Лабораторна робота №4. Екземпляри класів.

Мета роботи – вивчення об’єктів класів, колекцій об’єктів, виділення під них пам’яті.

Теоретичний матеріал лекції, [1, розділ 3], [2, розділ 12, 13], [3, розділи 10, 11, 12].

1. Короткі теоретичні відомості

1.1. Оголошення об’єктів

Після визначення класу можна створювати його екземпляри (об’єкти), посилання та вказівники з типом класу:

- у сегменті або стеку:

```
Тип_класу ідентифікатор_об’єкту(аргументи конструктора);
Тип_класу ідентифікатор_об’єкту = Тип класу(аргументи конструктора);
```

- динамічній пам’яті:

```
Тип_класу * вказівник = new Тип_класу(аргументи конструктора)
Тип_класу & вказівник = *new Тип_класу(аргументи конструктора)
```

Звільнення динамічної пам’яті:

```
delete вказівник
delete & посилання
```

1.2. Види об’єктів

Залежно від можливості зміни полів даних класу розрізняють:

- звичайні змінні об’єкти;
- константні об’єкти (оголошуються зі словом `const`);
- `volatile`-об’єкти.

Звичайні об’єкти можуть викликати будь-які методи класу. Константні об’єкти можуть викликати тільки константні методи класу. Дані константного об’єкта не можна змінити у програмі. Якщо деяке поле константного об’єкта необхідно змінити, то воно оголошується як `mutable`.

Об’єкти з оголошенням `volatile` можуть бути змінені деяким стороннім процесом (апаратурою, операційною системою, паралельними потоками коду).

Залежно від місця оголошення розрізняють:

- глобальні об’єкти;
- локальні об’єкти.

Глобальні об’єкти оголошуються між функціями, а локальні – всередині функцій.

Об’єкти можуть мати наступні класи пам’яті:

- автоматичний (`auto` – за замовчуванням);
- статичний (`static`);
- зовнішній (`extern`).

Локальні об’єкти можуть мати усі класи пам’яті. Глобальні об’єкти не можуть бути автоматичними.

Автоматичні об’єкти існують тільки всередині блоку, в якому вони визначені. Статичні об’єкти існують від моменту їх створення до завершення роботи програми.

Об’єкт можна оголосити статичним або (та) константним. Якщо глобальний об’єкт оголошений як статичний, то його використання у програмі обмежується тільки тим файлом, у якому оголошено статичний об’єкт.

Локальний статичний об’єкт зберігає проміжні значення своїх полів між повторними викликами функції, у якому оголошено об’єкт.

Тимчасові об’єкти створюються у виразах та для виклику методів класу без попереднього створення об’єктів класу. Для створення тимчасових об’єктів вказується назва конструктора та

його аргументи:

```
Class Base {
//…
     Base(int a=0, int b=0) { x=a; y=b; }
     int Get() { retuen x; }
};
Int x=Base().Get();
Int y=(new Base)->Get();
Int z=(*new Base).Get();
void main() {
      Cout << Base(1).Get() << endl;
      Cout << base(2,3).y << endl;
}
```

1.3. Колекції об'єктів. Об'єкти масивів і масиви об'єктів

Об'єкт класу може містити масиви даних. Такий об'єкт є контейнером для масиву. Наприклад, клас, закрита частина якого містить вказівник на двовимірний масив у динамічній пам'яті `int **p`. Двовимірний масив створюється за допомогою одновимірного масиву вказівників, за кожним елементом якого закріплюється відповідний рядок матриці:

```
p = new int* [row];
for(int i=0; i<row;i++)  p[i] = new int[col];
```

Масиви об'єктів можуть оголошуватися одним із таких способів:

```
Тип_класу ідентифікатор_масиву[кількість_об'єктів];
Тип_класу * вказівник =  new Тип_класу[кількість_об'єктів];
Тип_класу & посилання = *new Тип_класу[кількість_об'єктів];
```

При оголошенні масивів без їх початкової ініціалізації вимагається наявність конструктора без параметрів або конструктора, всі параметри якого набувають значення за замовчуванням.

```
class A{
   int x,y;
public:
   A(int x=0, int y=0{
   this->x=x;
   this->y=y;
}
// …
};
void main() {
     A b[]={A(1,2), A(3,4){;
//…
}
```

Звільнення динамічної пам'яті, виділеної для масиву об'єктів:

```
delete [] вказівник
delete []& посилання
```

1.4. Об'єкт список

Контейнерні об'єкти можуть містити в собі різноманітні набори елементів. Розглянемо організацію класу для роботи з однонаправленим списком. У закритій частині класу `list` оголошена структура `elem`, яка описує елемент списку, та вказівник на початок списку. Структура містить поле даних `int`, вказівник `next` - на сусідній елемент списку та конструктор ініціалізації полів структури.

У відкртій частині класу `list` розміщені конструктор, деструктор та допоміжні методи: `add()` - для добавлення елементу у список, `del()` - для вилучення елементу зі списку, `print()` - для виведення списку на екран.

```
#include <iostream>
using namespace std;
```

```
class list
{
private:
    struct elem
    {
      int info;
      elem *next;  // попередній елемент списку
      elem(int x, elem *p) {info=x; next=p;}
    };
    elem *head;      // вказівник на початок списку
    // конструктор списку з клавіатури
public:
    list()
    {
      int x;
      head=NULL;
      cout << "Введіть 5 цілих чисел" << endl;
      for(int i=0;i<5;i++) { cin>>x; add(x); }
    }
   // добавлення елемента
    void add(int x) {elem *p=new elem(x,head); head=p; }
  // вилучення початкового елемента списку
    void del() { elem *p=head->next; delete head; head=p; }
    void print()   // друк елемента
    {
      elem *p=head;
      while (p!=NULL)
      {
          cout << p->info << ' ';
          p=p->next;
      }
      cout << endl;
    }
};

int main()
{
    list s1;
    s1.print();
    s1.add(7);
    s1.print();
    s1.del();
    s1.print();
    return 0;
}
```

Введіть 5 цілих чисел
5 4 3 2 1
7 5 4 3 2 1
5 4 3 2 1

1.5. Список об’єктів

Замість списку, поміщеного у клас, можна створити список об’єктів. Тепер клас `list` містить у закритій частині поле даних `info` та вказівник на сусідній у списку об’єкт класу `list`. У відкритій частині знаходиться конструктор для ініціалізації елементів та методи доступу до полів закритої частини класу.

```
#include <iostream>
using namespace std;

class list {
    int info;
    list *next;
```

```
public:
    list(int x, list *p) { info=x; next=p; }
    int get_info()    { return info; }
    list * get_next() { return next; }
};

int main() {
    list *head = NULL, *p;
    int x;
    cout << " Введіть 5 цілих чисел" << endl;
    for(int i=0;i<5;i++) { cin>>x; p=new list(x,head); head=p; }
    cout << "Виведення списку" << endl;
    p=head;
    while( p!=NULL) { cout << p->get_info() << ' '; p=p->get_next(); }
    cout << endl;
     // Витирання списку
    while(head != NULL ) { p=head; head=head->get_next(); delete p; }

    return 0;
}
```

Введіть 5 цілих чисел
5 4 3 2 1
Виведення списку
5 4 3 2 1

1.6. Розміщення та оголошення класів

Всередині класу можна оголошувати дані, що мають тип вказівника на цей самий клас, але не можна оголосити посилання або об’єкт цього самого класу (табл. 1). Методи класу, їх параметри та локальні оголошення можуть мати посилання на цей же клас або тип цього самого класу.

Таблиця 1

Використання імен вищерозміщених класів

Клас		**Можливі оголошення у класі**
A	Дані	A*
	Параметри, типи методів, локальні дані	A&, A*, A
B	Дані	A&, A*, A, B*
	Параметри, типи методів, локальні дані	A&, A*, A, B&, B*, B

Нижче оголошення класу можна оголосити його об’єкт, посилання або вказівник. Такі оголошення можна виконати всередині іншого зовнішнього класу, всередині зовнішньої функції або на глобальному рівні.

Щоб мати можливість оголосити посилання або вказівник на нижче розміщений клас, необхідно виконати випереджувальне оголошення класу (табл. 2).

Таблиця 2

Використання імен нищерозміщених класів

Клас B;		**Можливі оголошення у класі**
A	Дані	A*, B&, B*
	Параметри, типи методів, локальні дані	A&, A*, A, B&, B*
B	Дані	A&, A*, A, B*
	Параметри, типи методів, локальні дані	A&, A*, A, B&, B*, B

Випереджуюче оголошення не дає можливості оголосити об’єкт нижче розміщеного класу у вище розміщеному класі (або на глобальному рівні). Метод вище розміщеного класу не може отримувати або повертати об’єкт нижче розміщеного класу “за значенням”.

У вищерозміщеному класі дозволяється тільки цілісне використання вказівників та посилань на нижчерозміщений клас. За допомогою таких вказівників або посилань не можна отримати доступ до елементів (даних або методів) нижчерозміщеного класу.

У програмі потрібно уникати звернень до нижчерозміщених класів.

Запитання.

1. Як оголошуються об’єкти класів.
2. Які є види об’єктів.
3. Як створити тимчасові об’єкти.
4. Як генеруються і обробляються винятки при відсутності місця під динамічну пам’ять.
5. Як створити двовимірний масив у динамічній пам’яті конструктором класу.
6. Як створити масив об’єктів класу в статичній, стековій і динамічній пам’яті.
7. Як створити масив об’єктів у динамічній пам’яті і їх ініціалізувати.
8. Як створити об’єкт список.
9. Як створити список об’єктів.
10. Як використовуються всередині класу імена вище розміщених класів.
11. Як використовуються всередині класу імена вище нищерозміщених класів.

Завдання.

1. Написати програму, в якій оператор `new` використовується для динамічного виділення пам’яті під змінні типу `float[2][2], long[2][2], char[20]`. Присвоїти масивам значення і вивести на екран. Конструктором звільнити динамічно виділену область пам’яті.

2. Написати функцію `neg()`, яка міняє знак свого цілого параметра. Функцію написати двома способами: з використанням параметра вказівника і з використанням параметра посилання.

3. Використовуючи наступне оголошення класу, створити масив з 10 елементів і ініціалізувати змінну `ch` значеннями від A до J, вивести значення на екран

```
class letters {
    char ch;
public:
    letters(char c) { ch=c; }
    char get_ch() { t=return ch; }
};
```

4. Виконати завдання згідно номеру студента в журналі групи:

4.1. Створити клас Вектор цілих чисел. Розробити клас СТЕК, що містить об'єкт класу ВЕКТОР. Визначити необхідні конструктори, деструктори, методи занесення елемента у стек та читання зі стека. Вивести вміст стека на екран.

4.2. Створити клас СПИСОК цілих чисел. Розробити клас СТЕК, що містить об’єкт класу СПИСОК. Визначити необхідні конструктори, деструктори, методи занесення елемента у стек та читання зі стеку. Вивести вміст стека на екран.

4.3. Розробити клас КОМП’ЮТЕР, який містить об’єкти класів СИСТЕМНИЙ БЛОК, ЕКРАН, КЛАВІАТУРА, МИША. Визначити необхідні елементи даних, конструктори, деструктори, методи роботи з елементами даних. Створити масив об’єктів класу КОМП’ЮТЕР та визначити комп’ютер з мінімальним відхиленням параметрів від заданих.

4.4. Створити клас АВТОМОБІЛЬ, який містить об’єкти класу ДВИГУН. У класі ДВИГУН визначено дані про об’єм двигуна, потужність, заводський номер, Клас АВТОМОБІЛЬ додатково містить марку, колір, та номер державної реєстрації. Визначити конструктори без параметрів і з різним числом параметрів, деструктор, методи для роботи з полями даних. Визначити методи зміни номера і кольору.

4.5. Створити клас ДВИГУН із заданням потужності та об’єму двигуна. Створити клас ВАНТАЖІВКА, що містить об’єкт класу ДВИГУН. Додатково задається марка (вказівник на рядок), вантажопідйомність та номер державної реєстрації. Визначити конструктори,

деструктори, методи встановлення та читання елементів даних.

4.6. Створити клас ФАЙЛ, який містить дані про файл: назву, розмір, дату та час його створення, атрибути. Створити клас КАТАЛОГ, що містить масив об'єктів класу ФАЙЛ. Визначити необхідні конструктори, деструктори, методи роботи з файлами та каталогами. Дані про файли отримати командою `Linux ls`.

4.7. Розробити клас УНІВЕРСИТЕТ, який містить прізвище ректора та масив об'єктів ІНСТИТУТ. Об'єкт класу ІНСТИТУТ містить прізвище директора та масив об'єктів класу КАФЕДРА. Клас КАФЕДРА містить прізвище завідувача кафедри та кількість співробітників. Визначити необхідні конструктори, деструктори та методи роботи з елементами даних. Обчислити загальну кількість співробітників університету.

4.8. Клас ГРУПА містить масив об'єктів класу СТУДЕНТ. Клас СТУДЕНТ містить прізвище студента, номер залікової книжки та масив оцінок. Визначити необхідні конструктори, деструктори та методи роботи з елементами даних. Знайти середній бал кожного студента. Вивести на екран прізвище п'яти студентів з найвищим балом.

4.9. Клас ФАЙЛ містить масив об'єктів класу РЯДОК символів. Клас рядок символів містить вказівник на рядок символів в області динамічної пам'яті. Визначити необхідні конструктори, деструктори, методи роботи з файлом рядків символів - запису у файл, читання з файлу, поповнення, копіювання.

4.10. Клас ВІКНО містить об'єкти класів ПРЯМОКУТНИК, КОЛІР_ФОНУ, МЕНЮ, СМУГИ_ПРОКРУТКИ. Клас ПРЯМОКУТНИК містить координати головної діагоналі вікна, класс МЕНЮ - масив рядків з назвами команд меню, клас СМУГИ_ПРОКРУТКИ - поточні координати повзунків. Визначити необхідні конструктори, деструктори та методи роботи з елементами даних.

4.11. Клас ПРОГРАВАЧ містить масив об'єктів класу БІБЛІОТЕКА мультимедіа. Клас БІБЛІОТЕКА містить дані про відеофільми: заголовок, тему, акторів, оцінку, довжину, швидкість відтворення, назву файла, дату. Визначити необхідні конструктори, деструктори та методи роботи з елементами даних. Здійснити пошук фільмів за акторами, назвами, темами.

4.12. Клас БІБЛІОТЕКА містить масив об'єктів класу КНИГА. Клас КНИГА містить прізвище автора, назву, видавництво, рік видання, кількість сторінок. Визначити необхідні конструктори, деструктори та методи роботи з елементами даних. Здійснити пошук книг за прізвищем автора, назвою, видавництвом, роком видання.

4.13. Клас РОЗКЛАД руху поїздів містить масив об'єктів класу ІНФОРМАЦІЯ про час відправлення поїздів. У класі ІНФОРМАЦІЯ задано номер поїзда, напрям руху, час відправлення, номер платформи, час прибуття на кінцеву станцію. Визначити необхідні конструктори, деструктори та методи роботи з елементами даних. Ввести поточний час та визначити найближчий час відправлення поїзда за заданим напрямом.

4.14. Клас ЕЛЕКТРОННА таблиця містить масив об'єктів класу ВЕКТОР дійсних чисел. Розробити методи для опрацювання даних за встановленими формулами. Визначити необхідні конструктори, деструктори та методи доступу до елементів даних. Знайти суму елементів у кожному рядку, стовпчику, інші обчислення за формулами.

4.15. Клас МАТРИЦЯ містить масив об'єктів ВЕКТОР дійсних чисел. Визначити необхідні дані, конструктори, деструктори та методи роботи з елементами даних. Виконати додавання, віднімання матриць, множення матриці на вектор, множення матриці на матрицю.

4.16. Клас ВЕКТОР містить масив об'єктів РЯДОК символів. Визначити необхідні дані, конструктори, деструктори та методи роботи з елементами даних. Здійснити виділення лексем рядка, пошуку найдовшого і найкоротшого слова, найдовшого і найкоротшого рядка.

4.17. Клас ЧЕРГА містить масив об'єктів типу ОСОБА. Визначити необхідні дані, конструктори, деструктори та методи ставлення у чергу та вилучення з черги. Вивести поточний вміст черги на екран.

4.18. Клас МАГАЗИН містить масив об'єктів класу ТОВАРИ. Клас ТОВАРИ містить назву товару, свідоцтво якості, вартість. Клас ПОКУПЕЦЬ містить перелік потрібних товарів. Визначити необхідні дані, конструктори, деструктори та методи робот з елементами даних.

Здійснити купівлю та визначити вартість товарів згідно з переліком.

4.19. Клас домашній кінотеатр містить об’єкти класів ТЮНЕР, ТЕЛЕВІЗОР, ЗВУКОВІ_КОЛОНКИ, ПУЛЬТ_КЕРУВАННЯ. Визначити необхідні дані, конструктори, деструктори та методи роботи з елементами даних. Клас ТЮНЕР містить канали програм. Клас ПУЛЬТ керування містить методи перемикання каналів, регулювання гучності, включення/виключення. Використовуючи ПУЛЬТ керування здійснити перемикання і вибирати потрібний канал.

4.20. Клас ЦИФРОВИЙ ГОДИННИК містить об’єкти класів КОНТРОЛЕР_ЧАСУ і КОНТРОЛЕР_ДАТИ, ЕКРАН для відображення часу. Визначити необхідні дані, конструктори, деструктори та методи роботи з елементами даних. Встановити поточні дату і час. Встановити час будильника. Зімітувати спрацювання будильника.

4.21. Клас вентилятор містить класи ДВИГУН, КОНТРОЛЕР, ПУЛЬТ керування. Визначити необхідні дані, конструктори, деструктори та методи роботи з елементами даних. За допомогою пульта керування виставити необхідну швидкість обертів двигуна та час включення ВЕНТИЛЯТОРА.

4.22. Клас ЕКРАННИЙ_РЕДАКТОР тексту містить масив об’єктів класу РЯДОК символів та об’єкт класу КУРСОР. Клас КУРСОР містить координати екранного курсора. Клас РЯДОК символів містить список об’єктів СИМВОЛІВ. Визначити необхідні дані, конструктори, деструктори та методи роботи з елементами даних. Виконати моделювання набору та редагування тексту: включення символів у рядок, витирання символів з рядка. Відредагований текст зберегти у файл.

4.23. Клас АПТЕКА містить список об’єктів класу ЛІКИ. Клас ЛІКИ містить назву препарату, концентрацію, інструкцію до застосування, вартість. Клас РЕЦЕРТ містить перелік ліків. Визначити необхідні дані, конструктори, деструктори та методи роботи з елементами даних. Визначити наявність потрібних ліків в аптеці та вартість ліків за рецептом.

4.24. Клас ПРІОРИТЕТНА_ЧЕРГА містить масив об’єктів класу ОСОБА. Визначити необхідні конструктори, деструктори та методи ставлення у чергу і вилучення з черги. Вивести поточний вміст черги на екран.

4.25. Клас СТРІЛЬБА по мішені містить об’єкти класів МІШЕНЬ та СТРІЛЕЦЬ. Клас МІШЕНЬ задається набором очок влучності стрільбою У кожному акті стрільби МІШЕНЬ з’являється із заданою імовірністю. Влучність стрільця задається дискретним розподілом ймовірностей потрапляння у мішень. Визначити суму очок з десяти спроб стрільби по мішені.

4.26. Клас мобільний ТЕЛЕФОН містить об’єкти класів SIM-КАРТА, ТЕЛЕФОННА_КНИГА та РАХУНОК. Визначити необхідні дані, конструктори, деструктори та методи роботи з елементами даних. Виконати поповнення рахунку, редагування телефонної книги, визначення залишку на рахунку.

2. Завдання для самостійної роботи № 3

```
/* Розд. 13. 1. listing 1 — масив об'єктів */
#include <iostream>
using namespace std;

class cl {
  int i;
public:
  void set_i(int j) { i=j; }
  int get_i() { return i; }
};

int main() {
  cl ob[3];
  int i;
  for(i=0; i<3; i++) ob[i].set_i(i+1);
  for(i=0; i<3; i++)
    cout << ob[i].get_i() << " ";
  return 0;
}
```

1 2 3

```
/* 2. listing 2 клас з конструктором  */
#include <iostream>
using namespace std;

class cl {
  int i;
public:
  cl(int j) { i=j; }
  int get_i() { return i; }
};

int main() {
  cl ob[3] = {1, 2, 3}; // скорочена форма ініціалізації,
//  c1 ob[3]={ c1(1), c1(2), c1(3) };  повна форма ініціалізаціїї
  int i;
  for(i=0; i<3; i++)
    cout << ob[i].get_i() << " ";
  return 0;
}
```

1 2 3

```
/* 3. listing 4 — конструктор з декількома параметрами */
#include <iostream>
using namespace std;

class cl {
  int h;
  int i;
public:
  cl(int j, int k) { h=j; i=k; } // конструктор з 2-ма параметрами
  int get_i() {return i;}
  int get_h() {return h;}
};

int main() {
  cl ob[3] = {
    cl(1, 2), // іціалізація
    cl(3, 4),
    cl(5, 6)
```

```
  };
  int i;
  for(i=0; i<3; i++) {
    cout << ob[i].get_h();
    cout << ", ";
    cout << ob[i].get_i() << "\n";
  }
  return 0;
}
```
1, 2
3, 4
5, 6

```
/* 4. listing 8 — вказівники на об'єкти */
#include <iostream>
using namespace std;

class cl {
  int i;
public:
  cl(int j) { i=j; }
  int get_i() { return i; }
};

int main(){
  cl ob(88), *p;
  p = &ob; // тримати адрес ob
  cout << p->get_i(); // для виклику функції get_i() застосовується
                      // вказівник ->
  return 0;
}
```
88

```
/* 5. listing 9 — доступ до елементів масиву через вказівник*/
#include <iostream>
using namespace std;

class cl {
  int i;
public:
  cl() { i=0; }
  cl(int j) { i=j; }
  int get_i() { return i; }
};

int main() {
  cl ob[3] = {1, 2, 3};
  cl *p;
  int i;

  p = ob; // адрес першого елемента масиву p=&ob[0]
  for(i=0; i<3; i++) {
    cout << p->get_i() << " ";
    p++; // вказівник на наступний елемент масиву
  }
  return 0;
}
```
1 2 3

```
/* 6. listing 10 — доступ до члена об'єкта */
#include <iostream>
using namespace std;

class cl {
```

```
public:
  int i;
  cl(int j) { i=j; }
};

int main() {
  cl ob(1);
  int *p;
  p = &ob.i; // отримання адресу члена ob.i
  cout << *p; // звернення до члена  ob.i через вказівник p
  return 0;
}
```
1

/* 7. listing 13 — вказівник this */
```
#include <iostream>
using namespace std;

class pwr {
  double b;
  int e;
  double val;
public:
  pwr(double base, int exp);
  double get_pwr() { return val; }   // return this->val;
};

pwr::pwr(double base, int exp) {  // функції члену неявно передається
                                  // вказівник this
  b = base;   // this->b=base;
  e = exp;    // this->e=exp;
  val = 1;    // this->val=1;
  if(exp==0) return;
  for( ; exp>0; exp--) val = val * b;    // this->val = this->val * this->b;
}

int main() {
  pwr x(4.0, 2), y(2.5, 1), z(5.7, 0);
  cout << x.get_pwr() << " ";
  cout << y.get_pwr() << " ";
  cout << z.get_pwr() << "\n";
  return 0;
}
```
16 2.5 1

/* 8. listing 19 — доступ до об'єктів похідного класу через вказівник на базовий клас.
Вказівник базового класу *B може посилатися на об'єкти похідного класу */
```
#include <iostream>
using namespace std;

class base {
  int i;
public:
  void set_i(int num) { i=num; }
  int get_i() { return i; }
};

class derived: public base {
  int j;
public:
  void set_j(int num) { j=num; }
  int get_j() { return j; }
};
```

```
int main() {
  base *bp;
  derived d;

  bp = &d; // базовий вказівник вказує на об'єкт похідного класу

  // доступ до успадкованих елементів в похідному класі з використанням вказівника
на базовий клас
  bp->set_i(10);
  cout << bp->get_i() << " ";

/* Наступний оператор не працює! На нові елементи похідного класу не можна
вказувати базовим вказівником
   bp->set_j(88); // error
   cout << bp->get_j(); // error
*/
/* для доступу до нових членів похідного класу потрібно перетворити тип вказівника
на похідний клас */
   ((derived * ) bp)->set_j(88);
   cout << ((derived *) bp) ->get_j();
  return 0;
}
10
88

/* 9. listing 21 — адресна арифметика залежить від типу базового вказівника */
#include <iostream>
using namespace std;

class base {
  int i;
public:
  void set_i(int num) { i=num; }
  int get_i() { return i; }
};

class derived: public base {
  int j;
public:
  void set_j(int num) {j=num;}
  int get_j() {return j;}
};

int main() {
  base *bp;   // вказівник на базовий тип !
  derived d[2];

  bp = d;  // bp=&a[0] - базовий вказівник вказує на об'єкт похідного класу

  d[0].set_i(1);
  d[1].set_i(9);

  cout << bp->get_i() << " ";
  bp++;    // інкремент стосується базового, а не похідного типу !
  cout << bp->get_i(); // тому на екран виводиться невірне значення

  return 0;
}
1 0

/* 10. listing 22 — вказівники на члени класу через об'єкт */
#include <iostream>
using namespace std;
```

```
class cl {
public:
  cl(int i) { val=i; }
  int val;
  int double_val() { return val+val; }
};

int main() {
  int cl::*data; // вказівник на член дані класу
  int (cl::*func)(); // вказівник на член функцію класу
  cl ob1(1), ob2(2); // створення об'єктів

  data = &cl::val; // визначення зміщення члена val
  func = &cl::double_val; // визначення зміщення функції double_val()

  cout << "Значення: ";
  cout << ob1.*data << " " << ob2.*data << "\n";

  cout << "Подвоєне значення: ";
  cout << (ob1.*func)() << " ";
  cout << (ob2.*func)() << "\n";

  return 0;
}
```

Значення: 1 2
Подвоєне значення: 2 4

```
/* 11. listing 23 - вказівники на члени класу через вказівник на об'єкт */
#include <iostream>
using namespace std;

class cl {
public:
  cl(int i) { val=i; }
  int val;
  int double_val() { return val+val; }
};

int main() {
  int cl::*data;     // вказівник на член дані класу
  int (cl::*func)(); // вказівник на член функцію класу
  cl ob1(1), ob2(2); // створення об'єктів
  cl *p1, *p2;

  p1 = &ob1; // доступ до об'єктів через вказівники
  p2 = &ob2;

  data = &cl::val; // зміщення члена  даних val
  func = &cl::double_val; // зміщення члена функції double_val()

  cout << "Значення: ";
  cout << p1->*data << " " << p2->*data << "\n";

  cout << "Подвоєне значення: ";
  cout << (p1->*func)() << " ";
  cout << (p2->*func)() << "\n";

  return 0;
}
```

Значення: 1 2
Подвоєне значення: 2 4

```
/* 12. listing 25 — аргумент передається за допомогою явного вказівника  */
```

```
#include <iostream>
using namespace std;

void neg(int *i);
int main() {
  int x;
  x = 10;
  cout << x << " і його негативне значення ";
  neg(&x);
  cout << x << "\n";
  return 0;
}
void neg(int *i) {
  *i = -*i;
}
```
10 і його негативне значення -10

```
/* 13. listing 27 — аргумент передається через посилання (неявний вказівник) */
#include <iostream>
using namespace std;

void neg(int &i); // посилання

int main() {
  int x;

  x = 10;
  cout << x << " і його негативне значення ";
  neg(x); // оператор & більш не потрібний
  cout << x << "\n";

  return 0;
}

void neg(int &i) {
  i = -i; // i є посиланням і оператор * більш не потрібний
}
```
10 і його негативне значення -10

```
/* 14. listing 30 – використання посилань для переставлення елементів масиву */
#include <iostream>
using namespace std;

void swap(int &i, int &j);
int main() {
  int a, b, c, d;
  a = 1;
  b = 2;
  c = 3;
  d = 4;

  cout << "a i b: " << a << " " << b << "\n";
  swap(a, b); //оператор  & не потрібний
  cout << "a i b: " << a << " " << b << "\n";

  cout << "c i d: " << c << " " << d << "\n";
  swap(c, d);
  cout << "c i d: " << c << " " << d << "\n";

  return 0;
}

void swap(int &i, int &j) {
  int t;
```

```
  t = i; // оператор * не потрібний
  i = j;
  j = t;
}
a i b: 1 2
a i b: 2 1
c i d: 3 4
c i d: 4 3

/* 15. listing 31 — передача посилань на об'єкти */
#include <iostream>
using namespace std;

class cl {
  int id;
public:
  int i;
  cl(int i);
  ~cl();
  void neg(cl &o) { o.i = -o.i; } // при передачі посилання копія об’єкта не
                                  // створюється
};
cl::cl(int num) {
  cout << "Створення об'єкта " << num << "\n";
  id = num;
}
cl::~cl() {
  cout << "Знищення об'єкта " << id << "\n";
}

int main() {
  cl o(1);
  o.i = 10;
  o.neg(o);
  cout << o.i << "\n";

  return 0;
}
Створення об'єкта 1
-10
Знищення об'єкта 1

/* 16. listing 32 — функція повертає посилання */
#include <iostream>
using namespace std;

char &replace(int i); // повернення посилання
char s[80] = "Hello world!\n";

int main() {
  replace(5) = '='; // функція в лівій частині повертає посилання на символ
                    // який входить в рядок s, індекс якого заданий змінною i
                    // вставлення символу ”=” після слова Hello
  cout << s;
  return 0;
}

char &replace(int i)
{
  return s[i];
}
Hello=world!
```

```
/* 17. listing 33 — незалежні посилання, псевдоніми об'єктів*/
#include <iostream>
using namespace std;

int main() {
  int a;
  int &ref = a; // незалежне посилання
  a = 10;
  cout << a << " " << ref << "\n";
  ref = 100;
  cout << a << " " << ref << "\n";
  int b = 19;
  ref = b; // b->a
  cout << a << " " << ref << "\n";
  ref--; // a--
  cout << a << " " << ref << "\n";
  return 0;
}
10 10
100 100
19 19
18 18

/* 18. listing 36 — виділення динамічної пам'яті, оператори new, delete */
#include <iostream>
#include <new>      // заголовок для bad_alloc
using namespace std;
int main() {
  int *p;

  try {
    p = new int; // виділити пам'ять для int
  } catch (bad_alloc xa) {    // перехоплення виняткової ситуації
    cout << "Виняткова ситуація\n";
    return 1;
  }
  *p = 100;
  cout << "За адресом  " << p << "  ";
  cout << "записано значення " << *p << "\n";
  delete p;
  return 0;
}
По адресу 0x603010 записано значення 100

/* 19. listing 37 – ініціалізація динамічної пам'яті */
#include <iostream>
#include <new>
using namespace std;

int main() {
  int *p;
  try {
    p = new int (87); // ініціалізація значенням 87
  } catch (bad_alloc xa) {
    cout << "Виняткова сутуація\n";
    return 1;
  }
  cout << "По адресу " << p << "  ";
  cout << "записано число " << *p << "\n";
  delete p;
  return 0;
}
За адресою 0x603010 записано число 87
```

```
/* 20. listing 38 — виділення пам’яті під масив цілих чисел, new, delete [] */
#include <iostream>
#include <new>
using namespace std;
int main()
{
  int *p, i;
  try {
    p = new int [10]; // виділення пам'яті під вектор 10 int
  } catch (bad_alloc xa) {
    cout << "Виняткова ситуація\n";
    return 1;
  }
  for(i=0; i<10; i++ )   // ініціалізація вектора
    p[i] = i;
  for(i=0; i<10; i++)
    cout << p[i] << " ";
  delete [] p; // звільнення пам’яті, яка займав вектор
  return 0;
}
0 1 2 3 4 5 6 7 8 9

/* 21. listing 39 — виділення пам'яті для об'єктів */
#include <iostream>
#include <new>
#include <cstring>
using namespace std;

class balance {
  double cur_bal;
  char name[80];
public:
  void set(double n, char *s) {
    cur_bal = n;
    strcpy(name, s);
  }

  void get_bal(double &n, char *s) {
    n = cur_bal;
    strcpy(s, name);
  }
};

int main() {
  balance *p;
  char s[80];
  double n;

  try
 {
    p = new balance;
  } catch (bad_alloc xa) {
    cout << "Помилка при виділенні пам'яті\n";
    return 1;
  }

  p->set(1200.50, "Іванченко І.І."); // доступ до членів через вказівник
  p->get_bal(n, s);
  cout << s << " має на рахунку : " << n << " грн.\n";
  delete p;
  return 0;
}
Іванченко І.І. має на рахунку : 1200.5 грн.
```

```
/* 22. listing 40 — конструктор з параметрами */
#include <iostream>
#include <new>
#include <cstring>
using namespace std;

class balance {
  double cur_bal;
  char name[80];
public:
  balance(double n, char *s) {
    cur_bal = n;
    strcpy(name, s);
  }
  ~balance() {
    cout << "Знищення об'єкту ";
    cout << name << "\n";
  }
  void get_bal(double &n, char *s) {
    n = cur_bal;
    strcpy(s, name);
  }
};

int main() {
  balance *p;
  char s[80];
  double n;

  // конструктор з параметрами
  try {
    p = new balance (12387.87, "Іванченко І.І.");
  } catch (bad_alloc xa) {
    cout << "Виняткова ситуація\n";
    return 1;
  }

  p->get_bal(n, s);
  cout << s << " сума дорівнює: " << n;
  cout << "\n";
  delete p;
  return 0;
}
```

```
Іванченко І.І. сума дорівнює: 12387.9
Знищення об'єкту Іванченко І.І.
```

```
/* 23. listing 41 — масиви розміщені в динамічній пам'яті не можна ініціалізувати
                в операторі new, тому має бути конструктор без параметрів */
#include <iostream>
#include <new>
#include <cstring>
using namespace std;

class balance {
  double cur_bal;
  char name[80];
public:
  balance(double n, char *s) {
    cur_bal = n;
    strcpy(name, s);
  }
  balance() {} // конструктор без параметрів !
  ~balance() {
    cout << "Знищення ";
```

```
    cout << name << "\n";
  }
  void set(double n, char *s) {
    cur_bal = n;
    strcpy(name, s);
  }
  void get_bal(double &n, char *s) {
    n = cur_bal;
    strcpy(s, name);
  }
};

int main() {
  balance *p;
  char s[80];
  double n;
  int i;
  try {
    p = new balance [3]; // виділення пам'яті під масив об'єктів
  } catch (bad_alloc xa) {
    cout << "Виняткова ситуація\n";
    return 1;
  }

  // використовується оператор . , а не -> !
  p[0].set(12387.87, "Іванченко І.І."); // ініціалізація масиву об'єктів
  p[1].set(144.00, "Василенко В.В.");
  p[2].set(-11.23, "Степаненко С.С.");
  for(i=0; i<3; i++) {
    p[i].get_bal(n, s);
    cout << s << " сума = " << n;
    cout << "\n";
  }
  delete [] p;
  return 0;
}
```

Іванченко І.І. сума = 12387.9
Василенко В.В. сума = 144
Степаненко С.С. сума = -11.23
Знищення Степаненко С.С.
Знищення Василенко В.В.
Знищення Іванченко І.І.

/* 24. listing 42 - альтернативний оператор new(nothrow), який при відсутності пам'яті не генерує вийняткову ситуацію, а повертає 0. (для сумісності із старими програмами */

```
#include <iostream>
#include <new>
#define N 5
using namespace std;
int main()
{
  int *p, i;
  p = new(nothrow) int[N]; // застосування опції nothrow
  if(!p) {
    cout << "Помилка виділення пам'яті.\n";
    return 1;
  }
  for(i=0; i<N; i++) p[i] = i;
  for(i=0; i<N; i++) cout << p[i] << " ";
  delete [] p; // звільнення пам'яті
  return 0;
}
```

0 1 2 3 4