Лабораторне заняття 3 (2-й семестр)

Конструктори класу. Деструктор класу.

Конструктор – це метод, який виконується автоматично в момент створення об'єкта. Він може не лише ініціалізовувати змінні класу, але й виконувати будь-які інші завдання ініціалізації, потрібні для підготовки об'єкта до використання. Для створення екземпляра класу потрібно, щоб конструктор був методом типу *public.*

Основні властивості конструктора.

- ✓ **Ім'я конструктора** – це **ім'я класу**. Отже, компілятор відрізняє конструктор від інших методів класу.
- ✓ Конструктор **не повертає жодного значення**, навіть типу void. Неможна здобути і покажчик на конструктор. Конструктор автоматично викликається системою, а отже, не існує програми чи функції, яка його викликає, і якій конструктор міг би повернути значення.
- ✓ **Клас** може мати **кілька конструкторів** з різними параметрами для різних видів ініціалізації, при цьому використовується механізм перевантаження.
- ✓ Конструктор може прийняти яку завгодно кількість аргументів (включаючи нульову).
- ✓ Параметри конструктора можуть бути якого завгодно типу, окрім цього класу. Можна задавати значення параметрів за замовчуванням. Їх може містити лише один з конструкторів.
- ✓ Конструктори не успадковуються.
- ✓ Конструктори не можна оголошувати з модифікаторами *const, virtual* та *static*.

Види конструкторів

- ➢ Конструктор з параметрами
- ➢ Конструктор зі списком ініціалізації.
- ➢ Конструктор за замовчуванням.
- ➢ Конструктор копіювання.

Конструктор з параметрами

Конструктор може ініціалізувати поля класу через параметри, наприклад

```
someClass (int mm1, int mm2, int mm3)
{ m1 = mm1; m2 = mm2; m3 = mm3;}
```

У цьому разі створення об'єкта може мати вигляд

```
someClass obj (5, 20, 13);
```

Тут викликається конструктор, до якого передаються відповідні значення параметрів. Окрім того, при створюванні екземпляра класу значення параметрів можна передавати конструктору, використовуючи оператор `new`, наприклад:

```
someClass *Pobj = new someClass (5, 20, 13);
```

Конструктор може мати параметри за замовчуванням, які передаються автоматично , наприклад полям класу, при створенні екземплярів класу. Якщо, при створені екземпляру класу значення параметрів конструктора не вказуються, об'єкт ініціалізується зі значеннями параметрів за замовчуванням.

```
someClass (int mm1=1, int mm2=1, int mm3=1)
{ m1 = mm1; m2 = mm2; m3 = mm3;}
someClass obj;
```

Конструктор зі списком ініціалізації

Вище описаний конструктор не може бути використаним при ініціалізації полів-констант чи полів-посилань, оскільки їм не можуть бути присвоєні значення. Для цього передбачено спеціальну властивість конструктора, називану списком ініціалізації, який дозволяє ініціалізувати одну чи більше змінних і не надавати їм значення. Список розташовується наступним чином:

```
someClass(): m1(0), m2(10), m3(15) {}
```

При ініціалізації полів-об'єктів поля можна ініціалізувати за допомогою списку ініціалізаторів, у якому значення можуть бути виразами, наприклад:

```
someClass(int mm1,int mm2,int mm3):m1(mm1),m2(mm2),m3(mm3){}
```

В першому випадку конструктор викликається при створюванні об'єкта, наприклад, при визначенні:

someClass obj1, obj2;

У другому випадку при створюванні об'єкта obj буде викликано конструктор з відповідними, зазначеними у дужках параметрами, наприклад:

someClass obj(5,20,13);

Конструктор за замовчуванням

Конструктор може мати параметри за замовчуванням, які передаються автоматично, наприклад полям класу, при створенні об'єктів класу. Якщо, при створені об'єкту класу значення параметрів конструктора не вказуються, об'єкт ініціалізується значеннями параметрів за замовчуванням.

```
someClass (int mm1=1, int mm2=1, int mm3=1)
{ m1 = mm1; m2 = mm2; m3 = mm3;}
someClass obj;
```

Такий конструктор називається **конструктор за замовчуванням.**

Конструктор без параметрів також називають конструктором за замовчуванням. Якщо конструктор для будь-якого класу не визначено, то компілятор сам генерує конструктор за замовчуванням. Такі конструктори не присвоюють початкові значення полям класу, який визначає об'єкт класу без передавання параметрів, наприклад:

someClass obj1, obj2;

Кожен клас може включати лише один конструктор за замовчуванням.

Якщо у класі визначено будь-який конструктор, то компілятор не створить конструктора за замовчуванням.

Делегуючий конструктор

У пізніших версіях компілятору C++ з'явилась можливість створення **делегуючих конструкторів** класу.

Коли конструктор при створенні об'єкта класу викликає інший конструктор. Наприклад,

```
Person(string n)
{
  Name = n;
}
Person(int a): Person(string n)
{
```

```
  Age=a;
}
```

Таким чином здійснюється ієрархія виклику конструкторів. Ми делегуємо деяку операцію присвоювання імені першому конструктору. Це дуже часто застосовується при створенні класів.

```
Person obj("John"); //буде викликано перший конструктор Person(string n);
Person obj("John",40); //буде викликано спочатку перший конструктор, потім другий Person(int a);
```

Конструктор копіювання

Існує ще один спосіб ініціалізації об'єкта, який використовує значення полів вже існуючого об'єкта. Такий конструктор не треба самим створювати, він надається компілятором для кожного створюваного класу і називається конструктором копіювання за замовчуванням. Він має єдиний аргумент,який є об'єктом того ж самого класу.

Наприклад, створимо три об'єкти t1, t2, t3 у різні способи.

```
void main ()
{ Interval t1(2, 30);
Interval t2(t1);
Interval t3 = t1;
cout << "\nt1 = "; t1.showinterval();
cout << "\nt2 = "; t2.showinterval();
cout << "\nt3 = "; t3.showinterval();
cout << endl;
}
```

Результат роботи програми має вигляд:

```
t1 = 2 год. 30 хв.
t2 = 2 год. 30 хв.
t3 = 2 год. 30 хв.
```

Якщо ж клас містить покажчики чи посилання, виклик конструктора копіювання призведе до того, що й копія, й оригінал вказуватимуть на одну й ту саму ділянку пам'яті. В такому разі конструктор копіювання має бути створений програмістом і мати вигляд

```
T :: T(const T&)
```

де T – ім'я класу.

Наприклад:

```
someClass :: someClass (const someClass &D) { . . . }
```

Деструктор

Деструктор – це особлива форма методу, який застосовується для звільнення пам'яті, зайнятої об'єктом. Деструктор за суттю є антиподом конструктора. Він викликається автоматично, коли об'єкт виходить з області видимості.

Ім'я деструктора розпочинається з тільди (~), безпосередньо за якою йде ім'я класу. Деструктор має такий формат:

```
~ <ім'я класу> () {}
```

Деструктор не має аргументів і значення, яке повертається. Але він може виконувати деякі дії, наприклад, виведення остаточних значень елементів даних класу, що буває зручно при налагодженні програми.

Якщо деструктор явно не визначено, компілятор автоматично створює порожній деструктор.

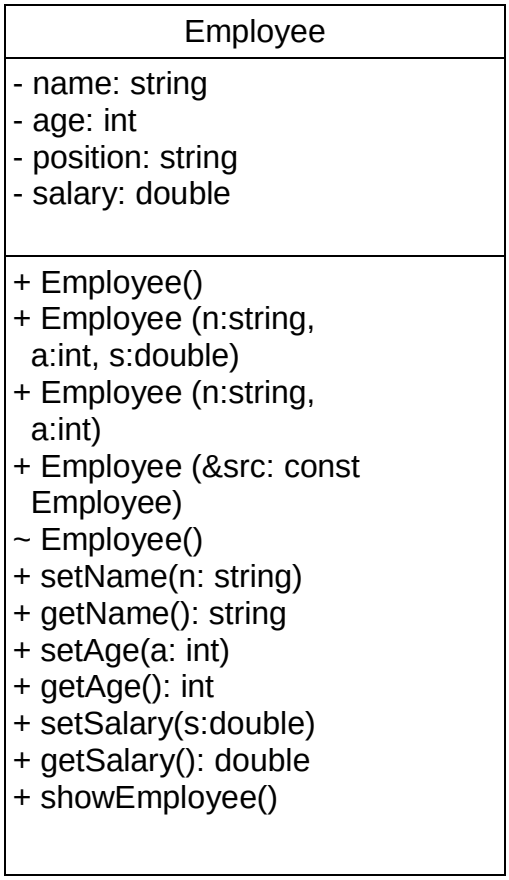
Розміщувати деструктор у класі явно треба у разі, якщо об'єкт містить покажчики на пам'ять, яка виділяється динамічно, – інакше при знищенні об'єкта пам'ять, на яку посилались його поля-покажчики, не буде позначено як звільнену. Покажчик на деструктор визначити не можна. Деструктор не успадковується.

Приклад: Розглянемо застосування конструкторів та деструктора на прикладі класу "Робітник компанії"(**див.лаб.раб.3**). Додамо в методи класу **Employee** :

- конструктор за замовчуваням;
- конструктор з параметрами, у тому числі з параметрами за замовчуванням;
- конструктор зі списком ініціалізації;
- конструктор копіювання;
- деструктор;

Ці методи знаходяться в розділі *public*.

Наведемо представлення розроблюваного класу в UML:



Наведемо приклад коду об'явлення та реалізації класу "Робітник":

- заголовковий файл «Employee.h»

```
#ifndef EMPLOYEE_H
#define EMPLOYEE_H
#include <iostream>
#include <string>
```

```
using namespace std;
class Employee
{
    string name;
    int age;
    string position;
    double salary;

public:
    Employee();//конструктором за замовчуванням
    Employee (string n, int a, double s);//конструктором з
//параметрами
    Employee (string n, int a=18);//конструктором з параметрами, у
//тому числі з параметрами за замовчуванням
    Employee (const Employee &src);//Конструктор копіювання
    ~Employee();// Деструктор
    void setName(string n);
    string getName();
    void setAge(int s);
    int getAge();
    void setSalary(double s);
    double getSalary();
    void showEmployee();
    };
    #endif // EMPLOYEE_H
```

файл з реалізацією класу «Employee.cpp»

```
#include <iostream>
#include "Employee.h"
#include <string>

 using namespace std;

//конструктором за замовчуванням
  Employee::Employee():name("N/A"),age(0),salary(0.0)
    {
      cout<<"Екземпляр класу створено конструкт за замовчув\n";
    }
//конструктором з параметрами
  Employee::Employee (string n, int a, double s)
    {
      name=n; age=a;
      if (s< 50000.0)
      salary = s;
      else // Недопустимий оклад
      salary = 0.0;
      cout<<"Екземпляр класу створено конструктором з
             параметрами\n";
    }
//конструктором з параметрами, зі списком ініціалізації
  Employee::Employee (string n, int a):name(n),age(a)
    {
```

```
        do  {
        cout << "Введите оклад для " << name << "меньше $50000: ";
        cin >> salary;
    }
        while (salary >= 50000.0);
        cout<<"Екземпляр класу створено конструктором з парам\n";
    }
  Employee::Employee (const Employee &src)//Конструктор копіювання
    {
       cout<<"Конструктор копіювання\n";
        name=src.name;
        age=src.age;
        salary=src.salary;
    }
  Employee::~Employee()// Деструктор
    {
        cout<<"Працює деструктор\n";
    }
  void Employee::setName(string n)
    {
     name=n;
    }
  string Employee::getName()
    {
     return name;
    }
  void Employee::setAge(int s)
    {
     age=s;
    }
  int Employee::getAge()
    {
     return age;
    }
  void Employee::setSalary(double s)
    {
     salary = s;
    }
  double Employee::getSalary()
    {
     return salary;
    }
  void Employee::showEmployee()
    {
     cout<<"Employee: "<<name<<"\t"<<"Age: "<<age<<"\t"<<"Salary:"
<<salary<<endl;
    }
```

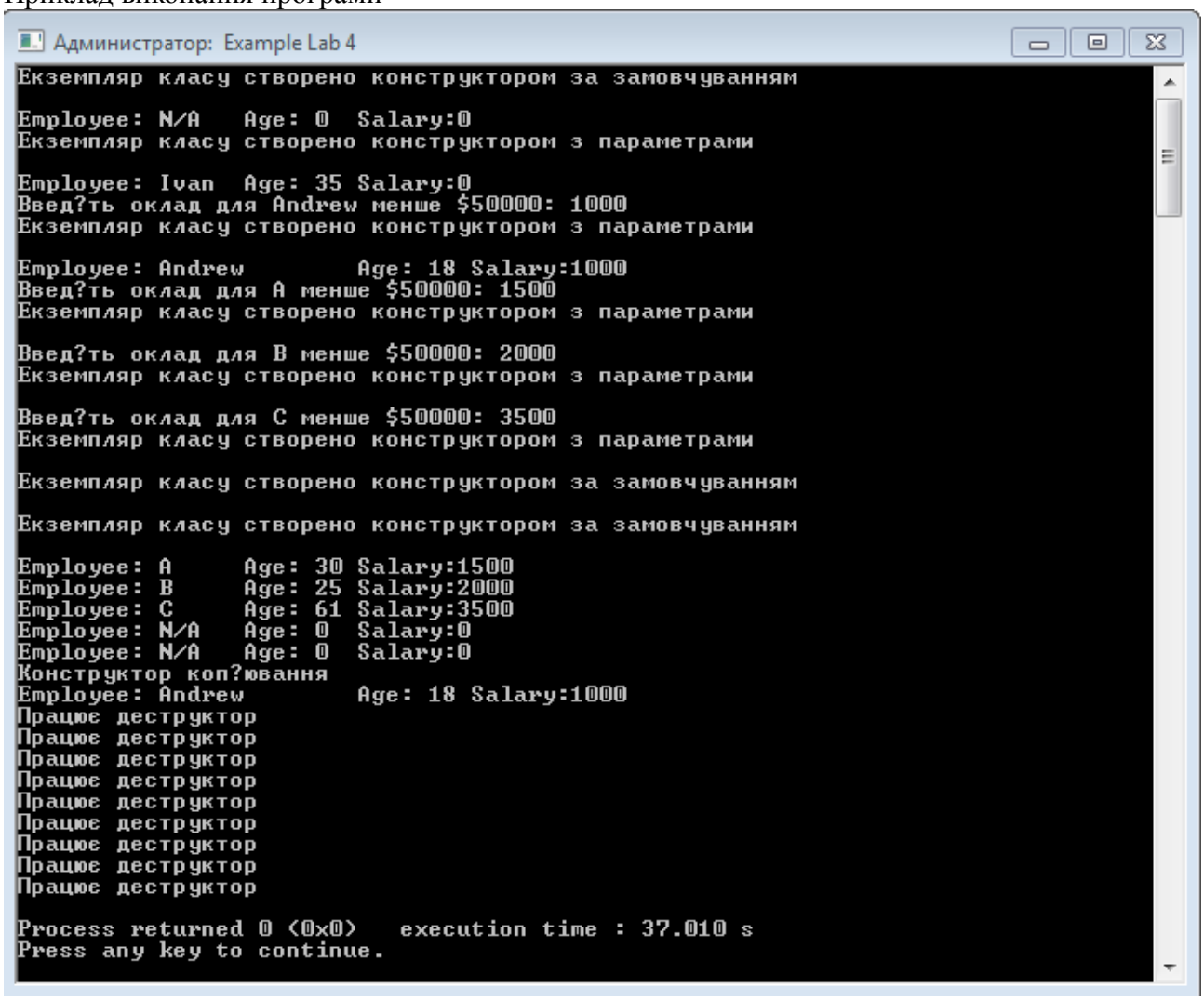
та файл «main.cpp» з прикладом створення і використання класу

```
#include <iostream>
#include "Employee.h"
```

```
using namespace std;

int main()
{
    setlocale(LC_ALL,"Russian");
    Employee emp1;
    emp1.showEmployee();
    Employee emp2("Ivan", 35,100000.0);
    emp2.showEmployee();
    Employee emp3("Andrew",18);
    emp3.showEmployee();
    Employee department[5]={Employee("A",30), Employee("B",25),
    Employee("C",61)};
     for (int i=0;i<5;i++)
     {
         department[i].showEmployee();
     }
     Employee emp4=emp3;
     emp4.showEmployee();
    return 0;
}
```

Приклад виконання програми



Лабораторна робота 4.

Хід виконання завдання:

1. Згідно свого варіанту завдання змоделювати клас для заданого поняття:
   - визначити необхідні атрибути (дані) класу
   - визначити необхідні методи для коректної роботи класу,
   - визначити режими доступу до елементів класу, що розглядається.

   До методів класу додати 3 конструктора на вибір і деструктор:
   - конструктор за замовчуванням;
   - конструктори з параметрами (у тому числі за замовчуванням);
   - зі списком ініціалізації;
   - конструктор копіювання;
   - делегуючий конструктор.

   Кожен з конструкторів також повинен друкувати повідомлення який це конструктор.
2. Зобразити UML-діаграму класу.
3. Реалізувати клас в вигляді програмного коду.
4. Написати програму, в якій користувач матиме можливість проводити маніпуляції зі створеним класом.
5. Організувати роботу з масивом екземплярів класу.

Варіанти завдань.

1. Трапеція.
2. Пловець
3. Багатокутник
4. Вектор в просторі.
5. Прямокутник.
6. Літак.
7. Автомобіль
8. Конус.
9. Пряма.
10. Ромб.
11. Книга.
12. Товар.
13. Студент.
14. Маршрут.
15. Страва
16. Матриця.
17. Трикутник.
18. Час.
19. Шахова фігура
20. Циліндр.
21. Лікар
22. Сфера.
23. Поїзд
24. Місто
25. Вектор на площині
26. Відрізок
27. Магазин
28. Таксі

Запитання для самоконтролю:

1) Що називається класом? Що таке атрибути класу
2) Як співвідносяться поміж собою поняття об’єкта та класу?
3) Які режими доступу до елементів класу Вам відомі. Для чого вони застосовуються?
4) Що таке конструктор та коли він викликається? Чи повертає він значення?
5) Які типи конструкторів Ви знаєте? Чи може клас мати декілька конструкторів?
6) Що таке деструктор і коли він викликається? Чи може в класі бути декілька деструкторів?
7) Чи можна передати початкові значення полям класу за допомогою конструктора?