Міністерство освіти і науки України
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт
з дисципліни
“Об’єктно-орієнтоване програмування”
для студентів спеціальностей
121 "Інженерія програмного забезпечення" та
122 "Комп'ютерні науки"
всіх форм навчання

2023

Методичні вказівки до лабораторних робіт з дисципліни “Об’єктно-орієнтоване програмування” для студентів спеціальностей 121 "Інженерія програмного забезпечення" та 122 "Комп'ютерні науки" всіх форм навчання / Г.В Табунщик, Н.О. Миронова, Т.В.Голуб, Л.Ю. Дейнега. – Запоріжжя: НУ «Запорізька політехніка», 2023. – 31 с.

Автори: Г. В. Табунщик, к.т.н., доцент,
Н. О. Миронова, к.т.н., доцент,
Т.В. Голуб, к.т.н., доцент
Л.Ю. Дейнега, ст. викладач

Рецензент: В.І Дубровін, к.т.н., доцент

Відповідальний
за випуск: Т.В. Голуб, к.т.н., доцент

Затверджено
на засіданні кафедри ПЗ
Протокол № 12 від 09.06.2023

Рекомендовано до видання
на засіданні НМК КНТ
Протокол №1 від 31.08.23

ЗМІСТ

Вступ.....4
Лабораторна робота №1 Введення в класи.....5
Лабораторна робота №2 Динамічні класові типи.....9
Лабораторна робота №3 Успадкування.....15
Лабораторна робота №4 Введення/виведення потоками. робота з файлами.....19
Лабораторна робота №5 Перевантаження операцій.....20
Лабораторна робота №6 Віртуальні функції.....25
Лабораторна робота №7 Введення в узагальнене програмування.....27
Лабораторна робота №8 Обробка виняткових ситуацій.....29
Література.....31

ВСТУП

Метою даного курсу є вивчення теоретичних основ та практичних аспектів об'єктно-орієнтованого програмування. Дисципліна “Об’єктно-орієнтоване програмування” спрямована на отримання студентом базових знань та практичних навичок з основ сучасної технології створення складних програмних продуктів на базі ідей і принципів об'єктно-орієнтованого методу. Такі знання призначені для використання у розробках програмного забезпечення інформаційних технологій з урахуванням сучасних вимог у відношенні до надійності, якості інтерфейсу та ефективності програмних продуктів, які створюються. Отримані знання та практичні навички мають служити базою для опанування у подальшому нових майбутніх систем програмування, які базуються на ідеях візуального програмування, CASE-технологіях, штучного інтелекту тощо.

Звіт з лабораторної роботи повинен містити: тему, мету, індивідуальне завдання; структурну схему організації даних відповідно до завдання; текст програми; результати роботи програми; висновки.

В якості інструментальної мови програмування для виконання лабораторних робот рекомендовано використовувати мову програмування C++. Головною вимогою для використання компілятора є підтримка стандарту ISO/IEC 14882 “Standard for the C++ Programming Language”.

ЛАБОРАТОРНА РОБОТА №1
ВВЕДЕННЯ В КЛАСИ

Мета роботи

Навчитись будувати класи та використовувати їх при створенні програм.

Порядок виконання лабораторної роботи

1. Застосувавши конспект лекцій та додаткову літературу, вивчити основні принципи створення класів.
2. Виконати індивідуальне завдання відповідно до номеру варіанта, застосувавши мову програмування С++. При виконанні програм слід використовувати конструктори та деструктори.
3. Оформити звіт.
4. Зробити висновки.
5. Відповісти на контрольні запитання.

Індивідуальні завдання

Варіант 1. Створити клас для роботи з тривимірними векторами. Передбачити функції для виконання наступних операцій: консольне введення і виведення значень вектора; ініціалізація вектора; складання та віднімання векторів; обчислення довжини вектора.

Варіант 2. Створити клас *Person*. Змінна типу *Person* повинна вміщувати наступну інформацію про деяку людину: прізвище, ім’я та по–батькові; адреса; рік народження; телефон; зріст, вага, стать. Передбачити функції для виконання наступних операцій: консольне введення і виведення значень типу *Person*; ініціалізація. Створити динамічний масив, що містить інформацію про деяку групу людей. Підрахувати кількість людей за статтю (*friend*-функція).

Варіант 3. Створити клас *Alfa* таким чином, щоб при створенні першого і знищенні останнього об’єкта цього типу на екран видавалися відповідні повідомлення. Вказівка: застосувати статичні компоненти класу.

Варіант 4. Створити клас *Beta* таким чином, щоб при знищенні останнього об’єкта на екран видавалося повідомлення про найбільшу кількість об’єктів типу *Beta*, що існували, та час існування популяції об’єктів. Вказівка: застосувати статичні компоненти класу.

Варіант 5. Створити клас *Delta* таким чином, щоб кожний об’єкт містив свій персональний номер (дескриптор об’єкта) та функцію, яка повертає його значення. Дескриптор об’єкта – унікальне для об’єктів даного типу ціле число. Вказівка: застосувати статичні компоненти класу.

Варіант 6. Створити клас таким чином, щоб він відстежував кількість існуючих у даний момент об’єктів та обсяг оперативної пам’яті, який вони займають. Вказівка: застосувати статичні компоненти класу.

Варіант 7. Створити клас для виконання операцій з комплексними числами. Передбачити операції: складання та віднімання; множення та ділення; обчислення модулю; консольне введення та виведення; ініціалізацію.

Варіант 8. Створити клас *Organization*. Змінна типу *Organization* повинна містити наступну інформацію: назва; адреса; директор; телефон. Передбачити функції для виконання наступних операцій: консольне введення і виведення значень типу *Organization*; ініціалізація. Створити каталог організацій міста (динамічний масив) та відсортувати за назвою (*friend*-функція).

Варіант 9. Створити клас *Book*. Змінна типу *Book* повинна містити наступні поля: назва; кількість авторів; автори; рік видання; кількість сторінок. Передбачити функції для виконання наступних операцій: ініціалізація книги, додавання авторів, консольне введення/виведення інформації про книгу. Створити каталог книг (динамічний масив), впорядкувати елементи за назвою (*friend*-функція).

Варіант 10. Створити клас *Film*. Змінна типу *Film* повинна містити наступні поля: назва; рік створення; режисер; кількість акторів;

актори. Передбачити функції для виконання наступних операцій: ініціалізація інформації, додавання акторів, консольне введення/виведення інформації про відео. Створити каталог відео (динамічний масив), впорядкувати елементи за назвою (*friend*-функція).

Варіант 11. Створити клас *House*. Змінна типу *House* повинна містити наступні поля: вулиця; номер; кількість квартир, поверхів, під'їздів, кількість квартир на поверсі; дата початку будівництва; дата закінчення будівництва; назва будівничої фірми. Передбачити функції для виконання наступних операцій: ініціалізація інформації, консольне введення/виведення інформації про будинок, розрахунок кількості квартир у під'їзді з використанням *friend*-функції.

Варіант 12. Створити клас *Train*. Змінна типу *Train* повинна містити наступні поля: номер; кількість зупинок; кількість вагонів; час відправлення; час прибуття. Передбачити функції для виконання наступних операцій: ініціалізація інформації, консольне введення та виведення інформації про потяг, розрахунок часу, що витрачається на маршрут. Створити реєстр потягів (динамічний масив).

Варіант 13. Створити клас вибірка *Sample* розмірності N. Передбачити функції для виконання наступних операцій: консольне введення/виведення значень вибірки, розрахунок середнього, дисперсії, розмаху, середньоквадратичного відхилення. Розробити дружню функцію для розрахунку критерію Кохрена ($G=S_{max} / \sum S_i$).

Варіант 14. Скласти опис класу багаточленів від однієї змінної, що задаються ступенем багаточлена і масивом коефіцієнтів. Передбачити методи для обчислення значення багаточлена для заданого аргументу, операції складання, віднімання та множення багаточленів з отриманням нового об'єкта – багаточлена, виведення на екран опису багаточлена.

Варіант 15. Створити клас вибірка *Sample* розмірності N. Передбачити функції для виконання наступних операцій: консольне введення/виведення значень вибірки, розрахунок середнього,

дисперсії, розмаху, середньоквадратичного відхилення. Розробити дружню функцію для розрахунку критерію Фішера ($F=\sigma_1/\sigma_2$).

Контрольні запитання

1. Чим відрізняються поняття класу та структури у C++? Дайте загальне визначення класу.
2. Наведіть загальну структуру класу.
3. Що таке інтерфейс класу та його реалізація?
4. У чому полягає роль конструкторів та деструкторів у класі?
5. Які способи можна застосувати для ініціалізації об'єкта класу? Як для цього застосовуються конструктори?
6. Чим відрізняються функції – члени класу від функцій, не пов'язаних з будь-яким класом?
7. Що означає об'єкт класу?
8. Представлення об'єкта у пам'яті.
9. Чи можливо перевантажувати функції-члени класу?
10. В чому полягає особливість статичних елементів класу?
11. Які області видимості використовують для створення класів?
12. В чому різниця між *inline*-функціями та іншими функціями-членами класу?
13. Коли використовують дружні функції?
14. Чи можна присвоювати значення одного об'єкта класу іншому?
15. Які специфікатори видимості для членів класу Ви знаєте?
16. Чи можливо створювати масиви об'єктів?
17. Для чого використовується покажчик this?
18. Чи можливо передавати об'єкти як аргументи функціям?
19. Чи можливо, щоб функція повертала значення типу клас?

ЛАБОРАТОРНА РОБОТА №2
ДИНАМІЧНІ КЛАСОВІ ТИПИ

Мета роботи

Навчитись використовувати динамічні класи при створенні програм.

Порядок виконання лабораторної роботи

1. Застосувавши конспект лекцій та додаткову літературу, вивчити основні принципи роботи з динамічними класовими типами.
2. Виконати індивідуальне завдання відповідно до номера варіанта, застосувавши мову програмування С++. Клас, що розроблюється, повинен містити конструктор копіювання.
3. Оформити звіт.
4. Зробити висновки.
5. Відповісти на контрольні запитання.

Індивідуальне завдання

Варіант 1. Створити динамічний клас *Catalog*, що базується на зв'язаному списку, де кожний елемент списку – структура типа *Book*. Клас повинен містити наступні операції:

- *Catalog*(*Catalog*&) – конструктор копіювання;
- *add_book()* – додавання книги до каталогу;
- *del_book()* – видалення книги з каталогу;
- *find_by_autor()* – пошук книги у каталозі за автором;
- *show_cataloque()* - відображення каталогу книг.

Варіант 2. Створити динамічний клас *Book*, що містить динамічний масив *Authors* (інформація про автора містить: П.І.Б., кількість книжок, рік народження), а також всю необхідну інформацію про книжку. Клас повинен містити наступні операції:

- *Book*(*Book*&) – конструктор копіювання;
- *add_author()* – додавання авторів;
- *del_author()* – видалення авторів;
- *count_authors()* – підраховує кількість авторів.

Також потрібно реалізувати масив книжок та *friend*-функції введення/виведення інформації до/з каталогу.

Варіант 3. Створити динамічний клас *University*, що базується на однозв'язному списку, де кожний елемент списку містить інформацію про факультети: назву факультету, прізвище декана, кількість груп на факультеті, загальну кількість студентів у групах. Клас повинен містити наступні операції:

- *University*(*University*&) – конструктор копіювання;
- *add_group()* – додавання групи;
- *del_group()* – видалення групи;
- *calc_stud()* – підрахунок кількості студентів на факультеті.

Варіант 4. Створити динамічний клас *Train*, де елементи типу *Carriage* (вагон) зв'язані структурою типу двозв'язний список. Клас повинен містити наступні операції:

- *Train*(*Train*&) – конструктор копіювання;
- *add_carriage()* – додавання вагонів;
- *del_carriage()* – вилучення вагонів;
- *calc_places()* – підрахунок кількості місць в потязі.

Варіант 5. Створити динамічний клас *Route* на основі двозв'язного списку, де кожний елемент – структура типу *stop* (зупинка). Клас повинен містити наступні операції:

- *Route(Route&)* – конструктор копіювання;
- *add_stop()* – додавання зупинки;
- *len_route()* – розрахунок довжини маршруту;
- *time_route()* – розрахунок часу руху.

Варіант 6. Створити клас для роботи з одновимірними динамічними масивами значень типа *unsigned int*. Передбачити функції-компоненти класу для виконання наступних операцій: конструктор копіювання, операції динамічного присвоєння; поелементного складання та віднімання; об'єднання двох масивів у один (конкатенація); упорядкування масиву за збільшенням та за зменшенням; консольне введення та виведення масиву.

Варіант 7. Створити динамічний клас для роботи з рядком - масивом символів. Максимальна довжина рядка – 65535 символів. Передбачити функції для виконання наступних операцій: ініціалізація ASCIIZ–рядка (рядка, що завершується нуль–байтом); введення з клавіатури; виведення на екран; повернення кількості символів у рядку; конкатенація рядків; пошук підрядка; пошук і заміна підрядка. Клас має містити конструктор копіювання.

Варіант 8. Створити динамічний класовий тип для роботи з рядком - масивом символів. Максимальна довжина рядка – 65535 символів. Вважати, що довжина рядка зберігається в нульовому елементі масиву. Передбачити операції створення, знищення, ініціалізації об'єкта, а також операції динамічного присвоєння та конкатенації. Операції створення, знищення та ініціалізації реалізувати за допомогою конструкторів та деструкторів, решту операцій – за допомогою компонентних функцій. Клас має містити конструктор копіювання.

Варіант 9. Створити клас для роботи з множинами цілих чисел. Максимальна кількість елементів множини – 65534, інтервал значень елементів множини відповідає типу ***int*.** Передбачити функції для виконання наступних операцій: ініціалізація за допомогою масиву цілого типу; введення з клавіатури; виведення на екран; об'єднання множин; віднімання множин; перетин множин; перевірка відношення включення; перевірка еквівалентності; перевірка належності даного числа до множини. Клас має містити конструктор копіювання.

Варіант 10. Створити клас *List* для роботи зі структурою типу "однозв'язний список". Елемент списку має тип, що відповідає бібліотечному класу *string*. Передбачити функції для виконання наступних операцій:

- *List*(*List*&) – конструктор копіювання;
- *putinbeg()* – створити новий елемент списку на його початку;
- *getoutbeg()* – добути і вилучити перший елемент списку;
- *putinend()* – створити новий елемент в кінці списку;
- *getoutend()* – добути і вилучити останній елемент списку;
- *print()* – вивести список на екран;

— *isempty()* – повернути значення true (тип *bool*), якщо список порожній.

Варіант 11. Створити клас *Astack* – стек, що базується на масиві покажчиків фіксованого розміру. Передбачити, щоб стек мав можливість зберігати значення типу, що відповідає бібліотечному класу *string*. Передбачити функції для виконання наступних операцій:

— *Astack*(*Astack*&) – конструктор копіювання;
— *push()*– занести у стек значення;
— *pop()* – добути та вилучити значення з вершини стека;
— *print()* – вивести всі значення стека на екран;
— *num()* – повернути кількість значень, що містяться у стеку;
— *isempty()* – повернути значення true (тип *bool*), якщо стек порожній.

Варіант 12. Створити клас *Aqueue* – структура типу черга, що базується на масиві покажчиків фіксованого розміру. Передбачити, щоб черга мала можливість вміщувати значення типу, що відповідає бібліотечному класу *string*. Передбачити функції – члени класу для виконання таких операцій:

— *Aqueue*(*Aqueue*&) – конструктор копіювання;
— *add()* – занести до черги надане значення;
— *pop()* – добути і вилучити значення із черги;
— *print()* – вивести всі значення черги на екран;
— *num()* – повернути кількість значень, що містяться в черзі;
— *isempty()* – повернути значення *true* (тип *bool*), якщо черга пуста.

Варіант 13. Створити клас *Lstack* – стек, що базується на структурі однозв'язного списку. Передбачити можливість зберігання значення, що відповідають бібліотечному класу *string*. Передбачити функції-члени класу для виконання наступних операцій:

— *Lstack Lstack*&) – конструктор копіювання;
— *push()* – занести до стека надане значення;
— *pop()* – добути і вилучити значення з вершини стека;
— *print()* – вивести усі значення стека на екран;
— *num()* – повернути кількість значень, що містяться в стеку;

— *isempty()* – повернути значення *true* (тип *bool*), якщо стек порожній.

Вказівки: передбачити обробку виключної ситуації: добування значення з порожнього стека.

Варіант 14. Створити клас *Lqueue* – структура типу "черга", що базується на структурі однозв'язного списку. Тип значення елементів черги – *int*.. Передбачити функції для виконання таких операцій:

— *Lqueue*(*Lqueue*&) – конструктор копіювання;
— *add()* – занести значення в кінець черги;
— *pop()* – добути і вилучити значення з початку черги;
— *print()* – вивести всі значення черги на екран;
— *num()* – знайти кількість значень, що знаходяться у черзі;
— *isempty()* – повернути значення *true* (тип *bool*), якщо черга пуста.

Вказівки: передбачити обробку виключної ситуації: добування значення із порожньої черги.

Варіант 15. Створити клас *Set* для роботи зі структурами типу "Множина". Тип елементу структури обрати самостійно. Множина є набором елементів, кожний з яких має унікальне значення. При додаванні елемента (операція *include*), який вже є у множині, він не додається. Для цієї структури визначаються операції, звичайні для математичних множин – об'єднання, перетин, віднімання, доповнення. Елементи множини можуть автоматично упорядковуватися, що дає змогу використовувати швидкі операції пошуку елемента з наданим значенням. Передбачити функції-члени класу для виконання таких операцій:

— *Set(Set&)* – конструктор копіювання;
— *include()* – додати новий елемент до множини;
— *exclude()* – вилучити наданий елемент з множини;
— *union()* – об'єднання множин;
— *inters()* – перетин множин;
— *substr()* – віднімання множин;
— *print()* – вивести усі значення множини на екран;
— *num()* – кількість значень множини (потужність).

Варіант 16. Створити клас *Bitv* для роботи з бітовими векторами довільної довжини. Бітовий вектор – послідовність значень, які можуть мати значення 0 (не істина) або 1 (істина). Кожен елемент бітового вектора повинен займати у пам'яті один біт. Для розміщення бітового вектора у пам'яті застосувати динамічний масив відповідного розміру. Тип елемента масиву – носія бітового вектора – беззнаковий, розміром 1, 2 або 4 байти. У класі *Bitv* передбачити такі функції:

Bitv() – конструктор без параметрів;

Bitv(char)* – конструктор для ініціалізації за допомогою значення, наданого символьним рядком;

Bitv(Bitv&) – конструктор копіювання;

~Bitv() – деструктор;

void setin0(int) – встановлення біта з наданим номером в 0;

void setin1(int) – встановлення біта в 1;

void flip(int) – інвертування біта з наданим номером;

void print() – виведення бітового вектора на екран;

size() – розмір бітового вектора.

Контрольні запитання

1. Чим відрізняється динамічний клас від статичного класу?
2. Яку структуру має об'єкт динамічного класу, у чому полягають його особливості?
3. Поясніть особливості виконання операції присвоєння для об'єктів динамічного класу.
4. У чому полягають особливості операцій створення та знищення об'єктів динамічного класу?
5. Коли використовується конструктор копіювання?
6. Де у Вашій програмі використовується конструктор копіювання?
7. На що вказує покажчик **this*?
8. Особливості присвоювання значень об'єктів.
9. Розкрийте алгоритм передачі об'єкта класу як параметра функції.
10. Який механізм повернення значень типу клас?
11. Чи можливо об'єкти класу передавати за посиланням?

ЛАБОРАТОРНА РОБОТА №3
УСПАДКУВАННЯ

Мета роботи

Навчитись використовувати успадкування при розробці об’єктно-орієнтованих програм.

Порядок виконання лабораторної роботи

1. Застосувавши конспект лекцій та додаткову літературу, вивчити основні принципи успадкування.
2. Виконати індивідуальне завдання відповідно до номера варіанта, застосувавши мову програмування C++.
3. Оформити звіт.
4. Зробити висновки.
5. Відповісти на контрольні запитання.

Індивідуальне завдання

Варіант 1. Розробити клас *CFile*, що інкапсулює у себе такі функції роботи з файлами, як *Open, Close, Seek, Read, Write, GetPosition* і *GetLength*. На базі цього класу створити похідний клас *CMyDataFile* – файл, що містить дані деякого визначеного типу *MyData*, а також заголовок, що полегшує доступ до цього файлу.

Варіант 2. Описати базовий клас *Рядок*. Об’єкти класу повинні містити наступні дані: покажчик на *char*, що зберігає адресу динамічно виділеної пам'яті для розміщення символів рядка; значення типа *int*, що зберігає довжину рядка в байтах.

Клас повинен містити наступні методи: конструктор без параметрів; конструктор, який приймає як параметр рядок, що закінчується нульовим байтом; конструктор, що приймає як параметр символ; конструктор копіювання; отримання довжини рядка; очищення довжини рядка; очищення рядка (зробити рядок порожнім); деструктор.

Створити похідний від класу *Рядок* клас *Пароль*, для якого повинні виконувати наступні вимоги: рядки даного класу складаються з букв і цифр; регістр букв розрізняється; довжина пароля не може бути

менше заданого числа; при введенні символи відображаються зірочками *.

Клас *Пароль* повинний містити: конструктор без параметрів; конструктор, який приймає як параметр рядок, що закінчується нульовим байтом; конструктор копіювання; деструктор.

Варіант 3. Створити базовий клас *Рядок*, як у завданні варіанту 2. Створити похідний від класу *Рядок* клас *Рядок_Ідентифікатор*.

Рядки даного класу будуються за правилами запису ідентифікаторів у мові С і можуть містити лише ті символи, які можуть входити до складу С-ідентіфікаторів. Якщо початкові дані не задовольняють правилам запису ідентифікатора, то створюється порожній *Рядок_Ідентифікатор*.

Клас повинен містити: конструктор без параметрів; конструктор, який приймає як параметр рядок, що закінчується нульовим байтом; конструктор, що приймає як параметр символ; конструктор копіювання; переведення всіх символів рядка у верхній регістр; переведення всіх символів рядка в нижній регістр; пошук першого входження символу в рядку; деструктор.

Варіант 4. Створити базовий клас *Людина*. Кожний об’єкт класу повинен містити наступні дані: ПІБ, рік народження, стать.

Клас повинен виконувати наступні дії: ініціалізація інформації, введення-виведення інформації.

Створити похідний клас *Студент*, що має додаткові дані: рік вступу, № залікової книжки, кількість дисциплін, що вивчаються, дисципліни (динамічний масив), середній бал.

Клас повинен виконувати наступні функції: ініціалізація інформації, додавання дисциплін, розрахунок середнього балу, виведення загальної інформації про студента.

Варіант 5. Створити базовий клас *Людина*, відповідно до варіанту 4.

Створити похідний клас *Інженер*, що містить додаткові дані: рік закінчення, ВНЗ, спеціальність, тип диплому, тип навчання, перекваліфікація (динамічний масив), місце роботи, заробітня плата.

Клас повинен містити наступні методи: ініціалізації інформації, розрахунок заробітної плати, розрахунок щорічного доходу, додавання інформацій про перекваліфікацію.

Варіант 6. Створити базовий клас *Людина*, відповідно до варіанту 4.

Створити похідний клас *Користувач_бібліотеки*, що містить наступні дані: номер читацького квитку, дата видачі, перелік книжок, що видавались (динамічний масив), щомісячній читацький внесок, статус користувача.

Клас повинен містити наступні методи: ініціалізація інформації, розрахунок знижки (залежить від кількості виданих книжок), розрахунок щорічного читацького внеску, додавання інформації про книжки, введення-виведення загальної інформації про користувача.

Варіант 7. Створити базовий клас *Людина*, відповідно до варіанту 4.

Створити похідний клас *Співробітник*, що містить наступні дані: табельний номер, початок роботи, кінець роботи, посада.

Клас повинен містити наступні методи: конструктор, деструктор, ініціалізація інформації, методи для введення-виведення загальної інформації.

Варіант 8. Створити базовий клас *Продукт*. Кожний об’єкт класу повинен містити наступні дані: назва, вартість, вага.

Клас повинен виконувати наступні дії: ініціалізація інформації, введення-виведення інформації.

Створити похідний клас *Купівля*, що має додаткові дані: кількість придбаного товару в штуках, загальна вартість всього придбаного товару, загальна вага всього придбаного товару.

Клас повинен виконувати наступні функції: ініціалізація інформації, зміна кількості придбаного товару, розрахунок вартості товану за певний обсяг– задається користувачем (наприклад: за 3 кг, за 7 кг), виведення загальної інформації про товар.

Варіант 9. Створити базовий клас *Animal*. Кожний об’єкт класу повинен містити наступні дані: ім’я тварини, вага, вік, стать.

Клас повинен виконувати наступні дії: ініціалізація інформації, введення-виведення інформації.

Створити похідний клас *Cat*, що має додаткові дані: порода кота, колір кота, середня тривалість життя, середня вага кота цієї породи, середня кількість корму на добу.

Клас повинен виконувати наступні функції: ініціалізація інформації, зміна кольору тварини, розрахунок необхідної кількості корму на заданий період часу, виведення загальної інформації про кота.

Варіант 10. Створити базовий клас *Animal*. Кожний об'єкт класу повинен містити наступні дані: вид тварини, порода тварини, ім'я тварини, вага, вік, стать.

Клас повинен виконувати наступні дії: ініціалізація інформації, введення-виведення інформації.

Створити похідний клас *Dog*, що має додаткові дані: порода собаки, колір собаки, середня тривалість життя, середній зріст собаки цієї породи, середня кількість корму на добу.

Клас повинен виконувати наступні функції: ініціалізація інформації, зміна кольору тварини, розрахунок необхідної кількості корму на заданий період часу, виведення загальної інформації про собаку.

Контрольні запитання

1. Чи залежить представлення об'єкту похідного класу в пам'яті комп'ютера від атрибутів доступу?
2. На що впливають атрибути доступу?
3. Що означає множинне успадкування?
4. Перелічіть основні правила успадкування.
5. Коли використовується простір імен?
6. Що містить простір імен *std*?
7. Як підключити простір імен?
8. Чи можливо підключити декілька просторів імен?
9. Як зробити компоненти доступними у похідному класі, але закритими від зовнішнього доступу?

ЛАБОРАТОРНА РОБОТА №4
ВВЕДЕННЯ/ВИВЕДЕННЯ ПОТОКАМИ. РОБОТА З ФАЙЛАМИ

Мета роботи

Навчитись маніпулювати потоками введення/виведення, працювати з файлами та форматувати дані при введенні/виведенні.

Порядок виконання лабораторної роботи

1. Застосувавши конспект лекцій та додаткову літературу, вивчити основні принципи введення/виведення потоками, роботи з файлами.
2. Виконати індивідуальне завдання відповідно до номера варіанта, застосувавши мову програмування C++.
3. Оформити звіт.
4. Зробити висновки.
5. Відповісти на контрольні запитання.

Індивідуальне завдання

Для завдання з лабораторної роботи №2 реалізувати методи консольного та файлового введення/виведення, створити маніпулятори *insetup* та *outsetup* для форматування потоків введення/виведення та відповідно їх застосувати.

Контрольні запитання

1 Поясніть зміст поняття "потік" у сучасному програмуванні.

2 Що собою представляє ієрархія класів для виконання потокових операцій?

3 Поясніть, як виконуються операції з потоками.

4 Поясніть особливості програмування та виконання операцій створення та знищення потоків, зв’язування їх з файлами.

5 В чому полягають переваги потокових операцій введення/виведення? Які вони мають недоліки?

6 Для чого використовуються маніпулятори?

7 Перелічіть стандартні потоки введення/виведення.

ЛАБОРАТОРНА РОБОТА №5
ПЕРЕВАНТАЖЕННЯ ОПЕРАЦІЙ

Мета роботи

Навчитись використовувати перевантаження математичних операцій та операцій введення-виведення при розробці класів.

Порядок виконання лабораторної роботи

1. Застосувавши конспект лекцій та додаткову літературу, вивчити основні принципи перевантаження.
2. Виконати індивідуальне завдання відповідно до номера варіанта, застосувавши мову програмування C++.
3. Оформити звіт.
4. Зробити висновки.
5. Відповісти на контрольні запитання.

Індивідуальне завдання

Варіант 1. Створити динамічний клас для роботи з рядками (послідовностями символів). Максимальна довжина послідовності – 65535, код завершення послідовності – нуль. Здійснити перевантаження символів операцій:

"=" – динамічне присвоєння,

" << " , " >> " – консольне введення-виведення значень;

" << " , " >> " - введення з файлу і виведення в файл.

Передбачити можливість множинного введення.

Варіант 2. Створити динамічний клас для виконання операцій з динамічними масивами чисел дійсного типу. Зробити перевантаження символів операцій:

"=" – динамічне присвоєння,

"+" – поелементне додавання,

"–" – поелементне віднімання,

"*" – скалярне множення масивів,

"<<" – циклічний зсув елементів масиву вліво,

">>" – циклічний зсув елементів масиву вправо,

"[]" – доступ до елементу за індексом з контролем існування символа.

Варіант 3. Створити динамічний клас для виконання операцій з динамічними масивами. Зробити перевантаження символів операцій:

"=" – динамічне присвоєння,

" << " , " >> " – консольне введення-виведення значень;

" << " , " >> " - введення з файлу і виведення в файл таким чином:

f << A або A >> f – виведення (запис) значення A в файл f,

f >> A або A << f – введення (читання) значення A з файлу f.

Тип елементу масиву обрати самостійно.

Варіант 4. Створити динамічний клас для роботи з рядками (послідовностями символів). Максимальна довжина послідовності – 65535, код завершення послідовності – нуль. Здійснити перевантаження символів операцій:

" + " – конкатенація (з'єднання) рядків,

" – " – вилучення підрядку,

" < " – відношення "менше",

" <= " – відношення "менше чи дорівнює",

" > " – відношення "більше",

" >= " – відношення "більше чи дорівнює",

" == " – відношення "дорівнює",

" != " – відношення "не дорівнює".

За відношення порядку ("менше", "більше", "менше чи дорівнює", "більше чи дорівнює") вважати відношення лексикографічного порядку (тобто того порядку, що реалізується у звичайному словнику або у картотеці).

Варіант 5. Створити динамічний клас для роботи з рядками (послідовностями символів). Максимальна довжина послідовності – 65535, код завершення послідовності – нуль. Здійснити перевантаження символів операцій:

"=" – динамічне присвоєння,

" << " , " >> " – консольне введення-виведення значень;

" << " , " >> " - введення із файлу і виведення у файл з символами таким чином:

f << A або A >> f – виведення (запис) значення A в файл f,

f >> A або A << f – введення (читання) значення A з файлу f.

Варіант 6. Створити динамічний клас для роботи з рядками. Максимальна довжина послідовності – 254. Перший байт повинен містити інформацію про фактичну кількість елементів масиву. Здійснити перевантаження операцій:

" = "– присвоєння,
" + " – конкатенація (з'єднання) рядків,
" <= " – відношення "менше чи дорівнює",
" >= " – відношення "більше чи дорівнює",
" == " – відношення "дорівнює",
" != " – відношення "не дорівнює".

Варіант 7. Створити динамічний клас для роботи з рядками. Максимальна довжина послідовності – 254. Перший байт повинен містити інформацію про фактичну кількість елементів масиву. Здійснити перевантаження операцій:

" = "– присвоєння,
" << " , " >> " – консольне введення-виведення значень;
" << " , " >> " – введення з файлу і виведення в файл з символами таким чином:
f << A або A >> f – виведення (запис) значення A в файл f,
f >> A або A << f – введення (читання) значення A з файлу f.

Варіант 8. Створити клас для виконання точних обчислень з раціональними числами. Створити функцію скорочення раціонального числа (приведення до нормальної форми). Здійснити перевантаження символів операцій:

" = " – присвоєння,
" + " – додавання,
" – " – віднімання,
" * " – добуток,
" == " , " != " – рівність, не рівність,
" < " , " <= " , " > " , " >= " – відношення порядку.

Варіант 9. Створити клас для виконання операцій з тривимірними векторами. Вважати, що компоненти вектора мають дійсний тип. Здійснити перевантаження символів операцій:

" + " – додавання векторів,
" – " – векторне віднімання,

" * " – скалярний добуток векторів,
" == " , " != " – рівність, нерівність,
Передбачити функцію обчислення довжини вектора.

Варіант 10. Створити клас для виконання операцій з двовимірними векторами. Вважати, що компоненти мають цілий тип. Здійснити перевантаження операцій:

“ ^ ” – розрахунок модуля вектора;
“ + ” – скалярний добуток векторів;
“ == ”, “ != ” – рівність, нерівність векторів;
“ — ” – відстань між двома векторами.

Варіант 11. Створити динамічний клас для роботи з матрицями (динамічними). Вважати, що компоненти мають цілий тип. Здійснити перевантаження символів операцій:

“ = ” – присвоєння,
“ + ” – додавання,
“ – ” – віднімання,
“ ~ ” – отримання оберненої матриці,
“ * ” – поелементне множення.

Варіант 12. Створити динамічний клас для роботи з матрицями. Вважати, що компоненти мають цілий тип. Здійснити перевантаження символів операцій:

“ = ” – присвоєння,
“ * ” – поелементне множення,
“ ~ ” – розрахунок визначника матриці,
“ ^ ” – піднесення елементів матриці до ступеню.

Варіант 13. Перевантажити символ операції “>>” таким чином, щоб можна було робити копіювання файлу за допомогою такого оператору:

fa >> fb ;

де fa, fb – потоки, що пов’язані з файлами.

Операція “>>” повинна повертати значення потоку fa так, щоб можна було робити множинне копіювання:

fa >> fb >> fc;

Варіант 14. Створити динамічний клас для виконання операцій з динамічними масивами чисел дійсного типу. Зробити перевантаження символів операцій:

"=" – динамічне присвоєння,

" << " , " >> " – консольне введення-виведення значень;

" << " , " >> " – введення із файлу і виведення у файл з символами таким чином:

f << A або A >> f – виведення (запис) значення A в файл f,

f >> A або A << f – введення (читання) значення A з файлу f.

Передбачити можливість множинного введення-виведення.

Контрольні запитання

1. Для чого використовують перевантаження операцій?
2. В чому різниця між перевантаженням операцій як членів класу і *friend*-функцій?
3. Які оператори не можна перевантажувати як *friend*-функції?
4. Які оператори не можна перевантажувати як члени класу?
5. В чому полягає особливість перевантаження операторів введення/виведення “<<”, “>>”?
6. Назвіть особливості перевантаження унарних та бінарних операторів.
7. Чи можливо потік передавати до функції за значенням?
8. Чи можливо змінювати приорітет операції?

ЛАБОРАТОРНА РОБОТА №6
ВІРТУАЛЬНІ ФУНКЦІЇ

Мета роботи

Навчитись використовувати віртуальні функції при проектуванні успадкування в об'єктно-орієнтованих програмах.

Порядок виконання лабораторної роботи

1. Застосувавши конспект лекцій та додаткову літературу, вивчити основні принципи використання віртуальних функцій.
2. Виконати індивідуальне завдання відповідно до номера варіанта, застосувавши мову програмування C++.
3. Оформити звіт.
4. Зробити висновки.
5. Відповісти на контрольні запитання.

Індивідуальне завдання

Варіант 1. Наданий такий клас:

```
class Base
{
public:
    virtual void myname() { cout << "This is class Base" << endl; }
};
```

Від цього класу треба створити два похідних класа: *DerA* і *DerB*.

Від класів *DerA* і *DerB* шляхом множинного успадкування створити клас *DerAB*.

Перевизначити у кожному з створених класів функцію *myname()* таким чином, щоб вона виводила на екран дійсне ім'я класу об'єкту, для якого вона викликається.

Для кожного з класів створеної ієрархії створити по одному об'єкту і для кожного з них викликати функцію *myname()*.

Створити масив із 4 покажчиків на базовий клас ієрархії. Для кожного з них створити динамічний об'єкт, по одному для кожного з класів ієрархії. Зробити виклик функції *myname()* для кожного з динамічних об'єктів за допомогою покажчиків.

Варіант 2. Створити абстрактний клас для роботи з геометричними фігурами на екрані. Передбачити такі компоненти–властивості класу: координати центру фігури; кут повороту (у градусах); масштабний фактор; і такі функції–методи: показати фігуру на екрані - відобразити центр фігури; повернути фігуру на заданий кут - вивести нові координати (кут надається у градусах); пересунути фігуру на наданий вектор - вивести нові координати.

Застосовуючи успадкування і наведений вище абстрактний клас створити похідний клас для роботи з фігурою типу "трикутник". Визначити інтерфейсну частину у класах, застосувати атрибути доступу.

Варіант 3. Для класу «Геометрична фігура» з варіанту 2, створити похідний клас для роботи з фігурою типу «пряма». Визначити інтерфейсну частину у класах, застосувати атрибути доступу.

Варіант 4. Для класу «Геометрична фігура» з варіанту 2, створити похідний клас для роботи з фігурою типу «коло». Визначити інтерфейсну частину у класах, застосувати атрибути доступу.

Варіант 5. Для класу «Геометрична фігура» з варіанту 2, створити похідний клас для роботи з фігурою типу «прямокутник». Визначити інтерфейсну частину у класах, застосувати атрибути доступу.

Контрольні запитання

1. В чому різниця між раннім та пізнім зв’язуванням?
2. Що таке та для чого використовується віртуальна функція?
3. Чи можливо перевантажувати віртуальні функції?
4. Чи повинні співпадати прототипи віртуальних функцій?
5. Коли використовують абстрактні класи?
6. Чи можливо створювати об’єкти для абстрактних класів?
7. Чи може використовуватися покажчик на базовий клас при роботі з об’єктами похідних класів?
8. Чи можна отримати через покажчик на базовий клас доступ до унікальних членів пожідних класів?

ЛАБОРАТОРНА РОБОТА №7
ВВЕДЕННЯ В УЗАГАЛЬНЕНЕ ПРОГРАМУВАННЯ

Мета роботи

Навчитись використовувати шаблони при розробці програм.

Порядок виконання лабораторної роботи

1. Застосувавши конспект лекцій та додаткову літературу, вивчити основні принципи узагальненого програмування, та ознайомитись з особливостями використання стандартної бібліотеки шаблонів (Standard Template Library).
2. Виконати індивідуальні завдання відповідно до номера варіанта, застосувавши мову програмування C++.
3. Оформіти звіт.
4. Зробити висновки.
5. Відповісти на контрольні запитання.

Індивідуальне завдання

Незалежно від варіанту виконати всі три завдання.

1. Виконати завдання з лабораторної роботи № 1, де тип елементу заданої структури даних довільний. Використати шаблонні функції.
2. Виконати завдання з лабораторної роботи № 5 з довільним типом даних. Використати шаблонні класи.
3. Виконати завдання з лабораторної роботи № 1 використавши, для зберігання даних класи Standard Template Library (*STL*) або *list*, або *vector*. Поясніть різницю у використанні цих класів.

Контрольні запитання

1. Що таке шаблон?
2. Скільки аргументів у шаблонних функцій?
3. Скільки раз компілюється шаблонна функція?
4. Чи можливо, щоб шаблонна функція мала аргументи за замовчанням?
5. Назвіть правила визначення шаблонних класів.
6. Чи може шаблонний клас мати аргументи за замовченням?

7. Вкажіть правила оголошення об’єктів шаблонних класів.
8. З чого складається стандартна бібліотека шаблонів?
9. На які два типи розподіляються контейнери бібліотеки STL?
10. Що таке ітератори? Коли та для чого вони використовуються?
11. Коли використовуються алгоритми?

ЛАБОРАТОРНА РОБОТА №8
ОБРОБКА ВИНЯТКОВИХ СИТУАЦІЙ

Мета роботи

Навчитись обробляти виняткові ситуації засобами мови С++.

Порядок виконання лабораторної роботи

1. Застосувавши конспект лекцій та додаткову літературу, вивчити основні принципи обробки помилок (надзвичайних ситуацій).
2. Виконати індивідуальні завдання відповідно до номера варіанта, застосувавши мову програмування С++.
3. Створити базовий клас *Exception*, та відповідні класи-спадкоємці, що дозволяють обробляти наступні виняткові ситуації:
   а) помилки при роботі з потоками введення/виведення, зокрема при роботі з файлами;
   б) помилки арифметичних операцій (ділення на 0);
   в) помилки виділення динамічної пам’яті при перевантаженні операторів new та delete.
4. Додати класи до програми, та продемонструвати обробку виняткових ситуацій.
6. Зробити висновки.
7. Відповісти на контрольні запитання.

Індивідуальне завдання

Для завдання з лабораторної роботи № 5 виконати обробку виняткових ситуацій з використанням класу *Exception*.

Контрольні запитання

1. Які варіанти обробки помилок, що не стосуються виняткових ситуацій, існують?.
2. Які стандартні методи обробки виняткових ситуацій Ви знаєте та використовуєте?
3. Що означає «виняток» та як він утворюється?
4. Що робить оператор catch()? Які форми запису catch() Ви знаєте?
5. Як генерується виняткова ситуація?

6. Які типи виняткових ситуацій можна використовувати? Чи можна використовувати типи, визначені користувачем?

7. Для чого використовується оператор try?

8. Як обробляється помилка виділення пам’яті?

9. Чи можна переривати роботу деструктора через використання обробки виняткових ситуацій?

10. Коли виникає необхідність обробки виняткових ситуацій в конструкторах?

ЛІТЕРАТУРА

1. Schildt Herbert, C++: The Complete Reference. Fourth Edition / Herbert Schildt – McGraw-Hill,Osborne, 2003, – 1023 p.

2. C++. Основи програмування. Теорія та практика : підручник / [О.Г. Трофименко, Ю.В. Прокоп, І.Г. Швайко, Л.М. Буката та ін.] ; за ред. О.Г. Трофименко. – Одеса: Фенікс, 2010. – 544 с.

3. Learning C++ [Electronic resource]// RIP Tutorial. - 2019 - 897 p. - Access mode: https://www.dbooks.org/learning-c-5605981311/

4. C++ language documentation – Access mode: https://learn.microsoft.com/uk-ua/cpp/cpp/?view=msvc-170

5. Табунщик, Г.В. Методичні вказівки до практичних робіт з дисципліни “Об’єктно-орієнтоване програмування” для студентів професійного напряму 6.050101 «Комп’ютерні науки» всіх форм навчання [Текст] / Г.В. Табунщик, Н.О. Миронова. – Запоріжжя: ЗНТУ, 2009. – 32 с.