# Тема лекції:

# ОРГАНІЗАЦІЯ КЛАСІВ І ОСОБЛИВОСТІ РОБОТИ З ОБ'ЄКТАМИ

# ЗМІСТ

1. **Функції – «друзі»**
2. Перевантаження конструкторів
3. Застосування динамічної ініціалізації до конструкторів
4. **Конструктори, деструктори й передача об'єктів**
5. Повернення об'єктів функціями
6. Створення й використання конструктора копії
7. **Ключове слово this**

# ЛІТЕРАТУРА:


1. Бублик В.В. Об’єктно-орієнтоване програмування: [Підручник] / В.В. Бублик. – К.: ІТ-книга, 2015. – 624 с.
2. Вступ до програмування мовою С++. Організація обчислень : навч. посіб. / Ю. А. Бєлов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставровський. – К. : Видавничо-поліграфічний центр "Київський університет", 2012. – 175 с.
3. Зубенко В.В., Омельчук Л.Л. Програмування. Поглиблений курс. – К.:Видавничо-поліграфічний центр “Київський університет”, 2011. - 623 с.
4. Страуструп Бьярне. Программирование: принципы и практика с использованием С++, 2-е изд. : Пер. с англ. - М. : ООО "И.Д. Вильяме", 2016. - 1328 с.
5. Прата С. Язык программирования С++. Лекции и упражнения. Учебник. -СПб. ООО «ДиаСофтЮП», 2003. 1104 с.
6. Шилдт Г. С++: базовый курс, 3-е издание. : Пер. с англ. – М. : Издательский дом «Вильямс», 2010. – 624 с.
7. Stroustrup, Bjarne. The C++ programming language. — Fourth edition. – Addison-Wesley, 2013. – 1361 pp.

# Функції-"друзі "

У C++ існує можливість дозволити доступ до закритих членів класу функціям, які не є членами цього класу.

Для цього досить оголосити ці функції "дружніми" (або "друзями") класу.

Щоб зробити функцію “другом" класу, треба включити її прототип в public-розділ оголошення класу з ключовим словом friend попереду.

# Функції-"друзі"

Наприклад, функція frnd() оголошується "другом" класу cl.

```
class cl {
   // . . .
  public:
   friend void frnd(cl ob);
   // . . .
};
```

**Ключове слово friend надає функції, що не є членом класу, доступ до його закритих членів.**
Ключове слово friend випереджає іншу частину прототипу функції. **Функція може бути "другом" декількох класів.**

```
// Демонстрація використання функції-"друга".
#include <iostream>
using namespace std;
class myclass {
    int a, b;
  public:
    myclass(int i, int j) { a=i; b=j; }
    friend int sum(myclass x); //Функція sum() "друг" класу myclass
};

/* Функція sum() не є членом жодного класу,  але оскільки клас
myclass “назвав її своїм другом”, вона має право на прямий доступ
до його членів даних a й b. */

int sum(myclass x) {
   return x.a + x.b;
}

int main() {
  myclass n(3, 4);
  cout << sum(n);
  return 0;
}
```

## Корисність функцій- «друзі»

**По-перше**, функції-"друзі" можуть бути корисні для перевантаження операторів певних типів.

**По-друге**, функції-"друзі" спрощують створення деяких функцій вводу-виводу.

**Третя причина** використання функцій-"друзів": у деяких випадках два (або більше) класи можуть містити члени, які знаходяться у взаємному зв'язку з іншими частинами програми.

## Корисність функцій- «друзів»

Наприклад, є два різних класи, які при виникненні певних подій відображають на екрані "спливаючі" повідомлення. Інші частини програми, які призначені для виводу даних на екран, повинні знати, чи є "спливаюче" повідомлення активним, щоб випадково не перезаписати його.

**За допомогою функції, "дружньої" для обох класів, можна прямо перевіряти статус кожного об'єкта, викликаючи тільки одну функцію, що буде мати доступ до обох класів**.

У подібних ситуаціях функція-"друг" дозволяє написати більш ефективний код.

```
#include <iostream>
```
// **Використання функції-"друга”**

```
using namespace std;
const int IDLE=0; const int INUSE=1; class C2; //
випереджальне оголош.
class C1 {
    int status; // IDLE - повідомлення неактивне,
INUSE - на екрані
  public:  void set_status(int state) { status =
state; };
    friend int idle(C1 a, C2 b);
};
class C2 {
   int status; // IDLE - повідомлення неактивне,
INUSE - на екрані
  public:  void set_status(int state) { status =
state; };
    friend int idle(C1 a, C2 b);
```

```
int idle(C1 a, C2 b) {
// Функція idle() - "друг" для класів C1 і C2.
  if(a.status || b.status) return 0;   else
return 1;
}
int main() {
  C1 x;  C2 y;
  x.set_status(IDLE);
  y.set_status(IDLE);
  if(idle(x, y)) cout << "Екран вільний.\n";
  else cout << "Відображається повідомлення.\n";
  x.set_status(INUSE);
  if(idle(x, y)) cout << "Екран вільний.\n";
  else cout << "Відображається повідомлення.\n";
  return 0;
}
```

```
// Використання функції-"друга".
#include <iostream>
using namespace std;
const int IDLE=0;
const int INUSE=1;
class C2; // випереджальне оголошення
class C1 {
  int status;
// IDLE, якщо неактивне, INUSE, якщо виведене на екран.
 public:
  void set_status(int state) { status = state; };
  int idle(C2 b); // тепер це член класу C1
};
class C2 {
  int status; // IDLE, якщо неактивно, INUSE, якщо виведене на екран.
 public:
  void set_status(int state) { status = state; };
  friend int C1::idle(C2 b); //функ.-"друг"
};
```

```
int C1::idle(C2 b) {
// Функція idle()-член класу C1 і "друг" класу C2
  if(status || b.status) return 0;  else return 1;
}
int main() {
  C1 x;  C2 y; x.set_status(IDLE); y.set_status(IDLE);
  if(x.idle(y)) cout << "Екран вільний.\n";
  else cout << "Відображається повідомлення.\n";
x.set_status(INUSE);
  if(x.idle(y)) cout << "Екран вільний.\n";
  else cout << "Відображається повідомлення.\n"; return 0;
}
```

```
#include <iostream>      // Перевантаження конструкторів
#include <cstdlib>
#include <ctime>
using namespace std;
class timer {
  int seconds;
 public:
  timer(int t) {seconds=t;}
// секунди у вигляді цілого числа
  timer(char *t) {seconds=atoi(t);}
// секунди у вигляді рядка
  timer(int min,int sec) {seconds=min*60+sec;}
// час у хвил. і секунд.
  void run();
};
```

```
void timer::run(){
  clock_t t1;
  t1 = clock();
  while( (clock() / CLOCKS_PER_SEC - t1 /
CLOCKS_PER_SEC )  <  seconds );
  cout << "\a"; // звуковий сигнал
}
int  main() {
  timer a (10), b("20"), c(1, 10);
  a.run(); // відлік 10 секунд
  b.run(); // відлік 20 секунд
  c.run(); // відлік 1 хвилини й 10 секунд
  return 0;
}
```

# Динамічна ініціалізація

**В C++ як локальні, так і глобальні змінні можна ініціалізувати під час виконання програми, використовуючи будь-який C++-вираз, дійсний на момент оголошення цієї змінної.**

Це означає, що змінну можна ініціалізувати не тільки значенням константи, а й значенням будь-якого виразу, операндами якого можуть бути інші змінні, виклики функцій тощо. Головне, щоб в момент виконання інструкції оголошення вираз ініціалізації мав значення (міг бути обчислений).

```
#define MAXLEN 80
char str[MAXLEN+1];
   // . . .
cin >> str;
```

## Застосування динамічної ініціалізації до конструкторів

Подібно простим змінним, об'єкти можна ініціалізувати динамічно при їхньому створенні.

Цей засіб дозволяє створювати об'єкт потрібного типу з використанням інформації, що стає відомою тільки під час виконання програми.

У наступній версії головної функції main() програми таймера для створення двох об'єктів b і c використовується динамічна ініціалізація.

# Застосування динамічної ініціалізації до конструкторів

```
int main()
{
  timer a(10);
  a.run();
  cout << "Уведіть кількість секунд: ";
  char str[80];
  cin >> str;
  timer b(str); // ініціалізація в динаміці
  b.run();
  cout << "Уведіть хвилини й секунди: ";
  int min, sec;
  cin >> min >> sec;
  timer c(min, sec); // ініціалізація в динаміці
  c.run();
  return 0;
}
```

# Присвоювання об'єктів

**Якщо два об'єкти однотипні (тобто обоє вони - об'єкти одного класу), то один об'єкт можна присвоїти іншому.**

```
#include <iostream>
using namespace std;
class myclass {
    int a, b;
  public:
    void setab(int i, int j)
      { a = i, b = j; }
    void showab();
};
void myclass::showab() {
  cout << “a=" << a << “ b=" << b << '\n';
}
int main() {
  myclass ob1, ob2; ob1.setab(10, 20); ob2.setab(0, 0);
  cout << "ob1 до присвоювання: ";  ob1.showab();
  cout << "ob2 до присвоювання: ";  ob2.showab(); cout << ‘\n’;
  ob2 = ob1; // Присвоюємо об'єкт ob1 об'єкту ob2.
  cout << "ob1 після присвоювання: ";  ob1.showab();
  cout << "ob2 після присвоювання: ";  ob2.showab();
  return 0;
}
```

```
ob1 до присвоювання: a=10 b=20
ob2 до присвоювання: a=0 b=0

ob1 після присвоювання: a=10 b=20
ob2 після присвоювання: a=10 b=20
```

# *Передача об'єктів функціям*

Об'єкт можна передати функції як аргумент з використанням звичайної C++-угоди про передачу параметрів за значенням. Тобто, функції передається не сам об'єкт, а його копія.

```
#include <iostream>
using namespace std;
class OBJ{
    int i;
  public:
    void set_i(int x) { i = x; }
    void out_i() { cout << i << "  "; }
};
void f(OBJ x) {
  x.out_i(); // Виводить число.
  x.set_i(100); // Установлює тільки локальну копію.
  x.out_i(); // Виводить число 100.
}
int main() {
  OBJ o;
  o.set_i(10);
  f(o);      // Виводить 10 100
  o.out_i(); // Виводить 10, значення змінної i не змінилося.
  return 0;
}
```

```
10  100  10
```

# Конструктори, деструктори й передача об'єктів

```
#include <iostream>
using namespace std;
class myclass {
    int val;
  public:
    myclass(int i) { val = i; cout << "Створення\n"; }
    ~myclass() { cout << "Руйнування\n"; }
    int getval() { return val; }
};
void display(myclass ob) {
  cout << ob.getval() << '\n';
}
int main() {
  myclass a(10);
  display(a);
  return 0;
}
```

При виконанні ця програма виводить такі несподівані результати.

```
Створення
10
Руйнування
Руйнування
```

## Конструктори, деструктори й передача об'єктів

При передачі об'єкта функції створюється його копія (і ця копія стає параметром у функції).
Створення копії означає "народження" нового об'єкта.
Коли виконання функції завершується, копія аргументу (тобто параметр) руйнується.
Коли при виклику функції створюється копія аргументу, звичайний конструктор не викликається - викликається **конструктор копії об'єкта**.
Конструктор копії визначає, як повинна бути створена саме копія об'єкта, а не новий об’єкт.
Якщо в класі явно не визначений конструктор копії, C++ надає його за замовчуванням.
**Конструктор копії за замовчуванням створює побітову (тобто ідентичну) копію об'єкта на момент виклику функції**

## Конструктори, деструктори й передача об'єктів

Коли функція завершується й руйнується копія об'єкта-аргумента, викликається деструктор цього об'єкта.
Необхідність виклику деструктора пов'язана з виходом об'єкта з області видимості.
Саме тому програма мала два звертання до деструктора.
Перше відбулося при виході з області видимості параметра **ob** функції `display()`, а друге - при руйнуванні об'єкта **a** у функції `main()` по завершенні програми.

**Підсумок:**

**Коли об'єкт передається функції як аргумент, звичайний конструктор не викликається.**
**Замість нього викликається конструктор копії, що за замовчуванням створює побітову (ідентичну) копію цього об'єкта-аргумента.**
**Але коли ця копія руйнується (звичайно при виході за межі області видимості по завершенні функції), обов'язково викликається деструктор.**

```
#include <iostream> //Потенційні проблеми при передачі параметрів
#include <cstdlib>
using namespace std;
class myclass {
    int *p;
  public:
    myclass(int i);   ~myclass();
    int getval() { return *p; }
};
myclass::myclass(int i){
  cout << "Виділення пам'яті, що адресується покажчиком p.\n";
  p = new int;   *p = i;
}
myclass::~myclass() {
  cout <<"Звільнення пам'яті, що адресується покажчиком p.\n";
  delete p;
}

void display(myclass ob) {  cout << ob.getval() << '\n‘;  }

int main() {
  myclass a(10);
  display(a);
  return 0;
}
```

```
Виділення пам'яті, що адресується покажчиком p.
10
Звільнення пам'яті, що адресується покажчиком p.
```

```
// Одне з рішень проблеми передачі об'єктів.
#include <iostream>
#include <cstdlib>
using namespace std;
class myclass {
    int *p;
  public:
    myclass(int i);    ~myclass();
    int getval() { return *p; }
};
myclass::myclass(int i) {
  cout << "Виділення пам'яті, що адресується покажчиком p.\n";
  p = new int;    *p = i;
}
myclass::~myclass() {
  cout <<"Звільнення пам'яті, що адресується покажчиком p.\n";
  delete p;
}
void display(myclass &ob) {   // Ця функція НЕ створює проблем.
  cout << ob.getval() << '\n';
}
int main() {
  myclass a(10);  display(a);
  return 0;
}
```

```
Виділення пам'яті, що адресується покажчиком p.
10
Звільнення пам'яті, що адресується покажчиком p.
```

# Повернення об'єктів функціями

```
#include <iostream>
#include <cstring>
using namespace std;
class sample {
    char s[80];
  public:
    void set(char *str) { strcpy(s, str); }
    void show() { cout << s << "\n"; }
};
sample input() {      // Ця функція повертає  об'єкт типу sample.
  char instr[80];
  sample str;
  cout << "Уведіть рядок: “;    cin >> instr;
  str.set(instr);
  return str;
}
int main() {
  sample ob;
  // Об'єкту ob присвоюємо об'єкт, що повертається функцією
input().
  ob = input();
  ob.show();
  return 0;
}
```

# Потенційна проблема при поверненні об'єктів функціями

```
#include <iostream>
#include <cstring>
#include <cstdlib>
using namespace std;
class sample {
    char *s;
  public:
    sample() { s = 0; }
    ~sample() { if(s) delete [] s; cout << "Звільнення s-пам'яті.\n“; }
    void show() { cout << s << "\n"; }
    void set(char *str);
};
void sample::set(char *str) {s=new char[strlen(str)+1]; strcpy(s,str);}
sample input() { // Ця функція повертає  об'єкт типу sample.
  char instr[80];   sample str;
  cout << "Уведіть рядок: "; cin >> instr; str.set(instr);
  return str;
}
int main() {
  sample ob;
  ob = input(); // Ця інструкція генерує помилку!!!!
  ob.show(); // Відображення "сміття".
  return 0;
}
```

```
Уведіть рядок: Привіт
Звільнення s-пам'яті.
Звільнення s-пам'яті.
Тут сміття
Звільнення s-пам'яті.
```

## Створення й використання конструктора копії

Ініціалізація обєкта може виконуватися трьома способами:

- один об'єкт явно ініціалізує інший об'єкт, як, наприклад, в оголошенні;
- копія об'єкта передається параметру функції;
- генерується тимчасовий об'єкт (найчастіше як значення, що повертається функцією).

У основі розглянутих проблем лежить створення **побітової** копії об'єкта.

Конструктор копії дозволяє управляти діями, що складають процес створення копії об'єкта.

**Конструктор копії застосовується тільки до ініціалізацій.**

**Він не застосовується до присвоювань**.

# Створення й використання конструктора копії

```
ім'я_класу (const ім'я_класу &obj) {
  // тіло конструктора
}
```

**Тут елемент obj означає посилання на об'єкт, що використовується для ініціалізації іншого об'єкта.**
**Наприклад, є клас myclass і об'єкт b типу myclass, тоді при виконанні наступних інструкцій буде викликаний конструктор копії класу myclass.**

```
myclass x = b;
// Об'єкт b явно ініціалізує об'єкт x.
x.func1(b);    // Об'єкт b передається як аргумент.
b = func2();   // Об'єкт b приймає об'єкт, що повертається функцією.
```

**У перших двох випадках конструктору копії буде передане посилання на об'єкт b, а в третьому - посилання на об'єкт, що повертається функцією func2().**

# Конструктори копії й параметри функції

```
#include <iostream>
#include <cstdlib>
using namespace std;
class myclass {
    int *p;
  public:
        // звичайний конструктор
    myclass(int i);
        // конструктор копії
    myclass(const myclass &ob);
    ~myclass();
    int getval() { return *p; }
};
// Конструктор копії.
myclass::myclass(const myclass &obj){
  p = new int;
  *p = *obj.p; // значення копії
  cout << “Конструктор копії.\n";
}
myclass::myclass(int i) {
  cout << "Виділення пам'яті.\n";
  p = new int;   *p = i;
}
```

```
myclass::~myclass() {
  cout <<"Звільнення пам'яті.\n";
  delete p;
}

/* Ця функція приймає
   один об'єкт-параметр. */
void display(myclass ob) {
  cout << ob.getval() << '\n';
}

int main()
{
  myclass a(10);
  display(a);
  return 0;
}
```

Ця програма генерує такі результати.

Виділення пам'яті.
Конструктор копії.
10
Звільнення пам'яті.
Звільнення пам'яті.

# Використання конструкторів копії при ініціалізації об'єктів

```
#include <iostream>
#include <cstdlib>
using namespace std;
class myclass {
    int *p;
  public:
    myclass(int i); // звичайний конструктор
    myclass(const myclass &ob); // конструктор копії
    ~myclass();
    int getval() { return *p; }
};
myclass::myclass(const myclass &ob) {
  p = new int;   *p = *ob.p; cout << “Конструктор копії.\n"; }
myclass::myclass(int i) {
  p = new int;  *p = i; cout << “Звичайний конструктор.\n“; }
myclass::~myclass() {
  cout << "Звільнення р-пам’яті.\n"; delete p; }

int main() {
  myclass a(10);
  myclass b = a; // Ініціалізація
  return 0;
```

```
Звичайний конструктор.
Конструктор копії.
Звільнення р-пам’яті.
Звільнення р-пам’яті.
```

## Використання конструктора копії при поверненні функцією об'єкта

```
#include <iostream>
using namespace std;
class myclass {
  public:
    myclass() { cout << "Звичайний конструктор.\n"; }
    myclass(const myclass &obj) {cout << "Конструктор копії.\n"; }
};
myclass f()
{
  myclass ob; // Викликається звичайний конструктор
  return ob; // Викликається конструктор копії для тимчасового об’єкта
}
int main()
{
  myclass a;  // Викликається звичайний конструктор
  a = f();    // Присвоєння тимчасового об’єкта
  return 0;
}
```

Ця програма генерує такі результати.

```
Звичайний конструктор.
Звичайний конструктор.
Конструктор копії.
```

# Ключове слово **this**

***Ключове слово this - це покажчик на об'єкт, для якого викликається функція-член.***

```
class cl {
  int i;
  void f() { ... };
  // . . .
};
```

У функції `f()` припустима інструкція `i = 10;` Це скорочена форма `this->i = 10;`

```
#include <iostream>
using namespace std;
class cl {
    int i;
  public:
    void load_i(int val) { this->i = val; } // те ж саме, що i = val
    int get_i() { return this->i; } // те ж саме, що return i
};
int main() {
  cl o;
  o.load_i (100);
  cout << o.get_i();
  return 0;
}
```