть увагу на те, що короткі функції get_status() і set_status() визначені усередині оголошення класу book, тобто зроблені, що підставляються. Це приймання часто застосовується при створенні програм мовою C++.

3. Конструктори з одним параметром: особливий випадок

Якщо конструктор має один параметр, виникає третій спосіб передачі початкового значення. Розглянемо наступний приклад.

```
#include <iostream>
using namespace std;
class X {
	int a;
public:
	X(int j) { a = j; }
	int geta() { return a; }
};
int main() {
	X ob = 99; // Передає параметру j значення 99.
	cout << ob.geta(); // Виводить на екран число 99.
	return 0;
    }
```

Тут конструктор класу x має один параметр. Звернете особливу увагу на спосіб оголошення об'єкта obь у функції main (). У цьому випадку число 99 автоматично передається параметру j конструктора x(). Інакше кажучи, компілятор уважає цей оператор еквівалентним наступному:

X ob = X(99);

Як правило, якщо конструктор має один аргумент, об'єкт можна ініціалізувати або за допомогою вираження ob(i), або за допомогою оператора ob=j. Це дозволяє неявно виконувати перетворення типу аргументу в тип класу.

Памятайте, що ця альтернатива ставиться лише до конструкторів, що мають тільки один параметр.

4. Виклик конструкторів і деструкторів

Як правило, конструктор об'єкта викликається при створенні об'єкта, а деструктор - при його знищенні. Точний зміст цих подій ми розглянемо нижче

Конструктори локальних об'єктів послідовно викликаються при їх оголошенні. Деструктори локальних об'єктів викликаються у зворотному порядку.

Конструктори глобальних об'єктів виконуються до функції main(). Глобальні конструктори виконуються в порядку їх оголошення у файлі. Порядок виклику глобальних конструкторів, розподілених у декількох файлах, заздалегідь не визначений. Глобальні деструктори викликаються у зворотному порядку після завершення роботи функції main().

Виклики конструкторів і деструкторів ілюструються наступною програмою.

```
#include <iostream>
using namespace std;
class myclass {
public:
	int who;
	myclass(int id);
	~myclass();
} glob_obl(1), glob_ob2(2);
myclass::myclass(int id) {
	cout << "Ініціалізація " << id << "\n";
```

```
        who = id;
}
myclass::~myclass() {
        cout << "Знищення " << who << "\n";
}
int main() {
        myclass local_obl(3);
        cout << "Цей рядок уже не буде першим.\n";
        myclass local_ob2(4);
        return 0;
    }
```

У результаті на екрані з'являться наступні рядки.

Ініціалізація 1
Ініціалізація 2
Ініціалізація 3
Цей рядок уже не буде першим.
Ініціалізація 4
Знищення 4
Знищення 3
Знищення 2
Знищення 1

І ще: оскільки компілятори й операційні системи відрізняються один від одного, останні два рядки не завжди з'являються на екрані.

5. Конструктор копіювання

Одним з найважливіших видів перевантаженого конструктора є конструктор копіювання (copy constructor). Він дозволяє запобігти проблемам, які можуть виникнути при присвоєнні одного об'єкта іншому.

Розглянемо проблему, для вирішення якої необхідний конструктор копіювання. За замовчуванням, якщо один об'єкт ініціалізується іншим, створюється побітова копія присвоюваного об'єкту. Інакше кажучи, ініціалізованому об'єкту присвоюється його ідентична копія. Іноді побітова копія виявляється неприйнятною. Зазвичай це відбувається, коли об'єкт виділяє динамічну пам'ять. Наприклад, припустимо, що клас MyClass виділяє динамічну пам'ять для кожного створюваного об'єкта, а об'єкт A є його екземпляром. Це означає, що об'єкт A вже виділив для себе динамічну пам'ять. Тепер припустимо, що об'єкт A ініціалізує об'єкт B, як показано нижче.

```
myclass B = A;
```

Якщо при цьому створюється побітова копія об'єкта A, то об'єкт B буде точною копією об'єкта A. Отже, об'єкт B також буде посилатися на область пам'яті, виділену об'єктом A, не виділяючи свою власну ділянку пам’яті. Очевидно, це зовсім не те, до чого ми прагнули. Наприклад, якщо клас MyClass містить деструктор, який звільняє пам'ять, то при знищенні об'єктів A й B одна і та ж область пам'яті буде звільнятися двічі!

Ця проблема може виникнути ще в двох ситуаціях. По-перше, коли функції

передається копія параметра, і, по-друге, коли створюється тимчасовий об'єкт, що повертається функцією. Нагадаємо, що функція створює тимчасовий об'єкт, в якому зберігається значення, що повертається функцією.

Для вирішення цих проблем призначений конструктор копіювання, який не використовує побітове копіювання. Найбільш часто застосовується така його форма.

```
ім’я_класу(const ім’я_класу &посилання_на_об’єкт) {
	// Тіло конструктора
}
```

Тут посилання на об'єкт пов'язане з об'єктом, який розташований в правій частині ініціалізації. Конструктор копіювання може мати додаткові параметри. Однак в будь-якому випадку першим параметром має бути посилання на ініціалізуючий об'єкт.

Слід пам'ятати, що існують дві ситуації, в яких один об'єкт може присвоюватися іншому. По-перше, при виконанні оператора присвоювання. Подруге, при ініціалізації, яку можна здійснити трьома способами.

- Явна ініціалізація, наприклад, при оголошенні об'єкту.
- Створення копії об'єкта, переданого функції як параметр.
- Створення тимчасового об'єкта (найчастіше, при поверненні значення функції).

Конструктор копіювання застосовується тільки для ініціалізації. Розглянемо клас myclass і його об'єкт Y. Ініціалізація відбуватиметься при виконанні кожного з цих операторів.

```
myclass x = y; // Об'єкт у явно ініціалізовується об'єктом x
func(y); // Об'єкт у передається як параметр
y = func(); // Об'єкту у присвоюється об'єкт, що повертається функцією
```

Розглянемо програму, в якій необхідний явний конструктор копіювання. Ця програма створює дуже обмежений і "безпечний" тип цілочислового масиву, що запобігає виходу індексу за межі допустимого діапазону. За допомогою оператора new кожному елементу масиву виділяється динамічна пам'ять. Причому вказівник на виділену область пам'яті зберігається всередині кожного об'єкта.

```
/*Програма створює "безпечний" тип масиву.
Оскільки пам'ять для кожного елемента масиву виділяється за допомогою оператора new, для їх
ініціалізації застосовується конструктор копіювання.
*/
#include <iostream>
#include <new>
#include <cstdlib>
using namespace std;
class array {
	int *p;
	int size;
public:
	array(int sz) {
		try {
			p = new int[sz];
		}
		catch (bad_alloc xa) {
			cout << "Виняткова ситуація \n";
```

```
                        exit(EXIT_FAILURE);
                }
                size = sz;
        }
        ~array() { delete[].p; }
        // Конструктор копиювання
        array(const array &a);
        void put(int i, int j) {
                if (i >= 0 && i<size) p[i] = j;
        }
        int get(int i) {
                return p[i];
        }
};
// Конструктор копіювання
array::array(const array &a) {
        int i;
        try {
                p = new int[a.size];
        }
        catch (bad_alloc xa) {
                cout << " Виняткова ситуація \n";
                exit(EXIT_FAILURE);
        }
        for (i = 0; i<a.size; i++) p[i] = a.p[i];
}
int main()
{
        array num(10);
        int i;
        for (i = 0; i<10; i++) num.put(i, i);
        for (i = 9; i >= 0; i--) cout << num.get(i);
        cout << "\n";
        // Створюється новий масив, який ініціалізується масивом num
        array x(num); // Виклик конструктора копіювання
        for (i = 0; i<10; i++) cout << x.get(i);
        return 0;
}
```

Розглянемо, що станеться, коли масив num ініціалізує масив x за допомогою оператора
array x (num); // Виклик конструктора копіювання

При виклику конструктора копіювання для нового масиву виділяється область динамічної пам'яті. Вказівник на її перший елемент зберігається в вказівнику x.p. Таким чином, об'єкти x і num містять однакові елементи, проте кожен з них займає окрему область пам'яті. Якби в програмі не використовувався конструктор копіювання, об'єкти x і num займали б одну область динамічної пам'яті. (Це означає, що вказівники num.p і x.p містили б одну і туж адресу.)

Нагадаємо, що конструктор копіювання викликається тільки для ініціалізації. Наприклад, такі оператори не викликають конструктор копіювання.

```
array a(10);
array b(10);
b = a; // Конструктор копіювання не викликається
```

В даному випадку оператор b = a виконує присвоювання одного об'єкта іншому. Якщо оператор присвоювання не перевантажений (як в даному випадку), створюється побітова копія об'єкта, розташованого в його правій частині. Отже, в деяких ситуаціях виникає необхідність не тільки передбачити конструктор копіювання, але і перевантажити оператор присвоювання