Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

Об’єктно-орієнтоване програмування

Методичні вказівки до виконання лабораторних робіт
для студентів денної форми навчання за спеціальностями 123 “Комп’ютерна інженерія”, 122 “Комп’ютерні науки”, 125 “Кібербезпека”

ЗАТВЕРДЖЕНО
на засіданні кафедри кібербезпеки та
програмного забезпечення, протокол №1
від 28.08.2023.

Кропивницький
2023

Об’єктно-орієнтоване програмування: методичні вказівки до виконання лабораторних робіт для студентів за спеціальностями 123 “Комп’ютерна інженерія”, 122 “Комп’ютерні науки”, 125 “Кібербезпека”/ М-во освіти і науки України, Центральноукр. нац. техн. ун-т; [уклад. П. С. Усік] – Кропивницький: ЦНТУ, 2023. – 105 с.

Укладачі: Усік П. С., доктор філософії, викладач;

Рецензенти: Смірнов О. А., докт. техн. наук, професор;
Коваленко О. В., докт. техн. наук, професор.

ЗМІСТ

ВСТУП

Мета: Метою викладання дисципліни «Об’єктно-орієнтованого програмування» є ознайомлення з ключовими концепціями об'єктно-орієнтованого програмування та навчання навичкам розробки програм за допомогою мови програмування C++. Вивчення структури класів, взаємодії об'єктів, спадкування та поліморфізму. Набуття практичного досвіду вирішення завдань програмування з використанням об'єктно-орієнтованого підходу на базі мови C++.

Завдання:

– Вивчення теоретичних основ об’єктно-орієнтованого програмування.

– Набути практичних навичок реалізації об’єктно-орієнтованого програмування мовою програмування C++.

У результаті вивчення навчальної дисципліни студент повинен:

– знати новітні технології в галузі інформаційних технологій.

Використовувати сучасні методи і мови програмування для розроблення алгоритмічного та програмного забезпечення. Визначати критерії, яким повинен задовольняти проект, щоб його легко було супроводжувати і модифікувати. Створювати системне та прикладне програмне забезпечення комп’ютерних систем та мереж. Використовувати класи-шаблони стандартної бібліотеки мови C ++ (STL) та узагальнені алгоритми при написанні програм.) та узагальнені алгоритми при написанні програм. Застосовувати шаблони (патерни). Використовувати засоби і системи автоматизації проектування до розроблення компонентів комп'ютерних систем та мереж. Інтернет додатків, кіберфізичних систем, тощо. Впевнено створювати додатки в середовищах швидкої розробки(QtCreator, MVS, MatL) та узагальнені алгоритми при написанні програм.ab(FreeMat) т.і.). Прогамувати ПЗ для мікропроцесорних роботизованих систем мовою C++. Усвідомлювати необхідність навчання впродовж усього життя з метою поглиблення набутих та здобуття нових фахових знань, удосконалення креативного мислення.

– вміти застосовувати знання для ідентифікації, формулювання і розв’язування технічних задач спеціальності, використовуючи методи, що є найбільш придатними для досягнення поставлених цілей. Розв’язувати задачі аналізу та синтезу засобів, характерних для спеціальності. Адаптуватись до нових ситуацій, обґрунтовувати, приймати та реалізовувати у межах компетенції рішення.

Структурно логічна схема підготовки бакалавра.

Враховуючи послідовність накопичення знань та інформації, дисципліна вивчається після викладання наступних дисциплін: «Базові методології та технології програмування».

Для опанування матеріалу дисципліни «Об’єктно-орієнтованого програмування» окрім лекційних та лабораторних занять, тобто аудиторного навантаження, значна увага приділяється самостійній роботі.

До основних видів самостійної роботи студента відносимо:

1. Вивчення лекційного матеріалу.

2. Робота з літературними джерелами.

3. Розв’язання практичних задач.

4. Підготовка до модульних, підсумкового контролю, екзамену (денна) та заліку (заочна).

5. Виконання контрольної роботи для заочної форми навчання.

В ході викладання дисципліни викладачем застосовуються види занять, які згідно з програмою навчальної дисципліни передбачають лекційні, та лабораторні заняття, а також виконання самостійної роботи.

Основна мета лекції – дати систематизовані основи знань з навчальної дисципліни, зосередити увагу студентів на найбільш складних та ключових питаннях.

Основна мета лабораторної роботи – закріплення й деталізація знань, а головне – формування навичок і вмінь.

Шкала оцінювання: національна та ЄКТС

Сума балів за всі види навчальної діяльності	Оцінка ЄКТС	Оцінка за національною шкалою
		для екзамену
90-100	A	відмінно
82-89	B	добре
74-81	C	
64-73	D	задовільно
60-63	E	
35-59	FX	незадовільно з можливістю повторного складання
1-34	F	незадовільно з обов’язковим повторним вивченням дисципліни

Ознайомившись з теоретичним матеріалом, ви повинні виконати завдання до лабораторних робіт. Звіт повинен містити хід виконання завдань, програмну реалізацію, а також графічні матеріали, що підтверджують виконання цих завдань.

Лабораторна робота №1 Основні поняття ООП. Класи та об’єкти. Функції доступу. Вказівник this

Мета: ознайомитись з основними поняттями ООП. Вивчити поняття клас, об’єкт, сеттер, геттер та навчитись їх програмно реалізовувати мовою С++.

Теоретична частина

Клас - це конструкція в ООП, яка представляє собою шаблон або опис структури та поведінки об'єктів. Клас є абстракцією, яка групує дані (властивості) та функції (методи), що пов'язані між собою логічно. Він визначає набір атрибутів (члени класу), які представляють стан об'єктів даного класу, а також методи, які визначають операції, які можуть бути виконані над цими об'єктами.

Класи визначаються за допомогою ключового слова class та мають ім'я, за допомогою якого їх можна ідентифікувати. Клас може мати конструктори, які використовуються для ініціалізації об'єктів класу, а також може містити статичні поля та методи, які використовуються для спільного використання даних між усіма об'єктами даного класу.

Об'єкт - це конкретний екземпляр класу. Він створюється на основі класу шляхом процесу, який називається інстанціюванням або створенням об'єкта. Кожен об'єкт має свій власний набір значень властивостей, які відображають його стан, та може виконувати методи, що визначені в класі.

Об'єкти зберігають свій внутрішній стан у вигляді значень властивостей, які можуть бути унікальними для кожного об'єкта. Методи об'єктів представляють собою операції або функції, які можуть бути виконані над цими об'єктами. Об'єкти взаємодіють один з одним шляхом виклику методів та обміну даними.

Основна ідея ООП полягає в організації програми як набору взаємодіючих об'єктів, які співпрацюють для вирішення певних задач. Класи визначають загальну структуру та поведінку, а об'єкти є конкретними представниками цих класів, здатними зберігати стан та виконувати дії, визначені класом.

Наприклад:

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

// Клас "Автомобіль"
class Car {
public:
    string brand;
    string model;
    int year;

    void startEngine() {
        cout << "Двигун автомобіля запущено!" << endl;
    }
```

```
    void stopEngine() {
        cout << "Двигун автомобіля зупинено!" << endl;
    }
};

int main() {
    // Створення об'єкта класу "Автомобіль"
    Car myCar;

    // Налаштування властивостей об'єкта
    myCar.brand = "BMW";
    myCar.model = "X5";
    myCar.year = 2022;

    // Виклик методів об'єкта
    cout << "Марка автомобіля: " << myCar.brand << endl;
    cout << "Модель автомобіля: " << myCar.model << endl;
    cout << "Рік випуску автомобіля: " << myCar.year << endl;

    myCar.startEngine(); // Запуск двигуна
    myCar.stopEngine(); // Зупинка двигуна

    return 0;
}
```

У цьому прикладі ми оголосили клас "Car", який представляє автомобіль. Він має властивості brand, model та year, а також методи startEngine() та stopEngine(), які відповідають за запуск та зупинку двигуна.

У функції main() ми створюємо об'єкт myCar класу Car. Потім ми налаштовуємо властивості об'єкта, використовуючи оператор крапки (.), і викликаємо методи об'єкта.

В результаті на консоль будуть виведені властивості автомобіля (марка, модель, рік випуску), а також повідомлення про запуск та зупинку двигуна.

Функції доступу (сеттери та геттери) є методами, використовуваними в об'єктно-орієнтованому програмуванні (ООП) для отримання (читання) та встановлення (запису) значень приватних членів класу. Ці методи надають контроль доступу до даних класу та допомагають забезпечити принцип інкапсуляції, що означає, що дані класу знаходяться внутрішньо і мають обмежений доступ ззовні.

Геттери (getter methods) - це методи, які використовуються для отримання (читання) значень приватних членів класу. Вони зазвичай повертають значення цих членів і мають повернутий тип, що відповідає типу члена класу. Геттери дозволяють зовнішньому коду отримувати доступ до значень приватних даних класу без безпосереднього доступу до них. Вони можуть бути корисними для отримання значень для подальшого використання або виведення на екран.

Наприклад, у класі "Person" може бути приватний член name, і для отримання цього значення може бути створений геттер:

```
class Person {
private:
    string name;

public:
    // Геттер для отримання значення name
    string getName() {
        return name;
    }
};
```

Сеттери (setter methods) - це методи, які використовуються для встановлення (запису) значень приватних членів класу. Вони приймають параметр з новим значенням і встановлюють це значення для відповідного приватного члена класу. Сеттери дозволяють контролювати доступ до приватних даних класу і забезпечувати перевірку або обробку значень перед їх збереженням.

Продовжуючи приклад з класом "Person", може бути створений сеттер для встановлення нового значення для name:

```
class Person {
private:
    string name;

public:
    // Геттер для отримання значення name
    string getName() {
        return name;
    }

    // Сеттер для встановлення значення name
    void setName(string newName) {
        name = newName;
    }
};
```

За допомогою сеттера зовнішній код може встановити нове значення для name, і внутрішній код класу може здійснити додаткову перевірку або обробку цього значення перед збереженням.

Використання геттерів та сеттерів дозволяє забезпечити кращий контроль над доступом до приватних даних класу, зберігаючи їх недоступними безпосередньо ззовні класу і забезпечуючи правильність операцій з цими даними.

Ключове слово this в об'єктно-орієнтованому програмуванні (ООП) вказує на поточний об'єкт, з яким пов'язаний виклик методу або доступ до членів

класу. Воно представляє вказівник на сам об'єкт, на якому викликається метод або з якого доступ до членів класу.

Ключове слово this може використовуватись в контексті класу для посилання на його власні члени (змінні та методи). Це особливо корисно, коли ім'я параметра методу або локальної змінної конфліктує з іменем члена класу. Використання this дозволяє уникнути неоднозначностей і вказати на конкретний член класу.

Наприклад, розглянемо клас "Person" з методом "setName", який встановлює значення приватного члена "name":

```
class Person {
private:
    string name;

public:
    void setName(string name) {
        this->name = name;
    }
};
```

У цьому прикладі параметр методу "setName" має таке ж ім'я, як і приватний член "name". Використання this->name дозволяє чітко вказати, що ми звертаємось саме до члена класу "name", а не до параметра методу.

Крім того, ключове слово this може бути корисним при передачі посилання на поточний об'єкт як аргумент у виклику методу або при створенні ланцюжка методів.

Загалом, this використовується для доступу до членів класу зсередини самого класу і допомагає уникнути конфліктів ім'ен, які можуть виникати між параметрами методів та членами класу.

Завдання

Варіант 1

1. Напишіть клас "Book" для представлення книги. Клас повинен мати наступні властивості та функціональність:

Приватні поля класу:

- title (назва книги)
- author (автор книги)
- year (рік видання книги)

Публічні методи класу:

- Метод setTitle(), який дозволяє задати назву книги.
- Метод getTitle(), який повертає назву книги.
- Метод setAuthor(), який дозволяє задати автора книги.

- Метод getAuthor(), який повертає автора книги.
- Метод setYear(), який дозволяє задати рік видання книги.
- Метод getYear(), який повертає рік видання книги.

2. Створіть об'єкт класу "Book".

3. Задайте значення полів об'єкта за допомогою відповідних методів.

4. Виведіть інформацію про книгу на екран, використовуючи методи доступу до полів.

5. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 2

1. Напишіть клас "Rectangle" для представлення прямокутника. Клас повинен мати наступні властивості та функціональність:

Приватні поля класу:

- width (ширина прямокутника)
- height (висота прямокутника)

Публічні методи класу:

- Метод setWidth(), який дозволяє задати значення ширини прямокутника.
- Метод setHeight(), який дозволяє задати значення висоти прямокутника.
- Метод getArea(), який повертає площу прямокутника (обчислюється як добуток ширини та висоти).
- Метод getPerimeter(), який повертає периметр прямокутника (обчислюється як сума усіх сторін прямокутника).

2. Створіть об'єкт класу "Rectangle".

3. Задайте значення ширини та висоти прямокутника.

4. Викличте методи getArea() та getPerimeter() для отримання площі та периметру прямокутника.

5. Виведіть результати на екран.

6. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 3

1. Напишіть клас "BankAccount" для керування банківським рахунком. Клас повинен мати наступні властивості та функціональність:

Приватні поля класу:

- balance (баланс рахунку, типу double)
- accountNumber (номер рахунку, типу string)

- ownerName (ім'я власника рахунку, типу string)

Публічні методи класу:

- Метод setBalance(), який дозволяє задати початковий баланс рахунку.
- Метод getBalance(), який повертає поточний баланс рахунку.
- Метод setAccountNumber(), який дозволяє задати номер рахунку.
- Метод getAccountNumber(), який повертає номер рахунку.
- Метод setOwnerName(), який дозволяє задати ім'я власника рахунку.
- Метод getOwnerName(), який повертає ім'я власника рахунку.
- Метод deposit(), який дозволяє здійснити поповнення рахунку на певну суму.
- Метод withdraw(), який дозволяє зняти гроші з рахунку на певну суму.

2. Створіть об'єкт класу "BankAccount".
3. Задайте значення полів об'єкта, включаючи баланс рахунку, номер рахунку та ім'я власника.
4. Використовуйте методи для отримання та зміни значень полів об'єкта.
5. Викличте методи для поповнення та зняття грошей з рахунку.
6. Виведіть інформацію про рахунок на екран.
7. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 4

1. Створіть клас "Person" (людина), який має наступні властивості та функціональність:

Приватні поля класу:

- name (ім'я людини)
- age (вік людини)
- address (адреса людини)

Публічні методи класу:

- Метод setName(), який дозволяє задати ім'я людини.
- Метод getName(), який повертає ім'я людини.
- Метод setAge(), який дозволяє задати вік людини.
- Метод getAge(), який повертає вік людини.
- Метод setAddress(), який дозволяє задати адресу людини.
- Метод getAddress(), який повертає адресу людини.

2. Створіть об'єкт класу "Person".
3. Задайте значення полів об'єкта за допомогою відповідних методів.

4. Виведіть інформацію про людину на екран, використовуючи методи для отримання значень полів.

5. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 5

1. Створіть клас "Employee" (співробітник), який має наступні властивості та функціональність:

Приватні поля класу:

- name (ім'я співробітника)
- id (ідентифікатор співробітника)
- salary (заробітна плата співробітника)

Публічні методи класу:

- Метод setName(), який дозволяє задати ім'я співробітника.
- Метод getName(), який повертає ім'я співробітника.
- Метод setId(), який дозволяє задати ідентифікатор співробітника.
- Метод getId(), який повертає ідентифікатор співробітника.
- Метод setSalary(), який дозволяє задати заробітну плату співробітника.
- Метод getSalary(), який повертає заробітну плату співробітника.

2. Створіть об'єкт класу "Employee".

3. Задайте значення полів об'єкта за допомогою відповідних методів.

4. Виведіть інформацію про співробітника на екран, використовуючи методи для отримання значень полів.

5. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 6

1. Створіть клас "Student" (студент), який має наступні властивості та функціональність:

Приватні поля класу:

- name (ім'я студента)
- age (вік студента)
- major (спеціальність студента)

Публічні методи класу:

- Метод setName(), який дозволяє задати ім'я студента.
- Метод getName(), який повертає ім'я студента.
- Метод setAge(), який дозволяє задати вік студента.

- Метод getAge(), який повертає вік студента.
- Метод setMajor(), який дозволяє задати спеціальність студента.
- Метод getMajor(), який повертає спеціальність студента.

2. Створіть об'єкт класу "Student".

3. Задайте значення полів об'єкта за допомогою відповідних методів.

4. Виведіть інформацію про студента на екран, використовуючи методи для отримання значень полів.

5. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 7

1. Створіть клас "Animal" (тварина), який має наступні властивості та функціональність:

Приватні поля класу:

- name (ім'я тварини)
- species (вид тварини)
- age (вік тварини)

Публічні методи класу:

- Метод setName(), який дозволяє задати ім'я тварини.
- Метод getName(), який повертає ім'я тварини.
- Метод setSpecies(), який дозволяє задати вид тварини.
- Метод getSpecies(), який повертає вид тварини.
- Метод setAge(), який дозволяє задати вік тварини.
- Метод getAge(), який повертає вік тварини.

2. Створіть об'єкт класу "Animal".

3. Використайте метод setName() для задання імені тварини.

4. Використайте метод getName() для отримання імені тварини та виведіть його на екран.

5. Використайте метод setSpecies() для задання виду тварини.

6. Використайте метод getSpecies() для отримання виду тварини та виведіть його на екран.

7. Використайте метод setAge() для задання віку тварини. Використайте метод getAge() для отримання віку тварини та виведіть його на екран.

8. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними

Варіант 8

1. Створіть клас "FootballPlayer" (футболіст), який має наступні властивості та функціональність:

Приватні поля класу:

- name (ім'я футболіста)
- age (вік футболіста)
- position (позиція на полі)

Публічні методи класу:

- Метод setName(), який дозволяє задати ім'я футболіста.
- Метод getName(), який повертає ім'я футболіста.
- Метод setAge(), який дозволяє задати вік футболіста.
- Метод getAge(), який повертає вік футболіста.
- Метод setPosition(), який дозволяє задати позицію футболіста.
- Метод getPosition(), який повертає позицію футболіста.

2. Створіть об'єкт класу "FootballPlayer".
3. Використайте метод setName() для задання імені футболіста.
4. Використайте метод getName() для отримання імені футболіста та виведіть його на екран.
5. Використайте метод setAge() для задання віку футболіста.
6. Використайте метод getAge() для отримання віку футболіста та виведіть його на екран.
7. Використайте метод setPosition() для задання позиції футболіста.
8. Використайте метод getPosition() для отримання позиції футболіста та виведіть її на екран.
9. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними

Варіант 9

1. Створіть клас "Movie" (фільм), який має наступні властивості та функціональність:

Приватні поля класу:

- title (назва фільму)
- director (режисер фільму)
- year (рік випуску фільму)
- duration (тривалість фільму в хвилинах)

Публічні методи класу:

- Метод setTitle(), який дозволяє задати назву фільму.
- Метод getTitle(), який повертає назву фільму.

- Метод setDirector(), який дозволяє задати режисера фільму.
- Метод getDirector(), який повертає режисера фільму.
- Метод setYear(), який дозволяє задати рік випуску фільму.
- Метод getYear(), який повертає рік випуску фільму.
- Метод setDuration(), який дозволяє задати тривалість фільму.
- Метод getDuration(), який повертає тривалість фільму.

2. Створіть об'єкт класу "Movie".

3. Використайте метод setTitle() для задання назви фільму.

4. Використайте метод getTitle() для отримання назви фільму та виведіть її на екран.

5. Використайте метод setDirector() для задання режисера фільму.

6. Використайте метод getDirector() для отримання режисера фільму та виведіть його на екран.

7. Використайте метод setYear() для задання року випуску фільму.

8. Використайте метод getYear() для отримання року випуску фільму та виведіть його на екран.

9. Використайте метод setDuration() для задання тривалості фільму.

10. Використайте метод getDuration() для отримання тривалості фільму та виведіть її на екран.

11. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними

Варіант 10

1. Створіть клас "TravelDestination" (туристичний напрямок), який має наступні властивості та функціональність:

Приватні поля класу:

- name (назва туристичного напрямку)
- country (країна, в якій знаходиться напрямок)
- rating (рейтинг популярності напрямку)

Публічні методи класу:

- Метод setName(), який дозволяє задати назву туристичного напрямку.
- Метод getName(), який повертає назву туристичного напрямку.
- Метод setCountry(), який дозволяє задати країну, в якій знаходиться напрямок.
- Метод getCountry(), який повертає країну, в якій знаходиться напрямок.
- Метод setRating(), який дозволяє задати рейтинг популярності напрямку.
- Метод getRating(), який повертає рейтинг популярності напрямку.

2. Створіть об'єкт класу "TravelDestination".

3. Задайте значення полів об'єкту за допомогою відповідних методів.

4. Виведіть інформацію про туристичний напрямок на екран, використовуючи методи getName(), getCountry() та getRating().

5. Змініть значення рейтингу популярності напрямку за допомогою методу setRating().

6. Виведіть оновлену інформацію про туристичний напрямок на екран, використовуючи знову методи getName(), getCountry() та getRating().

7. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними

Варіант 11

1. Створіть клас "Doctor" (лікар), який має наступні властивості та функціональність:

Приватні поля класу:

- name (ім'я лікаря)
- specialization (спеціалізація лікаря, наприклад, "хірург", "кардіолог" і т.д.)
- experience (стаж роботи лікаря в роках)

Публічні методи класу:

- Метод setName(), який дозволяє задати ім'я лікаря.
- Метод getName(), який повертає ім'я лікаря.
- Метод setSpecialization(), який дозволяє задати спеціалізацію лікаря.
- Метод getSpecialization(), який повертає спеціалізацію лікаря.
- Метод setExperience(), який дозволяє задати стаж роботи лікаря.
- Метод getExperience(), який повертає стаж роботи лікаря.

2. Створіть об'єкт класу "Doctor".

3. Використайте метод setName() для задання імені лікаря.

4. Використайте метод getName() для отримання імені лікаря та виведіть його на екран.

5. Використайте метод setSpecialization() для задання спеціалізації лікаря.

6. Використайте метод getSpecialization() для отримання спеціалізації лікаря та виведіть її на екран.

7. Використайте метод setExperience() для задання стажу роботи лікаря.

8. Використайте метод getExperience() для отримання стажу роботи лікаря та виведіть його на екран.

9. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 12

1. Створіть клас "Country" (країна), який має наступні властивості та функціональність:

Приватні поля класу:

- name (назва країни)
- capital (столиця країни)
- population (населення країни)

Публічні методи класу:

- Метод setName(), який дозволяє задати назву країни.
- Метод getName(), який повертає назву країни.
- Метод setCapital(), який дозволяє задати столицю країни.
- Метод getCapital(), який повертає столицю країни.
- Метод setPopulation(), який дозволяє задати населення країни.
- Метод getPopulation(), який повертає населення країни.

2. Створіть об'єкт класу "Country".

3. Використайте метод setName() для задання назви країни.

4. Використайте метод getName() для отримання назви країни та виведіть її на екран.

5. Використайте метод setCapital() для задання столиці країни.

6. Використайте метод getCapital() для отримання столиці країни та виведіть її на екран.

7. Використайте метод setPopulation() для задання населення країни.

8. Використайте метод getPopulation() для отримання населення країни та виведіть його на екран.

9. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 13

1. Створіть клас "Calculator" (калькулятор), який має наступні властивості та функціональність:

Приватні поля класу:

- num1 (перше число)
- num2 (друге число)

Публічні методи класу:

- Метод setNum1(), який дозволяє задати перше число.

- Метод getNum1(), який повертає перше число.
- Метод setNum2(), який дозволяє задати друге число.
- Метод getNum2(), який повертає друге число.
- Метод add(), який повертає суму першого та другого чисел.
- Метод subtract(), який повертає різницю першого та другого чисел.
- Метод multiply(), який повертає добуток першого та другого чисел.
- Метод divide(), який повертає результат ділення першого числа на друге число.

2. Створіть об'єкт класу "Calculator".

3. Використайте метод setNum1() для задання першого числа.

4. Використайте метод getNum1() для отримання першого числа та виведіть його на екран.

5. Використайте метод setNum2() для задання другого числа.

6. Використайте метод getNum2() для отримання другого числа та виведіть його на екран.

7. Використайте метод add() для обчислення суми першого та другого чисел та виведіть результат на екран.

8. Використайте метод subtract() для обчислення різниці першого та другого чисел та виведіть результат на екран.

9. Використайте метод multiply() для обчислення добутку першого та другого чисел та виведіть результат на екран.

10. Використайте метод divide() для обчислення результату ділення першого числа на друге число та виведіть результат на екран.

11. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 14

1. Створіть клас "Company" (компанія), який має наступні властивості та функціональність:

Приватні поля класу:

- name (назва компанії)
- revenue (прибуток компанії)
- expenses (витрати компанії)

Публічні методи класу:

- Метод setName(), який дозволяє задати назву компанії.
- Метод getName(), який повертає назву компанії.
- Метод setRevenue(), який дозволяє задати прибуток компанії.

- Метод getRevenue(), який повертає прибуток компанії.
- Метод setExpenses(), який дозволяє задати витрати компанії.
- Метод getExpenses(), який повертає витрати компанії.
- Метод calculateProfit(), який обчислює прибуток компанії (прибуток = прибуток - витрати).
- Метод getProfit(), який повертає прибуток компанії.

2. Створіть об'єкт класу "Company".

3. Використайте метод setName() для задання назви компанії.

4. Використайте метод getName() для отримання назви компанії та виведіть її на екран.

5. Використайте метод setRevenue() для задання прибутку компанії.

6. Використайте метод getRevenue() для отримання прибутку компанії та виведіть його на екран.

7. Використайте метод setExpenses() для задання витрат компанії.

8. Використайте метод getExpenses() для отримання витрат компанії та виведіть їх на екран.

9. Використайте метод calculateProfit() для обчислення прибутку компанії та виведіть його на екран.

10. Використайте метод getProfit() для отримання прибутку компанії та виведіть його на екран.

11. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 15

1. Створіть клас "Athlete" (спортсмен), який має наступні властивості та функціональність:

Приватні поля класу:

- name (ім'я спортсмена)
- sport (вид спорту, яким займається спортсмен)
- achievements (досягнення спортсмена, представлені у вигляді масиву)

Публічні методи класу:

- Метод setName(), який дозволяє задати ім'я спортсмена.
- Метод getName(), який повертає ім'я спортсмена.
- Метод setSport(), який дозволяє задати вид спорту, яким займається спортсмен.
- Метод getSport(), який повертає вид спорту спортсмена.
- Метод setAchievements(), який дозволяє задати досягнення спортсмена.

- Метод getAchievements(), який повертає досягнення спортсмена.

2. Створіть об'єкт класу "Athlete".

3. Використайте метод setName() для задання імені спортсмена.

4. Використайте метод getName() для отримання імені спортсмена та виведіть його на екран.

5. Використайте метод setSport() для задання виду спорту, яким займається спортсмен.

6. Використайте метод getSport() для отримання виду спорту спортсмена та виведіть його на екран.

7. Використайте метод setAchievements() для задання досягнень спортсмена.

8. Використайте метод getAchievements() для отримання досягнень спортсмена та виведіть їх на екран.

9. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 16

1. Створіть клас "Recipe" (рецепт), який має наступні властивості та функціональність:

Приватні поля класу:

- name (назва рецепту)
- ingredients (складники рецепту, представлені у вигляді масиву)
- instructions (інструкції для приготування рецепту, представлені у вигляді тексту)

Публічні методи класу:

- Метод setName(), який дозволяє задати назву рецепту.
- Метод getName(), який повертає назву рецепту.
- Метод setIngredients(), який дозволяє задати складники рецепту.
- Метод getIngredients(), який повертає складники рецепту.
- Метод setInstructions(), який дозволяє задати інструкції для приготування рецепту.
- Метод getInstructions(), який повертає інструкції для приготування рецепту.

2. Створіть об'єкт класу "Recipe".

3. Використайте метод setName() для задання назви рецепту.

4. Використайте метод getName() для отримання назви рецепту та виведіть її на екран.

5. Використайте метод setIngredients() для задання складників рецепту.

6. Використайте метод getIngredients() для отримання складників рецепту та виведіть їх на екран.

7. Використайте метод setInstructions() для задання інструкцій для приготування рецепту.

8. Використайте метод getInstructions() для отримання інструкцій для приготування рецепту та виведіть їх на екран.

9. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 17

1. Створіть клас "Train" (поїзд), який має наступні властивості та функціональність:

Приватні поля класу:

- trainNumber (номер поїзда)
- origin (місце відправлення поїзда)
- destination (місце призначення поїзда)
- departureTime (час відправлення поїзда)

Публічні методи класу:

- Метод setTrainNumber(), який дозволяє задати номер поїзда.
- Метод getTrainNumber(), який повертає номер поїзда.
- Метод setOrigin(), який дозволяє задати місце відправлення поїзда.
- Метод getOrigin(), який повертає місце відправлення поїзда.
- Метод setDestination(), який дозволяє задати місце призначення поїзда.
- Метод getDestination(), який повертає місце призначення поїзда.
- Метод setDepartureTime(), який дозволяє задати час відправлення поїзда.
- Метод getDepartureTime(), який повертає час відправлення поїзда.

2. Створіть об'єкт класу "Train".

3. Використайте метод setTrainNumber() для задання номера поїзда.

4. Використайте метод getTrainNumber() для отримання номера поїзда та виведіть його на екран.

5. Використайте метод setOrigin() для задання місця відправлення поїзда.

6. Використайте метод getOrigin() для отримання місця відправлення поїзда та виведіть його на екран.

7. Використайте метод setDestination() для задання місця призначення поїзда.

8. Використайте метод getDestination() для отримання місця призначення поїзда та виведіть його на екран.

9. Використайте метод setDepartureTime() для задання часу відправлення поїзда.

10. Використайте метод getDepartureTime() для отримання часу відправлення поїзда та виведіть його на екран.

11. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 18

1. Створіть клас "Weather" (погода), який має наступні властивості та функціональність:

Приватні поля класу:

- temperature (температура повітря в градусах Цельсія)
- humidity (вологість повітря в відсотках)
- precipitation (тип та інтенсивність опадів)

Публічні методи класу:

- Метод setTemperature(), який дозволяє задати температуру повітря.
- Метод getTemperature(), який повертає температуру повітря.
- Метод setHumidity(), який дозволяє задати вологість повітря.
- Метод getHumidity(), який повертає вологість повітря.
- Метод setPrecipitation(), який дозволяє задати тип та інтенсивність опадів.
- Метод getPrecipitation(), який повертає тип та інтенсивність опадів.

2. Створіть об'єкт класу "Weather".

3. Використайте метод setTemperature() для задання температури повітря.

4. Використайте метод getTemperature() для отримання температури повітря та виведіть її на екран.

5. Використайте метод setHumidity() для задання вологості повітря.

6. Використайте метод getHumidity() для отримання вологості повітря та виведіть її на екран.

7. Використайте метод setPrecipitation() для задання типу та інтенсивності опадів.

8. Використайте метод getPrecipitation() для отримання типу та інтенсивності опадів та виведіть їх на екран.

9. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 19

1. Створіть клас "Factory" (завод), який має наступні властивості та функціональність:

Приватні поля класу:

- factoryName (назва заводу)
- location (розташування заводу)
- productionCapacity (продуктивність заводу в одиницях виробництва)

Публічні методи класу:

- Метод setFactoryName(), який дозволяє задати назву заводу.
- Метод getFactoryName(), який повертає назву заводу.
- Метод setLocation(), який дозволяє задати розташування заводу.
- Метод getLocation(), який повертає розташування заводу.
- Метод setProductionCapacity(), який дозволяє задати продуктивність заводу.
- Метод getProductionCapacity(), який повертає продуктивність заводу.

2. Створіть об'єкт класу "Factory".

3. Використайте метод setFactoryName() для задання назви заводу.

4. Використайте метод getFactoryName() для отримання назви заводу та виведіть її на екран.

5. Використайте метод setLocation() для задання розташування заводу.

6. Використайте метод getLocation() для отримання розташування заводу та виведіть його на екран.

7. Використайте метод setProductionCapacity() для задання продуктивності заводу.

8. Використайте метод getProductionCapacity() для отримання продуктивності заводу та виведіть її на екран.

9. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Варіант 20

1. Створіть клас "Circle" (круг), який має наступні властивості та функціональність:

Приватні поля класу:

- radius (радіус круга)

Публічні методи класу:

- Метод setRadius(), який дозволяє задати радіус круга.
- Метод getRadius(), який повертає радіус круга.

- Метод calculateArea(), який обчислює площу круга.
- Метод calculateCircumference(), який обчислює довжину кола.

2. Створіть об'єкт класу "Circle".

3. Використайте метод setRadius() для задання радіуса круга.

4. Використайте метод getRadius() для отримання радіуса круга та виведіть його на екран.

5. Використайте метод calculateArea() для обчислення площі круга.

6. Виведіть обчислену площу круга на екран.

7. Використайте метод calculateCircumference() для обчислення довжини кола.

8. Виведіть обчислену довжину кола на екран.

9. Реалізувати програму за допомогою роздільної компіляції.

У вашому рішенні можуть бути додаткові методи та поля, якщо ви вважаєте їх необхідними.

Лабораторна робота №2 Конструктори і деструктори

Мета: ознайомитись з основними поняттями конструктор і деструктор в ООП та навчитись їх програмно реалізовувати мовою С++.

Теоретична частина

Конструктор є спеціальним методом в класі, який викликається автоматично при створенні нового об'єкта цього класу. Його основна функція полягає в ініціалізації об'єкта, тобто встановленні його початкового стану.

У мові С++, конструктор має той самий ім'я, що й клас, і не має повертаючого значення (навіть не вказується тип void). Конструктор може мати параметри, але також може бути й без параметрів. Зазвичай використовуються параметри конструктора для передачі значень до членів даних об'єкта.

Основні види конструкторів:

- **Конструктор за замовчуванням (default constructor):** Це конструктор без параметрів, який викликається при створенні об'єкта без передачі яких-небудь значень. Він може ініціалізувати члени даних об'єкта значеннями за замовчуванням.

Наприклад:

```
class MyClass {
public:
    MyClass() {
        // Конструктор за замовчуванням
        // Виконання початкових операцій
    }
};
```

- **Параметризований конструктор (parameterized constructor):** Це конструктор, який приймає параметри і використовує їх для ініціалізації членів даних об'єкта.

Наприклад:

```
class Point {
private:
    int x, y;
public:
    Point(int xCoord, int yCoord) {
        x = xCoord;
        y = yCoord;
    }
};
```

- **Конструктор копіювання (copy constructor):** Це конструктор, який приймає об'єкт того ж класу як параметр і створює новий об'єкт, який є копією цього переданого об'єкта. Конструктор копіювання використовується, коли

потрібно зробити копію об'єкта для подальшої роботи з ним, а не просто посилання на нього.

Наприклад:

```
class MyClass {
public:
    MyClass(const MyClass& other) {
        // Конструктор копіювання
        // Виконання операцій копіювання
    }
};
```

Конструктори дозволяють забезпечити коректну ініціалізацію об'єктів і можуть бути потужним інструментом у процесі програмування на C++.

Деструктор є спеціальним методом в класі, який викликається автоматично під час звільнення пам'яті, знищення об'єкта або виходу з блоку, в якому об'єкт був створений. Він використовується для виконання ресурсоозберігаючих операцій або очищення пам'яті, які можуть бути пов'язані з об'єктом.

У мові C++, деструктор має таку ж назву, як і клас, до якого він належить, перед якою ставиться символ тильда (~). Наприклад, якщо у нас є клас під назвою "MyClass", то його деструктор буде мати назву "~MyClass".

Деструктор використовується для звільнення ресурсів, таких як пам'ять, виділену динамічно, або закриття файлів, відкритих об'єктом класу. Це може бути корисно, наприклад, для видалення об'єктів, які були створені в конструкторі, або для закриття відкритих з'єднань з базами даних.

Деструктор викликається автоматично безпосередньо перед знищенням об'єкта. Він може бути визначений в класі та містити будь-який код, необхідний для очищення ресурсів. Якщо деструктор не визначений в класі, то компілятор створює деструктор за замовчуванням, який не виконує ніяких додаткових дій.

Особливістю деструктора є його автоматичне викликання, що дозволяє забезпечити коректне звільнення ресурсів та уникнути витоку пам'яті або інших проблем, пов'язаних з неочищенням ресурсів, коли об'єкт виходить з області видимості.

Приклад використання:

```
#include <iostream>

class MyClass {
private:
    int data;

public:
    // Конструктор за замовчуванням
    MyClass() {
        data = 0;
```

```
            std::cout << "Конструктор за замовчуванням викликаний!"
<< std::endl;
        }

        // Конструктор з параметрами
        MyClass(int value) {
            data = value;
            std::cout << "Конструктор з параметрами викликаний!" <<
std::endl;
        }

        // Копіюючий конструктор
        MyClass(const MyClass& other) {
            data = other.data;
            std::cout  <<  "Копіюючий  конструктор  викликаний!"  <<
std::endl;
        }

        // Деструктор
        ~MyClass() {
            std::cout << "Деструктор викликаний!" << std::endl;
        }

        // Метод для отримання значення даних
        int getData() {
            return data;
        }
    };

    int main() {
        // Виклик конструкторів та деструкторів

        // Об'єкт створений за замовчуванням
        MyClass obj1;
        std::cout << "obj1.data: " << obj1.getData() << std::endl;

        // Об'єкт створений з параметрами
        MyClass obj2(10);
        std::cout << "obj2.data: " << obj2.getData() << std::endl;

        // Копіюючий конструктор
        MyClass obj3 = obj2;
        std::cout << "obj3.data: " << obj3.getData() << std::endl;

        return 0;
    }
```

Завдання

1. Реалізуйте конструктор за замовчуванням, конструктор з параметрами та копіюючий конструктор для вашого класу з лабораторної роботи 1.

- Конструктор за замовчуванням має встановлювати значення полів за замовчуванням.
- Конструктор з параметрами має приймати значення для кожного поля.
- Копіюючий конструктор має копіювати значення полів з іншого об'єкта класу.

2. Реалізуйте деструктор для класу. Деструктор має виводити повідомлення про знищення об'єкта.
3. У функції main створіть об'єкт за допомогою конструктора за замовчуванням та виведіть значення його полів.
4. Створіть новий об'єкт за допомогою конструктора з параметрами та встановіть значення для полів. Виведіть значення полів цього об'єкта.
5. Створіть ще один об'єкт і скопіюйте значення полів з першого об'єкта за допомогою копіюючого конструктора. Виведіть значення полів цього об'єкта.
6. Завершіть функцію main, що призведе до виходу з області видимості створених об'єктів і виклику їх деструкторів. Переконайтесь, що повідомлення про знищення об'єктів виводяться.

Лабораторна робота №3 Успадкування, агрегація та композиція

Мета: ознайомитись з основними поняттями успадкування, агрегація та композиція в ООП та навчитись їх програмно реалізовувати мовою С++.

Теоретична частина

Агрегація є одним з важливих концепцій в ООП і відноситься до відношення між класами, де один клас представляє контейнер, що містить об'єкти іншого класу. У випадку агрегації, об'єкти-члени, які належать до контейнера, можуть існувати самостійно і не залежати від контейнера.

Зазвичай агрегація відбувається за допомогою включення (composition) або посилання (reference). У випадку включення, контейнер містить об'єкти-члени як свої прямі члени (наприклад, об'єкти класів або структур), і вони створюються та знищуються разом з контейнером. У випадку посилання, контейнер містить посилання або вказівники на об'єкти-члени, і вони можуть існувати незалежно від контейнера.

Наприклад, давайте розглянемо клас "Автомобіль" і клас "Двигун". У цьому випадку, клас "Автомобіль" може мати агрегаційне відношення з класом "Двигун". Автомобіль може містити об'єкт "Двигун" як свій прямий член, що означає, що двигун створюється та знищується разом з автомобілем. Або ж клас "Автомобіль" може містити посилання на об'єкт "Двигун", що дозволяє розділяти двигун між декількома автомобілями.

Агрегація дозволяє створювати більш складні структури і групувати пов'язані об'єкти разом для зручної організації та керування. Вона також сприяє розподілу відповідальностей і забезпечує зв'язок між класами.

Наприклад:

```
#include <iostream>

// Клас "Двигун"
class Engine {
public:
    void start() {
        std::cout << "Engine started!" << std::endl;
    }

    void stop() {
        std::cout << "Engine stopped!" << std::endl;
    }
};

// Клас "Автомобіль"
class Car {
private:
    Engine engine;   // Об'єкт класу "Двигун" як член класу
"Автомобіль"

public:
    void startCar() {
```

```
        engine.start();
    }

    void stopCar() {
        engine.stop();
    }
};

int main() {
    Car myCar;
    myCar.startCar();
    myCar.stopCar();

    return 0;
}
```

У цьому прикладі ми маємо клас "Двигун", який має методи start() і stop() для запуску та зупинки двигуна відповідно. Клас "Автомобіль" агрегує об'єкт класу "Двигун" як свій член.

У головній функції ми створюємо об'єкт "Автомобіль" з назвою myCar. Викликаючи метод startCar(), ми викликаємо метод start() об'єкта engine, який запускає двигун. Потім, викликаючи метод stopCar(), ми викликаємо метод stop() об'єкта engine, який зупиняє двигун.

Композиція - це тип відношення між класами, коли один клас (відомий як контейнер або композит) містить об'єкти іншого класу (відомі як компоненти). У випадку композиції, компоненти не можуть існувати без контейнера, тобто вони залежать від його існування і життєвого циклу.

Основна відмінність між агрегацією і композицією полягає в тому, що в композиції компоненти є прямими членами контейнера і створюються та знищуються разом з ним. Це означає, що коли контейнер створюється або знищується, його компоненти також автоматично створюються або знищуються. Крім того, контейнер відповідає за створення та управління життєвим циклом своїх компонентів.

Давайте розглянемо приклад знову. Припустимо, що у нас є клас "Автомобіль" і клас "Двигун". В цьому випадку, композиційне відношення між ними означає, що клас "Автомобіль" містить об'єкт "Двигун" як свій прямий член. Таким чином, коли створюється об'єкт "Автомобіль", він створює також об'єкт "Двигун". Якщо "Автомобіль" більше не потрібен або видаляється, то разом з ним також буде знищено об'єкт "Двигун".

Композиція дозволяє створювати складніші об'єкти, засновані на ієрархії класів, де один клас повністю залежить від іншого. Вона допомагає керувати взаємозв'язком між компонентами та забезпечує їх єдність в контексті контейнера.

Наприклад:

```
#include <iostream>

// Клас "Серце"
```

```
class Heart {
public:
    void beat() {
        std::cout << "Heart is beating!" << std::endl;
    }
};

// Клас "Людина"
class Person {
private:
    Heart heart;  // Об'єкт класу "Серце" як складова частина класу "Людина"

public:
    void doSomething() {
        heart.beat();
    }
};

int main() {
    Person person;
    person.doSomething();

    return 0;
}
```

У цьому прикладі ми маємо клас "Серце", який має метод beat() для симуляції роботи серця. Клас "Людина" композиційно містить об'єкт класу "Серце" як свою складову частину.

У головній функції ми створюємо об'єкт "Людина" з назвою person. Викликаючи метод doSomething(), ми викликаємо метод beat() об'єкта heart, що відображає роботу серця людини.

Успадкування (іноді його називають також наслідуванням) в об'єктно-орієнтованому програмуванні є одним із фундаментальних концепцій. Воно дозволяє створювати нові класи на основі вже існуючих (батьківських або базових класів), утворюючи ієрархію класів.

Успадкування дозволяє поширювати властивості і поведінку базового класу на похідні класи. Похідні класи отримують доступ до публічних і захищених членів базового класу (таких як поля, методи, властивості) і можуть додавати свої власні члени або перевизначати члени базового класу.

Один з основних принципів успадкування - це утворення спеціалізованих класів на базі загального базового класу. Наприклад, уявімо ієрархію класів "Транспортний засіб" як базовий клас, а "Автомобіль", "Велосипед" та "Мотоцикл" як його похідні класи. У цьому випадку "Транспортний засіб" міститиме загальну функціональність, а похідні класи розширять його функціональність, додаючи власні особливості.

Однією з головних переваг успадкування є полегшення повторного використання коду. Базовий клас може містити загальні функції і властивості,

які можуть бути успадковані і використовані у всіх похідних класах. Це забезпечує ефективніше управління кодом і зменшення дублювання.

Успадкування також сприяє поліморфізму, що дозволяє використовувати об'єкти похідних класів як об'єкти базового класу. Це розширює можливості поліморфного програмування і спрощує роботу зі змінними і колекціями об'єктів.

Наприклад, у мові C++ успадкування виконується за допомогою ключового слова class або struct, за яким слідує ім'я похідного класу і базового класу, відокремлені двокрапкою. Наприклад:

```
class BaseClass {
    // Тіло базового класу
};

class DerivedClass : public BaseClass {
    // Тіло похідного класу
};
```

У цьому прикладі клас DerivedClass успадковує властивості і методи класу BaseClass.

Успадкування є важливим інструментом в об'єктно-орієнтованому програмуванні, оскільки дозволяє створювати ієрархії класів з поліморфними можливостями та забезпечує повторне використання коду. Воно сприяє модульності, розширюваності і зрозумілості програмного коду.

Завдання

Варіант 1

1. Створіть клас "Людина" (базовий клас), який містить такі властивості:

- Ім'я
- Вік
- Клас "Адреса", який представляє адресу проживання людини (використовуйте композицію)

2. Створіть клас "Студент" (похідний клас від "Людина"), який містить додаткові властивості:

- Номер студентського квитка
- Курс

3. Створіть клас "Університет" (базовий клас), який містить такі властивості:

- Назва університету
- Список студентів (використовуйте агрегацію)

4. Реалізуйте методи для додавання та видалення студентів зі списку університету.

5. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Університет", "Людина" та "Студент". Додайте студентів до списку університету та виведіть інформацію про них.

6. Продемонструйте успадкування, викликавши методи базового класу та похідного класу для об'єктів типу "Студент".

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 2

1. Створіть клас "Бібліотека", який містить такі властивості:

- Назва бібліотеки
- Адреса бібліотеки
- Список книг, які є в наявності в бібліотеці (використовуйте агрегацію)
- Клас "Бібліотекар", який представляє бібліотекаря, що працює в бібліотеці (використовуйте композицію)

2. Створіть клас "Книга", який містить такі властивості:

- Назва книги
- Автор книги
- Рік видання

3. Створіть клас "Бібліотекар", який містить такі властивості:

- Ім'я бібліотекаря
- Посада бібліотекаря

4. Реалізуйте методи для додавання та видалення книг зі списку книг у бібліотеці.

5. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Бібліотека", "Книга" та "Бібліотекар". Додайте книги до списку книг у бібліотеці та виведіть інформацію про них.

6. Продемонструйте успадкування, створивши похідний клас від "Книга" (наприклад, "Роман") з додатковими властивостями та методами.

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 3

1. Створіть клас "Компанія", який містить такі властивості:

- Назва компанії
- Адреса компанії
- Список підрозділів, які входять до складу компанії (використовуйте агрегацію)
- Головний директор компанії (використовуйте композицію)

2. Створіть клас "Підрозділ", який містить такі властивості:
- Назва підрозділу
- Кількість працівників
- Керівник підрозділу

3. Створіть клас "Працівник", який містить такі властивості:
- Прізвище та ім'я працівника
- Посада працівника
- Заробітна плата працівника

4. Реалізуйте методи для додавання та видалення підрозділів зі списку підрозділів компанії.

5. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Компанія", "Підрозділ" та "Працівник". Додайте підрозділи до списку підрозділів компанії та виведіть інформацію про них.

6. Продемонструйте успадкування, створивши похідний клас від "Працівник" (наприклад, "Менеджер") з додатковими властивостями та методами.

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 4

1. Створіть клас "Університет", який містить такі властивості:
- Назва університету
- Адреса університету
- Список факультетів, які входять до складу університету (використовуйте агрегацію)
- Ректор університету (використовуйте композицію)

2. Створіть клас "Факультет", який містить такі властивості:
- Назва факультету
- Кількість студентів
- Декан факультету

3. Створіть клас "Лектор", який містить такі властивості:

- Прізвище та ім'я лектора
- Посада лектора
- Кафедра, на якій працює лектор (використовуйте композицію)

4. Створіть клас "Кафедра", який містить такі властивості:

- Назва кафедри
- Кількість викладачів
- Голова кафедри

5. Реалізуйте методи для додавання та видалення факультетів зі списку факультетів університету.

6. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Університет", "Факультет", "Лектор" та "Кафедра". Додайте факультети до списку факультетів університету та виведіть інформацію про них.

7. Продемонструйте успадкування, створивши похідний клас від "Лектор" (наприклад, "Доцент") з додатковими властивостями та методами.

8. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 5

1. Створіть клас "Музичний гурт", який містить такі властивості:

- Назва гурту
- Стиль музики
- Список музикантів, які входять до гурту (використовуйте агрегацію)
- Керівник гурту (використовуйте композицію)

2. Створіть клас "Музикант", який містить такі властивості:

- Ім'я музиканта
- Інструмент, на якому він грає

3. Створіть клас "Інструмент", який містить такі властивості:

- Назва інструменту
- Рік виготовлення

4. Реалізуйте методи для додавання та видалення музикантів зі списку музикантів гурту.

5. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Музичний гурт", "Музикант" та "Інструмент". Додайте музикантів до списку музикантів гурту та виведіть інформацію про них.

6. Продемонструйте успадкування, створивши похідний клас від "Музикант" (наприклад, "Вокаліст") з додатковими властивостями та методами.

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 6

1. Створіть клас "Телефон", який містить такі властивості:

- Марка телефону
- Модель телефону
- Екран (використовуйте композицію)
- Камера (використовуйте композицію)

2. Створіть клас "Екран", який містить такі властивості:

- Розмір екрану
- Тип екрану (LCD, OLED і т.д.)

3. Створіть клас "Камера", який містить такі властивості:

- Розмір матриці камери
- Роздільна здатність камери

4. Реалізуйте методи для встановлення та отримання інформації про телефон, екран та камеру.

5. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Телефон", "Екран" та "Камера". Встановіть значення для властивостей та виведіть інформацію про телефон.

6. Продемонструйте успадкування, створивши похідний клас від "Телефон" (наприклад, "Смартфон") з додатковими властивостями та методами.

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 7

1. Створіть клас "Університет", який містить такі властивості:

- Назва університету
- Адреса університету
- Список факультетів, які належать до університету (використовуйте агрегацію)

2. Створіть клас "Факультет", який містить такі властивості:

- Назва факультету
- Декан факультету (використовуйте композицію)
- Список студентів, які навчаються на факультеті (використовуйте агрегацію)

3. Створіть клас "Декан", який містить такі властивості:
- Ім'я декана
- Рік призначення на посаду декана

4. Реалізуйте методи для додавання та видалення студентів зі списку студентів факультету.

5. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Університет", "Факультет" та "Декан". Додайте факультети до списку факультетів університету, додайте студентів до списку студентів факультету та виведіть інформацію про них.

6. Продемонструйте успадкування, створивши похідний клас від "Студент" (наприклад, "Магістр") з додатковими властивостями та методами.

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 8

1. Створіть клас "Магазин", який містить такі властивості:
- Назва магазину
- Адреса магазину
- Список товарів, які є у магазині (використовуйте агрегацію)

2. Створіть клас "Товар", який містить такі властивості:
- Назва товару
- Ціна товару

3. Створіть клас "Кошик", який містить такі властивості:
- Список товарів, які знаходяться у кошику (використовуйте композицію)
- Загальна вартість товарів у кошику

4. Реалізуйте методи для додавання та видалення товарів з кошика та розрахунку загальної вартості.

5. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Магазин", "Товар" та "Кошик". Додайте товари до списку товарів магазину, додайте товари до кошика та виведіть інформацію про них та загальну вартість у кошику.

6. Продемонструйте успадкування, створивши похідний клас від "Товар" (наприклад, "Продукт") з додатковими властивостями та методами.

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 9

1. Створіть клас "Автомобіль", який містить такі властивості:

- Марка автомобіля
- Модель автомобіля
- Двигун (використовуйте композицію)
- Салон (використовуйте композицію)

2. Створіть клас "Двигун", який містить такі властивості:

- Тип двигуна (бензиновий, дизельний і т.д.)
- Об'єм двигуна

3. Створіть клас "Салон", який містить такі властивості:

- Кількість місць в салоні
- Тип салону (тканинний, шкіряний і т.д.)

4. Реалізуйте методи для встановлення та отримання інформації про автомобіль, двигун та салон.

5. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Автомобіль", "Двигун" та "Салон". Встановіть значення для властивостей та виведіть інформацію про автомобіль.

6. Продемонструйте успадкування, створивши похідний клас від "Автомобіль" (наприклад, "СпортивнийАвтомобіль") з додатковими властивостями та методами.

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 10

1. Створіть клас "Банк", який містить такі властивості:

- Назва банку
- Адреса банку
- Список відділень, які належать до банку (використовуйте агрегацію)

2. Створіть клас "Відділення", який містить такі властивості:

- Назва відділення
- Адреса відділення
- Менеджер відділення (використовуйте композицію)
- Список клієнтів, які обслуговуються у відділенні (використовуйте агрегацію)

3. Створіть клас "Менеджер", який містить такі властивості:

- Ім'я менеджера
- Посада менеджера

4. Реалізуйте методи для додавання та видалення клієнтів зі списку клієнтів відділення.

5. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Банк", "Відділення" та "Менеджер". Додайте відділення до списку відділень банку, додайте клієнтів до списку клієнтів відділення та виведіть інформацію про них.

6. Продемонструйте успадкування, створивши похідний клас від "Клієнт" (наприклад, "VIPКлієнт") з додатковими властивостями та методами.

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 11

1. Створіть клас "Комп'ютерна гра", який містить такі властивості:

- Назва гри
- Розробник гри
- Список персонажів у грі (використовуйте агрегацію)

2. Створіть клас "Персонаж", який містить такі властивості:

- Ім'я персонажа
- Рівень персонажа
- Характеристики персонажа (наприклад, здоров'я, сила, інтелект) (використовуйте композицію)

3. Створіть клас "Гравець", який успадковує клас "Персонаж" і містить такі додаткові властивості:

- Рівень гравця
- Кількість набраних очок

4. Реалізуйте методи для додавання та видалення персонажів зі списку персонажів гри.

5. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Комп'ютерна гра", "Персонаж" та "Гравець". Додайте персонажів до списку персонажів гри, створіть гравця і виведіть інформацію про нього.

6. Продемонструйте успадкування, створивши додаткові класи-нащадки від "Гравець" з додатковими властивостями та методами.

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 12

1. Створіть клас "Банк", який містить такі властивості:

- Назва банку
- Адреса банку
- Список клієнтів банку (використовуйте агрегацію)

2. Створіть клас "Клієнт", який містить такі властивості:

- ПІБ клієнта
- Рахунки клієнта (використовуйте агрегацію)

3. Створіть клас "Рахунок", який містить такі властивості:

- Номер рахунку
- Баланс рахунку

4. Реалізуйте методи для створення нового рахунку клієнта, внесення та зняття коштів з рахунку.

5. Створіть клас-нащадок "Вклад", який успадковує клас "Рахунок" і містить такі додаткові властивості:

- Відсоткова ставка вкладу
- Термін дії вкладу
- Реалізуйте методи для нарахування відсотків на вклад та закриття вкладу.

6. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Банк", "Клієнт", "Рахунок" та "Вклад". Додайте клієнтів до списку клієнтів банку, створіть рахунки та вклади для клієнтів, виконайте операції з рахунками та вкладами та виведіть інформацію про них.

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 13

1. Створіть клас "Музична група", який містить такі властивості:

- Назва групи
- Країна походження
- Список учасників групи (використовуйте агрегацію)

2. Створіть клас "Учасник", який містить такі властивості:

- Ім'я учасника
- Інструмент, яким він грає

3. Реалізуйте методи для додавання та видалення учасників у списку учасників групи.

4. Створіть клас-нащадок "Соліст", який успадковує клас "Учасник" і містить таку додаткову властивість:

- Жанр, в якому соліст співає

5. Реалізуйте метод для виведення інформації про соліста та його жанр.

6. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Музична група", "Учасник" та "Соліст". Додайте учасників до списку учасників групи, включаючи соліста, виведіть інформацію про музичну групу та її учасників.

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 14

1. Створіть клас "Будинок", який містить такі властивості:

- Адреса будинку
- Кількість поверхів
- Список квартир (використовуйте агрегацію)

2. Створіть клас "Квартира", який містить такі властивості:

- Номер квартири
- Площа квартири
- Кількість кімнат

3. Реалізуйте методи для додавання та видалення квартир у списку квартир будинку.

4. Створіть клас-нащадок "Багатоквартирний будинок", який успадковує клас "Будинок" і містить таку додаткову властивість:

- Кількість ліфтів у будинку

5. Реалізуйте метод для виведення інформації про кількість ліфтів у будинку та загальну площу квартир у ньому.

6. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Будинок", "Квартира" та "Багатоквартирний будинок". Додайте квартири до списку квартир будинку, включаючи квартири, що знаходяться в окремому багатоквартирному будинку. Виведіть інформацію про будинок та квартири, що знаходяться в ньому.

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 15

1. Створіть клас "Аеропорт", який містить такі властивості:

- Назва аеропорту
- Адреса аеропорту
- Список рейсів (використовуйте агрегацію)

2. Створіть клас "Рейс", який містить такі властивості:

- Номер рейсу
- Пункт відправлення
- Пункт призначення

3. Реалізуйте методи для додавання та видалення рейсів у списку рейсів аеропорту.

4. Створіть клас-нащадок "Міжнародний аеропорт", який успадковує клас "Аеропорт" і містить таку додаткову властивість:

- Список країн, до яких здійснюються рейси

5. Реалізуйте метод для виведення інформації про країни, до яких здійснюються рейси з міжнародного аеропорту.

6. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Аеропорт", "Рейс" та "Міжнародний аеропорт". Додайте рейси до списку рейсів аеропорту, включаючи рейси до певних країн. Виведіть інформацію про аеропорт, рейси та країни, до яких здійснюються рейси з міжнародного аеропорту.

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 16

1. Створіть клас "Тварина" (базовий клас), який містить такі властивості:

- Назва

- Вік

2. Створіть клас "Кіт" (похідний клас від "Тварина"), який містить додаткову властивість:

- Кольор шерсті

3. Створіть клас "Собака" (похідний клас від "Тварина"), який містить додаткову властивість:

- Розмір (маленька, середня, велика)

4. Створіть клас "Цирк" (базовий клас), який містить такі властивості:

- Назва цирку
- Список тварин (використовуйте агрегацію)

5. Реалізуйте методи для додавання та видалення тварин зі списку цирку.

6. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Цирк", "Тварина", "Кіт" та "Собака". Додайте тварин до списку цирку та виведіть інформацію про них.

7. Продемонструйте успадкування, викликавши методи базового класу та похідних класів для об'єктів типу "Кіт" та "Собака".

8. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 17

1. Створіть базовий клас "Об'єкт астрономії" (AstronomyObject), який містить такі властивості:

- Назва об'єкту
- Відстань до об'єкту

2. Створіть похідні класи від "Об'єкт астрономії":

- "Зірка" (Star), яка має додаткову властивість "Температура"
- "Планета" (Planet), яка має додаткову властивість "Радіус"

3. Створіть клас "Сонячна система" (SolarSystem), який містить такі властивості:

- Назва сонячної системи
- Список об'єктів астрономії (використовуйте агрегацію)

4. Реалізуйте методи для додавання та видалення об'єктів астрономії до/зі списку сонячної системи.

5. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Сонячна система", "Об'єкт астрономії", "Зірка" та "Планета". Додайте об'єкти астрономії до списку сонячної системи та виведіть інформацію про них.

6. Продемонструйте успадкування, викликавши методи базового класу та похідних класів для об'єктів типу "Зірка" та "Планета".

7. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 18

1. Створіть клас "Вектор" (Vector), який представляє математичний вектор. Вектор повинен мати такі властивості:

- Розмірність вектора (кількість компонент)
- Масив компонент вектора

2. Створіть клас "Матриця" (Matrix), який представляє математичну матрицю. Матриця повинна мати такі властивості:

- Кількість рядків
- Кількість стовпців
- Двовимірний масив компонент матриці

3. Реалізуйте клас "Математичний об'єкт" (MathObject), який представляє абстрактний математичний об'єкт. Цей клас повинен мати такі властивості:

- Ім'я об'єкта
- Значення об'єкта (це може бути число, вектор або матриця)

4. Створіть клас "Рівняння" (Equation), який представляє математичне рівняння. Рівняння повинно мати такі властивості:

- Ліва частина рівняння (об'єкт типу "Математичний об'єкт")
- Права частина рівняння (об'єкт типу "Математичний об'єкт")

5. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Вектор", "Матриця", "Математичний об'єкт" та "Рівняння". Використайте композицію та агрегацію для створення складних математичних об'єктів, таких як вектори та матриці. Виведіть інформацію про ці об'єкти.

6. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 19

1. Створіть клас "Водний Засіб" (базовий клас), який містить такі властивості:

- Назва
- Максимальна швидкість

2. Створіть клас "Корабель" (похідний клас від "Водний Засіб"), який містить додаткові властивості:

- Кількість пасажирів
- Класифікація (наприклад, вантажний, пасажирський, воєнний)

3. Створіть клас "Яхта" (похідний клас від "Водний Засіб"), який містить додаткові властивості:

- Розмір (наприклад, маленька, середня, велика)
- Кількість кают

4. Створіть клас "Морський Флот" (базовий клас), який містить такі властивості:

- Назва морського флоту
- Список водних засобів (використовуйте агрегацію)

5. Реалізуйте методи для додавання та видалення водних засобів до списку морського флоту.

6. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Морський Флот", "Водний Засіб", "Корабель" та "Яхта". Додайте водні засоби до списку морського флоту та виведіть інформацію про них.

7. Продемонструйте успадкування, викликавши методи базового класу та похідного класу для об'єктів типу "Корабель" та "Яхта".

8. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Варіант 20

1. Створіть клас "Людина" (базовий клас), який містить такі властивості:

- Ім'я
- Вік

2. Створіть клас "МедичнийПрацівник" (похідний клас від "Людина"), який містить додаткові властивості:

- Спеціальність
- Кількість років досвіду

3. Створіть клас "Пацієнт" (похідний клас від "Людина"), який містить додаткову властивість:

- Діагноз

4. Створіть клас "Відділення" (базовий клас), який містить такі властивості:

- Назва відділення
- Список медичних працівників (використовуйте агрегацію)
- Список пацієнтів (використовуйте агрегацію)

5. Створіть клас "Лікарня" (похідний клас від "Відділення"), який містить додаткову властивість:

- Адреса лікарні

6. Реалізуйте методи для додавання та видалення медичних працівників і пацієнтів зі списків відділення та лікарні.

7. Напишіть демонстраційну функцію main(), в якій створюються об'єкти класів "Лікарня", "Відділення", "МедичнийПрацівник" та "Пацієнт". Додайте медичних працівників та пацієнтів до списків відділення та лікарні та виведіть інформацію про них.

8. Продемонструйте успадкування, викликавши методи базового класу та похідного класу для об'єктів типу "МедичнийПрацівник" та "Пацієнт".

9. Розширте функціональність, додавши додаткові методи та властивості до класів за власним бажанням.

Не забудьте додати коментарі та пояснення до коду, щоб роз'яснити його функціональність та зрозумілість.

Лабораторна робота №4 Поліморфізм. Обробка винятків

Мета: Ознайомитись з поняттям поліморфізму у мові С++ та навчитись використовувати віртуальні функції для досягнення поліморфізму. Також вивчити принципи обробки винятків у мові С++.

Теоретична частина

Поліморфізм - це принцип об'єктно-орієнтованого програмування, який дозволяє об'єктам одного класу використовуватися замість об'єктів інших класів, що мають спільний інтерфейс. Це означає, що об'єкти різних класів можуть бути оброблені і використані за допомогою загального коду без необхідності знати про конкретний клас об'єкту.

У С++ поліморфізм можна реалізувати за допомогою віртуальних функцій. **Віртуальна функція** - це функція, оголошена у базовому класі і перевизначена (або заміщена) у похідних класах. Ключове слово "virtual" перед оголошенням функції вказує, що ця функція є віртуальною.

Коли ви викликаєте віртуальну функцію на об'єкті, поведінка функції залежить від типу об'єкта, на якому вона викликається, а не від типу змінної або посилання, яке її містить. Це означає, що коли ви маєте посилання або вказівник на базовий клас, але викликаєте віртуальну функцію, то викликається реалізація цієї функції у відповідному похідному класі.

Оголошення віртуальної функції у базовому класі виглядає так:

```
class Base {
public:
    virtual void virtualFunction() {
        // Реалізація функції у базовому класі
    }
};
```

Основна особливість віртуальних функцій полягає в тому, що при виклику цієї функції на об'єктах через вказівники або посилання базового класу, виконується реалізація цієї функції у відповідному похідному класі, якщо така реалізація існує. Іншими словами, виклик віртуальної функції залежить від фактичного типу об'єкта, на якому вона викликається.

Приклад використання віртуальних функцій у похідних класах:

```
class Derived : public Base {
public:
    void virtualFunction() override {
        // Перевизначена реалізація функції у похідному класі
    }
};
```

У цьому прикладі ми створили похідний клас "Derived", який успадковує від базового класу "Base". У похідному класі ми перевизначили віртуальну функцію "virtualFunction()" зі зміною її реалізації.

Щоб позначити перевизначення функції у похідному класі, використовується ключове слово override. Це покращує зрозуміння коду та дозволяє компілятору перевірити, чи дійсно ми перевизначаємо функцію з базового класу.

У результаті ми можемо працювати з об'єктами похідних класів через вказівники або посилання базового класу і викликати їхні віртуальні функції. При цьому буде виконуватись відповідна реалізація цих функцій у відповідних похідних класах.

Наприклад:

```
Copy code
Base* obj1 = new Derived();
obj1->virtualFunction(); // Виклик функції з похідного класу

Base* obj2 = new Base();
obj2->virtualFunction(); // Виклик функції з базового класу
```

У цьому прикладі ми створюємо об'єкти через вказівники базового класу Base, але викликаємо їхні віртуальні функції. Перший виклик виконує функцію virtualFunction() з похідного класу "Derived", оскільки вказівник obj1 посилається на об'єкт типу "Derived". Другий виклик виконує функцію virtualFunction() з базового класу "Base", оскільки вказівник obj2 посилається на об'єкт типу "Base".

Таким чином, віртуальні функції дозволяють досягати поліморфізму в C++, що дозволяє з легкістю працювати з об'єктами різних класів через їх спільний інтерфейс.

Обробка винятків в мові C++ здійснюється за допомогою механізму винятків (exceptions). Виняток представляє собою спеціальний об'єкт, який викидається під час виникнення помилки або непередбаченої ситуації і може бути перехоплений і оброблений відповідним кодом.

Основні елементи обробки винятків в C++:

1. Викидання винятка: Для викидання винятка використовується оператор throw. Ви можете викинути будь-який об'єкт, який може бути скопійований або кинутий за допомогою оператора throw. Наприклад:

```
throw SomeException(); // Викидання об'єкта винятка
```

2. Перехоплення винятка: Для перехоплення винятка використовується блок try-catch. Ви можете вказати блок коду, в якому виняток може бути перехоплений, і надати відповідний обробник для обробки винятка. Наприклад:

```
try {
    // Блок коду, в якому може виникнути виняток
} catch (const SomeException& e) {
    // Обробник для винятка типу SomeException
} catch (const AnotherException& e) {
    // Обробник для винятка типу AnotherException
} catch (...) {
    // Обробник для всіх інших винятків
}
```

У блоку try вказується код, в якому виняток може виникнути. У блоках catch вказуються обробники для різних типів винятків. Обробники співставляються з викинутими винятками на основі їхніх типів.

3. Розгортання стеку: Якщо виняток викидається в блоку try, система виконання розгортає стек (stack unwinding), що означає виклик деструкторів для об'єктів, які були створені в блоку try, перед тим, як виняток буде перехоплено.

4. Об'єкти винятків: Ви можете визначати власні типи винятків у вигляді класів. Ці класи можуть мати власні поля і методи, які дозволяють додатково ідентифікувати та обробляти винятки.

```
class MyException : public std::exception {
public:
    const char* what() const noexcept override {
        return "Це мій виняток!";
    }
};
```

У цьому прикладі MyException є власним класом винятка, який успадковує від std::exception. Він перевизначає метод what(), який повертає повідомлення про виняток.

5. Обробка винятків за межами функцій: Винятки можуть бути перехоплені не тільки у тому місці, де вони викидаються, але й у вищих рівнях виклику функцій. Якщо виняток не оброблений у поточній функції, він буде переданий наступному обробнику вищого рівня, і так далі, до тих пір, поки виняток не буде перехоплено або програма не завершиться.

Це основні принципи обробки винятків в мові C++. Використання винятків дозволяє контролювати та обробляти помилки у програмі, дозволяючи зручно реагувати на непередбачені ситуації і забезпечувати безпеку виконання коду.

Завдання

Варіант 1

1. Розробіть систему керування транспортними засобами, яка включатиме різні типи транспортних засобів, такі як автомобілі, мотоцикли і вантажівки. Кожен з цих типів має власні характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен транспортний засіб повинен мати наступні характеристики:

- Запас палива (у літрах)
- Швидкість (у км/год)
- Вартість (у доларах)

Кожен тип транспортного засобу має власні додаткові характеристики:

Автомобіль:

- Кількість дверей
- Максимальна швидкість (у км/год)

Мотоцикл:

- Тип двигуна (бензиновий, електричний і т.д.)
- Об'єм двигуна (у куб. см)

Вантажівка:

- Максимальне навантаження (у кг)
- Кількість вісей

2. Створіть базовий абстрактний клас Vehicle з віртуальними функціями та використати поліморфізм для реалізації методу обчислення вартості палива на відстань. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Car, Motorcycle і Truck, які успадковуються від класу Vehicle. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо введений негативний запас палива або швидкість, або якщо відстань для обчислення вартості палива менше або дорівнює нулю.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 2

1. Розробіть систему керування ЖД вокзалом, яка включатиме обробку різних типів поїздів, таких як пасажирські поїзди та вантажні поїзди. Кожен тип поїзда має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен поїзд має наступні характеристики:

- Номер поїзда
- Пункт відправлення
- Пункт прибуття
- Час відправлення
- Час прибуття

Кожен тип поїзда має власні додаткові характеристики:

Пасажирський поїзд:

- Кількість місць (сидячих та ліжачих)

Вантажний поїзд:

- Максимальна вага вантажу (у тоннах)
- Кількість вагонів

2. Створіть базовий абстрактний клас Train з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи PassengerTrain та FreightTrain, які успадковуються від класу Train. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо введений час прибуття раніше або дорівнює часу відправлення.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 3

1. Розробіть систему керування продуктовим магазином, яка включатиме обробку різних типів продуктів, таких як фрукти, овочі та молочні продукти. Кожен тип продукту має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен продукт має наступні характеристики:

- Назва продукту
- Ціна (за одиницю товару)
- Кількість на складі

Кожен тип продукту має власні додаткові характеристики:

Фрукти:

- Тип фрукту (яблуко, банан, апельсин і т.д.)
- Вага (у кілограмах)

Овочі:

- Тип овоча (морква, капуста, буряк і т.д.)
- Країна походження

Молочні продукти:

- Тип продукту (молоко, йогурт, сир і т.д.)
- Виробник

2. Створіть базовий абстрактний клас Product з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Fruit, Vegetable та DairyProduct, які успадковуються від класу Product. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо введена некоректна кількість на складі або негативна ціна.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 4

1. Розробіть систему керування банком, яка включатиме обробку різних типів банківських рахунків, таких як звичайний рахунок та рахунок з відсотковою ставкою. Кожен тип рахунку має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен банківський рахунок має наступні характеристики:

- Номер рахунку
- Власник рахунку
- Баланс

Кожен тип рахунку має власні додаткові характеристики:

Звичайний рахунок:

- Мінімальний дозволений баланс

Рахунок з відсотковою ставкою:

- Відсоткова ставка

2. Створіть базовий абстрактний клас BankAccount з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи RegularAccount та InterestAccount, які успадковуються від класу BankAccount. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо некоректний номер рахунку або негативна відсоткова ставка.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 5

1. Розробіть систему керування університетом, яка включатиме обробку різних типів студентів та викладачів. Кожен тип особи має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожна особа (студент або викладач) має наступні характеристики:

- Ім'я
- Вік

- Стать

Кожен тип особи має власні додаткові характеристики:

Студент:

- Курс
- Спеціальність

Викладач:

- Посада
- Предмет, який він викладає

2. Створіть базовий абстрактний клас Person з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Student та Professor, які успадковуються від класу Person. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо некоректний вік або порожнє ім'я.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 6

1. Розробіть систему керування магазином, яка включатиме обробку різних типів товарів. Кожен тип товару має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен товар має наступні характеристики:

- Назва
- Ціна
- Кількість на складі

Кожен тип товару має власні додаткові характеристики:

Меблі:

- Матеріал
- Колір

Електроніка:

- Бренд
- Гарантійний термін

2. Створіть базовий абстрактний клас Product з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Furniture та Electronics, які успадковуються від класу Product. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо некоректна ціна товару або від'ємна кількість на складі.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 7

1. Розробіть систему керування човнами, яка включатиме обробку різних типів човнів. Кожен тип човна має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен човен має наступні характеристики:

- Назва
- Максимальна швидкість
- Вантажопідйомність

Кожен тип човна має власні додаткові характеристики:

Весловий човен:

- Кількість весел
- Тип матеріалу

Моторний човен:

- Потужність двигуна
- Тип палива

2. Створіть базовий абстрактний клас Boat з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Rowboat та Motorboat, які успадковуються від класу Boat. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо некоректна швидкість човна або від'ємна вантажопідйомність.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 8

1. Розробіть систему керування одягом, яка включатиме різні типи одягу. Кожен тип одягу має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен одяг має наступні характеристики:

- Назва
- Розмір
- Колір

Кожен тип одягу має власні додаткові характеристики:

Футболка:

- Матеріал
- Тип коміра

Штани:

- Матеріал
- Тип застібки

2. Створіть базовий абстрактний клас Clothing з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи TShirt та Pants, які успадковуються від класу Clothing. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо некоректний розмір одягу або порожнє ім'я.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 9

1. Розробіть систему управління аеропортом, яка включатиме різні типи повітряних суден. Кожен тип повітряного судна має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожне повітряне судно має наступні характеристики:

- Номер рейсу
- Максимальна вага
- Максимальна кількість пасажирів

Кожен тип повітряного судна має власні додаткові характеристики:

Літак:

- Модель

- Кількість двигунів

Гвинтокрил:

- Кількість гвинтів
- Радіус дії

2. Створіть базовий абстрактний клас Aircraft з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Airplane та Helicopter, які успадковуються від класу Aircraft. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо некоректний номер рейсу або від'ємна вага повітряного судна.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 10

1. Розробіть систему управління лікарнею, яка включатиме різні типи медичних спеціалістів та пацієнтів. Кожен тип спеціаліста та пацієнта має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен медичний спеціаліст має наступні характеристики:

- Ім'я
- Спеціалізація
- Вік

Кожен пацієнт має наступні характеристики:

- Ім'я
- Вік
- Діагноз

Кожен тип медичного спеціаліста має власні додаткові характеристики:

Лікар:

- Стаж роботи
- Число успішно проведених операцій

Медсестра:

- Рівень кваліфікації
- Число виконаних процедур

2. Створіть базовий абстрактний клас MedicalPersonnel (Медичний персонал) з віртуальними функціями та використати поліморфізм для реалізації

додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Doctor та Nurse, які успадковуються від класу MedicalPersonnel. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо некоректне ім'я пацієнта або вік, який перевищує припустимий діапазон.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 11

1. Розробіть систему управління офісом, яка включатиме різні типи співробітників та офісне обладнання. Кожен тип співробітника та обладнання має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен співробітник має наступні характеристики:

- Ім'я
- Вік
- Посада

Кожен тип обладнання має наступні характеристики:

- Назва
- Кількість

Кожен тип співробітника має власні додаткові характеристики:

Менеджер:

- Департамент
- Кількість підлеглих

Розробник:

- Мова програмування
- Досвід роботи (у роках)

2. Створіть базовий абстрактний клас Employee (Співробітник) з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Manager та Developer, які успадковуються від класу Employee. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо некоректне ім'я співробітника або вік, який перевищує припустимий діапазон.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 12

1. Розробіть систему управління програмуванням, яка включатиме різні типи програмістів та проекти. Кожен тип програміста та проекту має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен програміст має наступні характеристики:

- Ім'я
- Вік
- Рівень навичок

Кожен проект має наступні характеристики:

- Назва
- Рівень складності
- Кількість розробників

Кожен тип програміста має власні додаткові характеристики:

Frontend-розробник:

- Володіння мовами розмітки (HTML, CSS)
- Досвід роботи (у роках)

Backend-розробник:

- Володіння мовами програмування (Java, Python, C++)
- Досвід роботи (у роках)

2. Створіть базовий абстрактний клас Programmer (Програміст) з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи FrontendDeveloper та BackendDeveloper, які успадковуються від класу Programmer. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо некоректне ім'я програміста або вік, який перевищує припустимий діапазон.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 13

1. Розробіть систему управління географічними об'єктами, яка включатиме різні типи об'єктів, такі як країни, міста та географічні області. Кожен тип об'єкту має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен географічний об'єкт має наступні характеристики:

- Назва
- Площа
- Населення

Кожен тип об'єкту має власні додаткові характеристики:

Країна:

- Континент
- Кількість областей

Місто:

- Країна
- Кількість жителів

Географічна область:

- Країна
- Видимість

2. Створіть базовий абстрактний клас GeographicalObject (Географічний об'єкт) з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Country (Країна), City (Місто) та GeographicalRegion (Географічна область), які успадковуються від класу GeographicalObject. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо некоректна площа географічного об'єкта або невірна кількість населення.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 14

1. Розробіть систему управління автомобілями, яка включатиме різні типи автомобілів, такі як легкові автомобілі, вантажні автомобілі та автобуси. Кожен

тип автомобіля має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен автомобіль має наступні характеристики:

- Марка
- Рік випуску
- Швидкість

Кожен тип автомобіля має власні додаткові характеристики:

Легковий автомобіль:

- Кількість пасажирів
- Об'єм двигуна

Вантажний автомобіль:

- Максимальне навантаження (в кілограмах)
- Об'єм кузова

Автобус:

- Кількість пасажирських місць
- Комфортність (від 1 до 10)

2. Створіть базовий абстрактний клас Car (Автомобіль) з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи PassengerCar (Легковий автомобіль), CargoTruck (Вантажний автомобіль) та Bus (Автобус), які успадковуються від класу Car. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо некоректний рік випуску автомобіля або невірна швидкість.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 15

1. Розробіть систему управління геометричними фігурами, яка включатиме різні типи фігур, такі як коло, прямокутник та трикутник. Кожен тип фігури має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожна геометрична фігура має наступні характеристики:

- Назва фігури
- Площа
- Периметр

Кожен тип фігури має власні додаткові характеристики:

Коло:

- Радіус

Прямокутник:

- Довжина
- Ширина

Трикутник:

- Довжина сторони A
- Довжина сторони B
- Довжина сторони C

2. Створіть базовий абстрактний клас Shape (Фігура) з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Circle (Коло), Rectangle (Прямокутник) та Triangle (Трикутник), які успадковуються від класу Shape. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо некоректний радіус кола або невірні довжини сторін прямокутника та трикутника.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 16

1. Розробіть систему управління IT компанією, яка включатиме різні типи співробітників, такі як програмісти, тестувальники та менеджери. Кожен тип співробітника має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен співробітник має наступні характеристики:

- Ім'я
- Вік
- Заробітна плата

Кожен тип співробітника має власні додаткові характеристики:

Програміст:

- Мови програмування, якими володіє

Тестувальник:

- Типи тестування, якими володіє

Менеджер:

- Відділ, яким керує

2. Створіть базовий абстрактний клас Employee (Співробітник) з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Programmer (Програміст), Tester (Тестувальник) та Manager (Менеджер), які успадковуються від класу Employee. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики, які були зазначені вище.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо некоректна заробітна плата або неправильно вказані мови програмування.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 17

1. Розробіть систему для виконання математичних операцій над числами, яка включатиме різні типи операцій, такі як додавання, віднімання та множення. Кожен тип операції має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожна операція має наступні характеристики:

- Назва операції
- Вхідні числа

Кожен тип операції має власні додаткові характеристики:

Додавання:

- Кількість чисел для додавання

Віднімання:

- Кількість чисел для віднімання

Множення:

- Кількість чисел для множення

2. Створіть базовий абстрактний клас Operation (Операція) з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Addition (Додавання), Subtraction (Віднімання) та Multiplication (Множення), які успадковуються від класу Operation. Реалізуйте в них відповідні віртуальні функції та додайте необхідні характеристики та методи для кожної операції.

4. У вашій програмі мають бути використані виняткові ситуації для обробки некоректних даних, наприклад, якщо вказано неправильну кількість чисел для операції або введені некоректні числа.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 18

1. Розробіть систему для роботи з різними видами тварин, яка включатиме різні типи тварин, такі як собаки, коти та птахи. Кожен тип тварини має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожна тварина має наступні характеристики:

- Ім'я
- Вік

Кожен тип тварини має власні додаткові характеристики:

Собака:

- Порода
- Розмір (маленька, середня, велика)

Кіт:

- Порода
- Характер (лагідний, грайливий, незалежний)

Птах:

- Тип птаха (попугай, канарка, сокіл)

2. Створіть базовий абстрактний клас Animal (Тварина) з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Dog (Собака), Cat (Кіт) та Bird (Птах), які успадковуються від класу Animal. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики та методи для кожного типу тварини.

4. У вашій програмі мають бути можливості для створення та взаємодії з різними видами тварин. Ви можете додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 19

1. Розробіть систему для отримання та відображення інформації про погоду у різних містах. Система має підтримувати різні джерела отримання погодних

даних, такі як Інтернет-сервіси та датчики. Кожне джерело має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожне джерело погодних даних має наступні характеристики:

- Назва джерела
- Місто

Кожен тип джерела має власні додаткові характеристики:

Інтернет-сервіс:

- URL-адреса для отримання погодних даних

Датчик:

- Тип датчика (температура, вологість, тиск)

2. Створіть базовий абстрактний клас WeatherSource (Джерело погоди) з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи InternetService (Інтернет-сервіс) та Sensor (Датчик), які успадковуються від класу WeatherSource. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики та методи для кожного типу джерела погоди.

4. У вашій програмі мають бути можливості для створення джерел погодних даних, отримання та відображення погоди з різних джерел. Ви можете додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 20

1. Розробіть систему для зберігання та керування інформацією про студентів. Система має підтримувати різні типи студентів, такі як бакалаври, магістри та аспіранти. Кожен тип студента має свої характеристики, які потрібно реалізувати за допомогою наслідування, сетерів та гетерів.

Кожен студент має наступні характеристики:

- Прізвище
- Ім'я
- Номер студентського квитка

Кожен тип студента має власні додаткові характеристики:

Бакалавр:

- Курс

Магістр:

- Спеціалізація

Аспірант:

- Науковий керівник

2. Створіть базовий абстрактний клас Student (Студент) з віртуальними функціями та використати поліморфізм для реалізації додаткових методів та функцій. Також, додайте виняткові ситуації для обробки некоректних даних.

3. Створіть похідні класи Bachelor (Бакалавр), Master (Магістр) та PhD (Аспірант), які успадковуються від класу Student. Реалізуйте в них відповідні віртуальні функції та додайте додаткові характеристики та методи для кожного типу студента.

4. У вашій програмі мають бути можливості для створення студентів різних типів, зберігання їх інформації та виведення на екран. Ви можете додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Ви можете розширити його, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Лабораторна робота №5 Перевантаження операторів

Мета: ознайомитись з поняттям перевантаження операторів та навчитись їх програмно реалізовувати мовою С++.

Теоретична частина

Перевантаження операторів в С++ дозволяє нам змінювати стандартну поведінку операторів для об'єктів класу. Вихідні типи операторів, такі як +, -, *, /, =, <, == та багато інших, можна змінити таким чином, щоб вони працювали з об'єктами класу, враховуючи наші власні правила.

Оператори можна перевантажити як функції-члени класу або як зовнішні функції. Коли оператор перевантажується як функція-член класу, вона має наступний формат:

```
return_type operatorоператор(parameters)
{
    // Тіло функції
}
```

Тут operatorоператор є ключовим словом operator з послідовністю операторів, які ми хочемо перевантажити. Наприклад, operator+ перевантажує оператор додавання.

Оператори також можуть бути перевантажені як зовнішні функції. У цьому випадку, формат функції виглядає наступним чином:

```
return_type operatorоператор(parameters)
{
    // Тіло функції
}
```

Зазвичай зовнішні функції-оператори викликаються з двома аргументами, але можуть мати інший кількість параметрів, в залежності від типу оператора, який ми перевантажуємо.

Важливо зазначити, що не всі оператори можна перевантажити. Наприклад, оператори . (крапка) і .* (стрілка на вказівник на член) не можуть бути перевантажені. Крім того, оператори, які змінюють пріоритети виразів, такі як () (дужки) і [] (квадратні дужки), також не можуть бути перевантажені.

При перевантаженні операторів важливо враховувати загальні правила їх використання і зберігати логічну інтерпретацію відповідних операцій. Перевантажені оператори повинні залишатися зрозумілими і прийнятними з точки зору інших розробників, які можуть використовувати ваш код.

Наприклад:

```
class Complex {
private:
    double real;
    double imaginary;

public:
    Complex(double real, double imaginary) : real(real),
imaginary(imaginary) {}
```

```
        // Перевантаження оператора +
        Complex operator+(const Complex& other) const {
            return   Complex(real   +   other.real,   imaginary   +
other.imaginary);
        }
    };
```

У цьому прикладі ми перевантажили оператор + для класу Complex. У функції-члені operator+ ми додаємо відповідні реальні та уявні частини об'єктів Complex і повертаємо новий об'єкт Complex, який є сумою двох комплексних чисел.

Тепер ми можемо використовувати перевантажений оператор + для додавання комплексних чисел:

```
    Complex a(2.0, 3.5);
    Complex b(1.5, 4.2);

    Complex c = a + b;  // Виклик перевантаженого оператора +

    // Виведення результату
    std::cout  <<  "Результат:  "  <<  c.getReal()  <<  "  +  "  <<
c.getImaginary() << "i" << std::endl;
```

У цьому прикладі ми створюємо два об'єкти Complex a і b. Потім ми використовуємо перевантажений оператор + для додавання цих об'єктів і отримання результату в об'єкті c. Нарешті, ми виводимо результат додавання.

Завдання

Варіант 1

Створіть клас Vector, який представляє тривимірний вектор у просторі. В цьому класі перевантажте наступні оператори:

1. Оператор + для додавання двох векторів.
2. Оператор - для віднімання двох векторів.
3. Оператор * для множення вектора на скаляр.
4. Оператор == для порівняння двох векторів на рівність.
5. Оператор != для порівняння двох векторів на нерівність.
6. Оператор << для виводу вектора у форматі (x, y, z).

Додайте в клас також необхідні конструктори, деструктор та інші методи, які можуть знадобитись для роботи з векторами.

Напишіть програму, де ви використовуєте цей клас та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу Vector і виконайте з ними операції додавання, віднімання, множення на скаляр, порівняння на рівність та виведення на екран.

Варіант 2

Створіть клас Complex, який представляє комплексне число у вигляді a + bi, де a та b - це дійсні числа, а i - уявна одиниця. В класі Complex перевантажте наступні оператори:

1. Оператор + для додавання двох комплексних чисел.
2. Оператор - для віднімання двох комплексних чисел.
3. Оператор * для множення двох комплексних чисел.
4. Оператор / для ділення одного комплексного числа на інше.
5. Оператор == для порівняння двох комплексних чисел на рівність.
6. Оператор != для порівняння двох комплексних чисел на нерівність.
7. Оператор << для виводу комплексного числа у форматі "a + bi".

Додайте в клас також необхідні конструктори, деструктор та інші методи, які можуть знадобитись для роботи з комплексними числами.

Напишіть програму, де ви використовуєте цей клас та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу Complex і виконайте з ними операції додавання, віднімання, множення, ділення, порівняння на рівність та виведення на екран.

Варіант 3

Створіть клас Matrix, який представляє квадратну матрицю розміром NxN. В класі Matrix перевантажте наступні оператори:

1. Оператор + для додавання двох матриць.
2. Оператор - для віднімання двох матриць.
3. Оператор * для множення двох матриць.
4. Оператор * для множення матриці на скаляр.
5. Оператор == для порівняння двох матриць на рівність.
6. Оператор != для порівняння двох матриць на нерівність.
7. Оператор << для виводу матриці на екран.

Додайте в клас також необхідні конструктори, деструктор та інші методи, які можуть знадобитись для роботи з матрицями.

Напишіть програму, де ви використовуєте цей клас та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу Matrix і виконайте з ними операції додавання, віднімання, множення, порівняння на рівність та виведення на екран.

Варіант 4

Створіть клас String, який представляє рядок символів. В класі String перевантажте наступні оператори:

1. Оператор + для конкатенації двох рядків.
2. Оператор == для порівняння двох рядків на рівність.

3. Оператор != для порівняння двох рядків на нерівність.

4. Оператор [] для доступу до символу за індексом.

5. Оператор << для виводу рядка на екран.

Додайте в клас також необхідні конструктори, деструктор та інші методи, які можуть знадобитись для роботи з рядками.

Напишіть програму, де ви використовуєте цей клас та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу String і виконайте з ними операції конкатенації, порівняння на рівність, доступу до символу за індексом та виведення на екран.

Варіант 5

Створіть клас Fraction, який представляє дріб у вигляді чисельника та знаменника. В класі Fraction перевантажте наступні оператори:

1. Оператор + для додавання двох дробів.

2. Оператор - для віднімання двох дробів.

3. Оператор * для множення двох дробів.

4. Оператор / для ділення одного дробу на інший.

5. Оператор == для порівняння двох дробів на рівність.

6. Оператор != для порівняння двох дробів на нерівність.

7. Оператор << для виводу дробу у форматі "чисельник/знаменник".

Додайте в клас також необхідні конструктори, деструктор та інші методи, які можуть знадобитись для роботи з дробами.

Напишіть програму, де ви використовуєте цей клас та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу Fraction і виконайте з ними операції додавання, віднімання, множення, ділення, порівняння на рівність та виведення на екран.

Варіант 6

Створіть клас Time, який представляє час у годинах, хвилинах та секундах. В класі Time перевантажте наступні оператори:

1. Оператор + для додавання двох часів.

2. Оператор - для віднімання двох часів.

3. Оператор ++ (постфіксний) для інкременту часу на одну секунду.

4. Оператор -- (постфіксний) для декременту часу на одну секунду.

5. Оператор == для порівняння двох часів на рівність.

6. Оператор != для порівняння двох часів на нерівність.

7. Оператор << для виводу часу у форматі "гг:хх:сс".

Додайте в клас також необхідні конструктори, деструктор та інші методи, які можуть знадобитись для роботи з часом.

Напишіть програму, де ви використовуєте цей клас та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу Time і виконайте з

ними операції додавання, віднімання, інкременту, декременту, порівняння на рівність та виведення на екран.

Варіант 7

Створіть клас Date, який представляє дату у форматі день, місяць, рік. В класі Date перевантажте наступні оператори:

1. Оператор ++ (префіксний) для інкременту дати на 1 день.
2. Оператор -- (префіксний) для декременту дати на 1 день.
3. Оператор + для додавання до дати заданої кількості днів.
4. Оператор - для віднімання від дати заданої кількості днів.
5. Оператор == для порівняння двох дат на рівність.
6. Оператор != для порівняння двох дат на нерівність.
7. Оператор << для виводу дати у форматі "дд/мм/рррр".

Додайте в клас також необхідні конструктори, деструктор та інші методи, які можуть знадобитись для роботи з датами.

Напишіть програму, де ви використовуєте цей клас та перевірте роботу всіх перевантажених операторів. Створіть об'єкт класу Date і виконайте з ним операції інкременту, декременту, додавання, віднімання, порівняння на рівність та виведення на екран.

Варіант 8

Створіть клас Rectangle, який представляє прямокутник з заданими шириною та висотою. Ширина та висота повинні бути додатними цілими числами. В класі Rectangle перевантажте наступні оператори:

1. Оператор + для об'єднання двох прямокутників. Результатом має бути новий прямокутник, який охоплює обидва початкових прямокутники.
2. Оператор - для віднімання одного прямокутника від іншого. Результатом має бути новий прямокутник, який представляє різницю між початковими прямокутниками.
3. Оператор * для масштабування прямокутника. Передавайте ціле число як аргумент, і результатом має бути новий прямокутник, який має ширину та висоту, помножені на це число.
4. Оператор / для зменшення прямокутника. Передавайте ціле число як аргумент, і результатом має бути новий прямокутник, який має ширину та висоту, поділені на це число.
5. Оператор == для порівняння двох прямокутників на рівність.
6. Оператор != для порівняння двох прямокутників на нерівність.
7. Оператор << для виводу прямокутника у зрозумілому форматі.

Додайте в клас також необхідні конструктори, деструктор та інші методи, які можуть знадобитись для роботи з прямокутниками.

Напишіть програму, де ви використовуєте цей клас та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу Rectangle і виконайте з ними операції об'єднання, віднімання, масштабування, зменшення, порівняння на рівність та виведення на екран.

Варіант 9

Створіть клас Point, який представляє точку в тривимірному просторі з координатами (x, y, z). Координати точки повинні бути дійсними числами. В класі Point перевантажте наступні оператори:

1. Оператор + для додавання двох точок. Результатом має бути нова точка, координати якої обчислюються як сума координат відповідних точок.
2. Оператор - для віднімання однієї точки від іншої. Результатом має бути нова точка, координати якої обчислюються як різниця координат відповідних точок.
3. Оператор * для множення точки на дійсне число. Передайте дійсне число як аргумент, і результатом має бути нова точка, координати якої обчислюються як добуток кожної координати точки на це число.
4. Оператор == для порівняння двох точок на рівність.
5. Оператор != для порівняння двох точок на нерівність.
6. Оператор << для виводу точки у зрозумілому форматі.

Додайте в клас також необхідні конструктори, деструктор та інші методи, які можуть знадобитись для роботи з точками.

Напишіть програму, де ви використовуєте цей клас та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу Point і виконайте з ними операції додавання, віднімання, множення на дійсне число, порівняння на рівність, а також виведіть їх на екран.

Варіант 10

Створіть клас Employee, який представляє працівника в офісі. Клас Employee повинен мати наступні властивості:

- Ім'я працівника (name) - рядок.
- Вік працівника (age) - ціле число.
- Заробітна плата працівника (salary) - дійсне число.

В класі Employee перевантажте наступні оператори:

1. Оператор + для об'єднання двох працівників. Результатом має бути новий об'єкт Employee, в якому значення властивостей встановлені згідно з наступними правилами:
   - Ім'я нового працівника має бути поєднанням імен обох початкових працівників.
   - Вік нового працівника має бути середнім значенням віку двох початкових працівників (округлено до найближчого цілого).

- Заробітна плата нового працівника має бути сумою заробітних плат двох початкових працівників.

2. Оператор == для порівняння двох працівників на рівність. Працівники вважаються рівними, якщо всі їхні властивості (ім'я, вік, заробітна плата) мають однакові значення.
3. Оператор != для порівняння двох працівників на нерівність. Працівники вважаються нерівними, якщо хоча б одна з їхніх властивостей відрізняється.

Додайте в клас Employee також необхідні конструктори, деструктор та інші методи, які можуть знадобитись для роботи з працівниками.

Напишіть програму, де ви використовуєте цей клас та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу Employee і виконайте з ними операції об'єднання, порівняння на рівність. Виведіть результати на екран.

Варіант 11

Створіть клас Book, який представляє книгу. Клас Book повинен мати наступні властивості:

- Назва книги (title) - рядок.
- Автор книги (author) - рядок.
- Рік видання книги (year) - ціле число.

В класі Book перевантажте наступні оператори:

1. Оператор == для порівняння двох книг на рівність. Книги вважаються рівними, якщо їхні назви, автори та роки видання мають однакові значення.
2. Оператор != для порівняння двох книг на нерівність. Книги вважаються нерівними, якщо хоча б одна з їхніх властивостей відрізняється.
3. Оператор < для порівняння двох книг за алфавітним порядком назви. Книга book1 вважається меншою за книгу book2, якщо назва book1 передує алфавітно назві book2.
4. Оператор > для порівняння двох книг за алфавітним порядком назви. Книга book1 вважається більшою за книгу book2, якщо назва book1 слідує алфавітно за назвою book2.
5. Оператор << для виводу книги на екран у зрозумілому форматі. Виведіть на екран назву, автора та рік видання книги.

Додайте в клас Book також необхідні конструктори, деструктор та інші методи, які можуть знадобитись для роботи з книгами.

Напишіть програму, де ви використовуєте цей клас та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу Book і виконайте з ними порівняння, виведення на екран та інші операції.

Варіант 12

Створіть клас Car, який представляє автомобіль. Клас Car повинен мати наступні властивості:

- Марка автомобіля (brand) - рядок.
- Рік випуску автомобіля (year) - ціле число.
- Швидкість автомобіля (speed) - десяткове число.

В класі Car перевантажте наступні оператори:

1. Оператор + для об'єднання двох автомобілів. Результатом має бути новий автомобіль, який має марку та рік випуску першого автомобіля, а швидкість - суму швидкостей обох автомобілів.
2. Оператор - для віднімання одного автомобіля від іншого. Результатом має бути новий автомобіль, який має марку та рік випуску першого автомобіля, а швидкість - різницю швидкостей обох автомобілів.
3. Оператор * для масштабування швидкості автомобіля. Передайте ціле число як аргумент, і результатом має бути новий автомобіль, який має марку та рік випуску початкового автомобіля, а швидкість - швидкість початкового автомобіля, помножену на це число.
4. Оператор / для зменшення швидкості автомобіля. Передайте ціле число як аргумент, і результатом має бути новий автомобіль, який має марку та рік випуску початкового автомобіля, а швидкість - швидкість початкового автомобіля, поділену на це число.
5. Оператор == для порівняння двох автомобілів на рівність. Автомобілі вважаються рівними, якщо марка, рік випуску та швидкість мають однакові значення.
6. Оператор != для порівняння двох автомобілів на нерівність. Автомобілі вважаються нерівними, якщо марка, рік випуску або швидкість мають різні значення.

Додайте в клас Car також необхідні конструктори, деструктор та інші методи, які можуть знадобитись для роботи.

Напишіть програму, де ви використовуєте клас Car та перевірте роботу всіх перевантажених операторів. Створіть кілька об'єктів класу Car і виконайте з ними операції об'єднання, віднімання, масштабування, зменшення, порівняння на рівність та нерівність.

Варіант 13

Створіть клас BankAccount, який представляє банківський рахунок. Клас BankAccount повинен мати наступні властивості:

- Номер рахунку (accountNumber) - рядок.
- Баланс рахунку (balance) - дійсне число.

В класі BankAccount перевантажте наступні оператори:

1. Оператор + для додавання двох рахунків. Результатом має бути новий об'єкт BankAccount, у якого номер рахунку буде зберігати поєднання номерів рахунків, а баланс - суму балансів початкових рахунків.
2. Оператор - для віднімання одного рахунку від іншого. Результатом має бути новий об'єкт BankAccount, у якого номер рахунку буде зберігати початковий номер першого рахунку, а баланс - різницю балансів початкових рахунків.
3. Оператор * для множення балансу рахунку на певний множник. Передавайте дійсне число як аргумент, і результатом має бути новий об'єкт BankAccount, у якого номер рахунку та баланс будуть зберігати початкові значення, помножені на заданий множник.
4. Оператор / для ділення балансу рахунку на певний дільник. Передавайте дійсне число як аргумент, і результатом має бути новий об'єкт BankAccount, у якого номер рахунку та баланс будуть зберігати початкові значення, поділені на заданий дільник.
5. Оператор += для додавання до поточного балансу рахунку певної суми. Передавайте дійсне число як аргумент, і змінюйте поточний об'єкт BankAccount, збільшуючи його баланс на задану суму.
6. Оператор -= для віднімання від поточного балансу рахунку певної суми. Передавайте дійсне число як аргумент, і змінюйте поточний об'єкт BankAccount, зменшуючи його баланс на задану суму.
7. Оператор == для порівняння двох рахунків на рівність за номером рахунку та балансом.
8. Оператор != для порівняння двох рахунків на нерівність за номером рахунку та балансом.
9. Оператор << для виводу рахунку у зрозумілому форматі. Виведіть номер рахунку та баланс у форматі "Account: {номер_рахунку}, Balance: {баланс}".

Додайте в клас також необхідні конструктори, деструктор та інші методи, які можуть знадобитись для роботи з рахунком.

Напишіть програму, де ви використовуєте цей клас BankAccount та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу BankAccount і виконайте з ними операції додавання, віднімання, множення, ділення, порівняння на рівність, зміни балансу та виведення на екран.

Варіант 14

Створіть клас Product, який представляє товар у магазині. Клас Product повинен мати наступні властивості:

- Назва товару (name) - рядок.

- Ціна товару (price) - дійсне число.
- Кількість товару на складі (quantity) - ціле число.

В класі Product перевантажте наступні оператори:

1. Оператор + для об'єднання двох товарів. Результатом має бути новий об'єкт Product, у якого назва товару буде складатись з початкових назв товарів, а ціна і кількість товару будуть сумою цін і кількостей початкових товарів.
2. Оператор - для віднімання одного товару від іншого. Результатом має бути новий об'єкт Product, у якого назва товару буде зберігати початкову назву першого товару, а ціна і кількість товару будуть різницею цін і кількостей початкових товарів.
3. Оператор * для масштабування ціни товару на певний множник. Передавайте дійсне число як аргумент, і результатом має бути новий об'єкт Product, у якого назва товару та кількість товару будуть зберігати початкові значення, а ціна буде множенням початкової ціни на заданий множник.
4. Оператор / для зменшення ціни товару на певний дільник. Передавайте дійсне число як аргумент, і результатом має бути новий об'єкт Product, у якого назва товару та кількість товару будуть зберігати початкові значення, а ціна буде поділенням початкової ціни на заданий дільник.
5. Оператор += для додавання до поточної кількості товару певної кількості. Передавайте ціле число як аргумент, і змінюйте поточний об'єкт Product, збільшуючи його кількість товару на задану величину.
6. Оператор -= для віднімання від поточної кількості товару певної кількості. Передавайте ціле число як аргумент, і змінюйте поточний об'єкт Product, зменшуючи його кількість товару на задану величину.
7. Оператор == для порівняння двох товарів на рівність за назвою, ціною та кількістю.
8. Оператор != для порівняння двох товарів на нерівність за назвою, ціною та кількістю.
9. Оператор << для виводу товару у зрозумілому форматі. Виведіть назву товару, ціну та кількість у форматі "Product: {назва}, Price: {ціна}, Quantity: {кількість}".

Додайте в клас також необхідні конструктори, деструктор та інші методи, які можуть знадобитись для роботи з товарами.

Напишіть програму, де ви використовуєте цей клас Product та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу Product і виконайте з ними операції об'єднання, віднімання, масштабування ціни, зменшення ціни, зміни кількості товару, порівняння на рівність та виведення на екран.

Варіант 15

Створіть клас Polynomial, який представляє поліном. Клас Polynomial повинен мати наступні властивості:

- Коефіцієнти поліному (coefficients) - список дійсних чисел, де кожен елемент представляє коефіцієнт при відповідному степені поліному.

В класі Polynomial перевантажте наступні оператори:

1. Оператор + для додавання двох поліномів. Результатом має бути новий об'єкт Polynomial, у якого коефіцієнти будуть сумою коефіцієнтів по відповідним степеням.
2. Оператор - для віднімання одного поліному від іншого. Результатом має бути новий об'єкт Polynomial, у якого коефіцієнти будуть різницею коефіцієнтів по відповідним степеням.
3. Оператор * для множення двох поліномів. Результатом має бути новий об'єкт Polynomial, у якого коефіцієнти будуть результатом множення коефіцієнтів по відповідним степеням та правильними степенями.
4. Оператор / для ділення одного поліному на інший. Результатом має бути новий об'єкт Polynomial, у якого коефіцієнти будуть результатом ділення коефіцієнтів по відповідним степеням та правильними степенями.
5. Оператор == для порівняння двох поліномів на рівність.
6. Оператор != для порівняння двох поліномів на нерівність.
7. Оператор << для виводу поліному у зрозумілому форматі. Виведіть поліном у форматі "a_n*x^n + a_(n-1)x^(n-1) + ... + a_1x^1 + a_0", де a_n, a_(n-1), ..., a_1, a_0 - коефіцієнти поліному, a n - степінь поліному.

Додайте в клас Polynomial необхідні конструктори, деструктор та інші методи, які можуть знадобитись для роботи з поліномами.

Напишіть програму, де ви використовуєте цей клас Polynomial та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу Polynomial і виконайте з ними операції додавання, віднімання, множення, ділення, порівняння на рівність та виведення на екран.

Варіант 16

Створіть клас Triangle, який представляє трикутник. Клас Triangle повинен мати наступні властивості та методи:

1. Сторони трикутника (side1, side2, side3) - дійсні числа, які представляють довжини сторін трикутника.
2. Периметр трикутника (perimeter) - метод, який обчислює та повертає периметр трикутника.
3. Площа трикутника (area) - метод, який обчислює та повертає площу трикутника за формулою Герона.

В класі Triangle перевантажте наступні оператори:

1. Оператор + для додавання двох трикутників. Результатом має бути новий об'єкт Triangle, в якому сторони трикутника є сумою відповідних сторін з двох трикутників.
2. Оператор - для віднімання одного трикутника від іншого. Результатом має бути новий об'єкт Triangle, в якому сторони трикутника є різницею відповідних сторін двох трикутників.
3. Оператор * для масштабування трикутника на заданий множник. Результатом має бути новий об'єкт Triangle, в якому довжини сторін трикутника множаться на заданий множник.
4. Оператор / для зменшення трикутника на заданий дільник. Результатом має бути новий об'єкт Triangle, в якому довжини сторін трикутника діляться на заданий дільник.
5. Оператор == для порівняння двох трикутників на рівність.
6. Оператор != для порівняння двох трикутників на нерівність.
7. Оператор << для виводу трикутника у зрозумілому форматі.

Створіть програму, де ви використовуєте клас Triangle та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу Triangle.

Не забудьте надати приклади вхідних даних (довжини сторін трикутників) та очікуваних результатів для кожної операції.

Варіант 17

Реалізуйте клас Circle, який представляє коло на площині. Клас повинен мати властивості для радіусу кола та центру кола (представленого об'єктом класу Point).

Додайте метод до класу Circle для обчислення площі кола. Формула для обчислення площі кола: π * r^2, де π - математична константа, а r - радіус кола.

1. Перевантажте оператор +, щоб додавати два об'єкти класу Circle. Результатом має бути новий об'єкт Circle, у якому радіус буде сумою радіусів двох кол і центром - центром першого кола.
2. Перевантажте оператор -, щоб віднімати одне коло від іншого. Результатом має бути новий об'єкт Circle, у якому радіус буде різницею радіусів двох кол і центром - центром першого кола.
3. Перевантажте оператор *, щоб масштабувати коло на заданий множник. Результатом має бути новий об'єкт Circle, у якому радіус буде добутком радіуса кола на заданий множник, а центр - центром початкового кола.
4. Перевантажте оператор /, щоб зменшувати коло на заданий дільник. Результатом має бути новий об'єкт Circle, у якому радіус буде діленням радіуса кола на заданий дільник, а центр - центром початкового кола.
5. Перевантажте оператор ==, щоб порівнювати два кола на рівність.
6. Перевантажте оператор !=, щоб порівнювати два кола на нерівність.

7. Перевантажте оператор <<, щоб виводити коло у зрозумілому форматі.

Створіть програму, де ви використовуєте клас Circle та перевірте роботу всіх перевантажених операторів, включаючи нові оператори +, -, *, /, ==, !=.

Не забудьте надати приклади вхідних даних (радіуси кол) та очікуваних результатів для кожної операції.

Варіант 18

Реалізуйте клас Sphere, який представляє сферу у тривимірному просторі. Клас повинен мати властивості для радіуса сфери та центру сфери (представленого об'єктом класу Point).

Додайте метод до класу Sphere для обчислення об'єму сфери. Формула для обчислення об'єму сфери: (4/3) * π * r^3, де π - математична константа, а r - радіус сфери.

1. Перевантажте оператор +, щоб додавати два об'єкти класу Sphere. Результатом має бути новий об'єкт Sphere, у якому радіус буде сумою радіусів двох сфер і центром - центром першої сфери.
2. Перевантажте оператор -, щоб віднімати одну сферу від іншої. Результатом має бути новий об'єкт Sphere, у якому радіус буде різницею радіусів двох сфер і центром - центром першої сфери.
3. Перевантажте оператор *, щоб масштабувати сферу на заданий множник. Результатом має бути новий об'єкт Sphere, у якому радіус буде добутком радіуса сфери на заданий множник, а центр - центром початкової сфери.
4. Перевантажте оператор /, щоб зменшувати сферу на заданий дільник. Результатом має бути новий об'єкт Sphere, у якому радіус буде діленням радіуса сфери на заданий дільник, а центр - центром початкової сфери.
5. Перевантажте оператор ==, щоб порівнювати дві сфери на рівність.
6. Перевантажте оператор !=, щоб порівнювати дві сфери на нерівність.
7. Перевантажте оператор <<, щоб виводити сферу на екран.

Напишіть програму, де ви використовуєте клас Sphere і перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу Sphere і виконайте з ними операції додавання, віднімання, масштабування та порівняння на рівність. Виведіть результати на екран.

Варіант 19

Створіть клас Line, який представляє пряму на площині. Пряма буде задаватись коефіцієнтами a, b і c у рівнянні ax + by + c = 0. Клас Line повинен мати методи для встановлення та отримання коефіцієнтів прямої, а також методи для виконання різних операцій над прямими. В класі Line перевантажте наступні оператори:

1. Оператор + для об'єднання двох прямих. Результатом має бути нова пряма, яка є об'єднанням обох початкових прямих.
2. Оператор - для перетину двох прямих. Результатом має бути точка перетину обох прямих або спеціальне значення, що вказує на те, що прямі не перетинаються.
3. Оператор == для порівняння двох прямих на рівність.
4. Оператор != для порівняння двох прямих на нерівність.
5. Оператор << для виводу рівняння прямої на екран у вигляді "ax + by + c = 0".

Напишіть програму, де ви використовуєте цей клас та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу Line і виконайте з ними операції об'єднання, перетину та порівняння. Виведіть результати на екран.

Варіант 20

Створіть клас Cube, який представляє куб у тривимірному просторі. Куб буде задаватись довжиною ребра. Клас Cube повинен мати методи для встановлення та отримання довжини ребра, а також методи для виконання різних операцій над кубами. В класі Cube перевантажте наступні оператори:

1. Оператор + для об'єднання двох кубів. Результатом має бути новий куб, який охоплює обидва початкові куби.
2. Оператор - для викреслення одного куба з іншого. Результатом має бути новий куб, який представляє область, що залишилась після викреслення.
3. Оператор * для масштабування куба. Передавайте дійсне число як аргумент, і результатом має бути новий куб, у якого довжина ребра помножена на цей масштабний коефіцієнт.
4. Оператор == для порівняння двох кубів на рівність.
5. Оператор != для порівняння двох кубів на нерівність.
6. Оператор << для виводу куба на екран у вигляді "Cube: a", де a - довжина ребра куба.

Напишіть програму, де ви використовуєте цей клас та перевірте роботу всіх перевантажених операторів. Створіть декілька об'єктів класу Cube і виконайте з ними операції об'єднання, викреслення, масштабування та порівняння. Виведіть результати на екран.

Лабораторна робота №6 Шаблони в С++

Мета: ознайомитись з основними поняттями шаблони та навчитись їх програмно реалізовувати мовою С++.

Теоретична частина

Шаблони є потужним інструментом у мові програмування С++. Вони дозволяють програмістам створювати загальні, параметризовані класи та функції, які можуть працювати з різними типами даних. Використання шаблонів дозволяє писати більш загальні й універсальні програми, що збільшує перевикористання коду, покращує його зрозумілість та підтримуваність.

Шаблони класів дозволяють визначити загальну структуру класу, незалежну від конкретного типу даних. Вони визначаються за допомогою ключового слова "template" і параметрів шаблону, які можуть бути типами даних, константами або іншими шаблонами. Наприклад, ось простий шаблон класу "Stack", який реалізує структуру стеку:

```
template <typename T>
class Stack {
private:
    std::vector<T> elements; // Зберігає елементи стеку

public:
    void push(const T& element) {
        elements.push_back(element);
    }

    void pop() {
        elements.pop_back();
    }

    T& top() {
        return elements.back();
    }

    bool empty() const {
        return elements.empty();
    }
};
```

У цьому прикладі тип даних "T" є параметром шаблону. Клас "Stack" може бути використаний для створення стеку будь-якого типу даних. Наприклад:

```
Stack<int> intStack; // Стек цілих чисел
Stack<std::string> stringStack; // Стек рядків
```

Крім шаблонів класів, в С++ також є шаблони функцій. Вони дозволяють визначати функції, які працюють з різними типами даних. Шаблони функцій використовують таку саму синтаксичну конструкцію з ключовим словом

"template" і параметрами шаблону. Наприклад, ось проста шаблонна функція "max", яка повертає більший із двох переданих значень:

```
template <typename T>
T max(T a, T b) {
    return (a > b) ? a : b;
}
```

Ця функція може бути використана для порівняння значень будь-якого типу даних, для якого визначені оператори порівняння.

```
int result1 = max(10, 20); // Поверне 20
double result2 = max(3.14, 2.71); // Поверне 3.14
```

Шаблони в C++ також підтримують спеціалізацію, що дозволяє визначати варіанти шаблону для конкретних типів даних або спеціальних випадків. Зазвичай спеціалізацію використовують для оптимізації або визначення специфічної поведінки для певного типу даних.

Шаблони в C++ - це потужний інструмент, який дозволяє створювати загальні та універсальні програми. Вони забезпечують перевикористання коду, покращують зрозумілість і підтримуваність програм, а також дають можливість програмістам працювати з різними типами даних без необхідності дублювання коду.

Завдання

Варіант 1

Створи програму для керування бібліотекою книжок. Кожна книжка має заголовок, автора та рік видання. Необхідно створити клас Book, що містить ці дані.

Крім того, створи шаблонний клас Library, який представляє колекцію книжок. Цей клас має методи для додавання книжки, видалення книжки за заголовком, виведення списку всіх книжок та пошуку книжок за автором.

Забезпеч універсальність класу Library, використовуючи шаблонний параметр для типу книжки. Це дозволить тобі працювати з різними типами книжок.

Для демонстрації роботи програми:

1. Створи об'єкт Library для зберігання книжок.
2. Додай до бібліотеки декілька книжок різних типів, наприклад, художніх, наукових, документальних тощо.
3. Виведи на екран список всіх книжок в бібліотеці.
4. Знайди та виведи на екран усі книжки певного автора.
5. Видали з бібліотеки книжку за заголовком.
6. Знову виведи на екран оновлений список книжок в бібліотеці.

Не забудь використовувати шаблонний клас Library для зберігання книжок будь-якого типу. Також, забезпеч, щоб клас Book мав методи доступу до своїх полів (заголовок, автор, рік видання).

Варіант 2

Створи програму для керування студентським списком. Кожен студент має ім'я, прізвище та середній бал. Необхідно створити клас Student, який містить ці дані.

Крім того, створи шаблонний клас StudentList, який представляє колекцію студентів. Цей клас має методи для додавання студента, видалення студента за прізвищем, виведення списку всіх студентів та пошуку студентів за середнім балом.

Забезпеч універсальність класу StudentList, використовуючи шаблонний параметр для типу студента. Це дозволить тобі працювати з різними типами студентів (наприклад, інженерний факультет, медичний факультет тощо).

Для демонстрації роботи програми:

1. Створи об'єкт StudentList для зберігання студентів.
2. Додай до списку декілька студентів різних факультетів.
3. Виведи на екран список всіх студентів у списку.
4. Знайди та виведи на екран усіх студентів з певним середнім балом.
5. Видали зі списку студента за прізвищем.
6. Знову виведи на екран оновлений список студентів у списку.

Не забудь використовувати шаблонний клас StudentList для зберігання студентів будь-якого типу. Також, забезпеч, щоб клас Student мав методи доступу до своїх полів (ім'я, прізвище, середній бал).

Варіант 3

Створи програму для керування банківським рахунком клієнтів. Кожен клієнт має ім'я, прізвище та баланс на рахунку. Необхідно створити клас Customer, який містить ці дані.

Крім того, створи шаблонний клас BankAccount, який представляє банківський рахунок. Цей клас має методи для додавання коштів на рахунок, зняття коштів з рахунку, переказу коштів між рахунками та виведення балансу рахунку.

Забезпеч універсальність класу BankAccount, використовуючи шаблонний параметр для типу клієнта. Це дозволить тобі працювати з різними типами клієнтів (наприклад, фізичні особи, юридичні особи тощо).

Для демонстрації роботи програми:

1. Створи об'єкт BankAccount для керування рахунком клієнта.
2. Додай кошти на рахунок за допомогою методу deposit().
3. Зніми кошти з рахунку за допомогою методу withdraw().
4. Перекажи кошти з одного рахунку на інший за допомогою методу transfer().

5. Виведи на екран баланс рахунку за допомогою методу getBalance().

Не забудь використовувати шаблонний клас BankAccount для керування рахунками різних типів клієнтів. Також, забезпеч, щоб клас Customer мав методи доступу до своїх полів (ім'я, прізвище, баланс).

Варіант 4

Створи програму для керування складом товарів у магазині. Кожен товар має назву, ціну та кількість на складі. Необхідно створити клас Product, який містить ці дані.

Крім того, створи шаблонний клас Inventory, який представляє інвентар магазину. Цей клас має методи для додавання товару на склад, видалення товару зі складу за назвою, виведення списку всіх товарів та пошуку товару за ціною.

Забезпеч універсальність класу Inventory, використовуючи шаблонний параметр для типу товару. Це дозволить тобі працювати з різними типами товарів (наприклад, продукти харчування, електроніка, одяг тощо).

Для демонстрації роботи програми:

1. Створи об'єкт Inventory для зберігання товарів на складі.
2. Додай до інвентаря декілька товарів різних типів.
3. Виведи на екран список всіх товарів на складі.
4. Знайди та виведи на екран всі товари за певною ціною.
5. Видали з інвентаря товар за назвою.
6. Знову виведи на екран оновлений список товарів на складі.

Не забудь використовувати шаблонний клас Inventory для зберігання товарів будь-якого типу. Також, забезпеч, щоб клас Product мав методи доступу до своїх полів (назва, ціна, кількість).

Варіант 5

Створи програму для керування рестораном. У ресторані є різні страви, які клієнти можуть замовити. Кожна страва має назву, опис та ціну. Необхідно створити клас Dish, який містить ці дані.

Крім того, створи шаблонний клас Menu, який представляє меню ресторану. Цей клас має методи для додавання страви до меню, видалення страви за назвою, виведення списку всіх страв та пошуку страви за ціною.

Забезпеч універсальність класу Menu, використовуючи шаблонний параметр для типу страви. Це дозволить тобі працювати з різними типами страв (наприклад, страви із м'ясом, страви для вегетаріанців тощо).

Для демонстрації роботи програми:

1. Створи об'єкт Menu для зберігання страв.
2. Додай до меню декілька різних страв.
3. Виведи на екран список всіх страв у меню.
4. Знайди та виведи на екран всі страви за певною ціною.
5. Видали з меню страву за назвою.

6. Знову виведи на екран оновлений список страв у меню.

Не забудь використовувати шаблонний клас Menu для зберігання страв будь-якого типу. Також, забезпеч, щоб клас Dish мав методи доступу до своїх полів (назва, опис, ціна).

Варіант 6

Створи програму для керування туристичним агентством. Туристичне агентство пропонує різні тури та послуги для подорожей. Кожен тур має назву, опис, ціну та тривалість. Необхідно створити клас Tour, який містить ці дані.

Крім того, створи шаблонний клас TravelAgency, який представляє туристичне агентство. Цей клас має методи для додавання туру до агентства, видалення туру за назвою, виведення списку всіх доступних турів та пошуку туру за ціною.

Забезпеч універсальність класу TravelAgency, використовуючи шаблонний параметр для типу туру. Це дозволить тобі працювати з різними типами турів (наприклад, пляжні тури, екскурсійні тури, гірськолижні тури тощо).

Для демонстрації роботи програми:

1. Створи об'єкт TravelAgency для зберігання турів.
2. Додай до агентства декілька різних турів.
3. Виведи на екран список всіх доступних турів.
4. Знайди та виведи на екран всі тури за певною ціною.
5. Видали з агентства тур за назвою.
6. Знову виведи на екран оновлений список доступних турів.

Не забудь використовувати шаблонний клас TravelAgency для зберігання турів будь-якого типу. Також, забезпеч, щоб клас Tour мав методи доступу до своїх полів (назва, опис, ціна, тривалість).

Варіант 7

Створи програму для керування автомобільним сервісним центром. Сервісний центр займається обслуговуванням автомобілів та наданням різних послуг. Кожен автомобіль має марку, модель та рік випуску. Необхідно створити клас Car, який містить ці дані.

Крім того, створи шаблонний клас ServiceCenter, який представляє сервісний центр. Цей клас має методи для додавання автомобіля до списку обслуговування, видалення автомобіля за маркою, виведення списку всіх автомобілів у сервісному центрі та пошуку автомобіля за роком випуску.

Забезпеч універсальність класу ServiceCenter, використовуючи шаблонний параметр для типу автомобіля. Це дозволить тобі працювати з різними типами автомобілів (наприклад, легкові, вантажні, спортивні тощо).

Для демонстрації роботи програми:

1. Створи об'єкт ServiceCenter для зберігання автомобілів.
2. Додай до сервісного центру декілька різних автомобілів.
3. Виведи на екран список всіх автомобілів у сервісному центрі.

4. Знайди та виведи на екран всі автомобілі за певним роком випуску.

5. Видали з сервісного центру автомобіль за маркою.

6. Знову виведи на екран оновлений список автомобілів у сервісному центрі.

Не забудь використовувати шаблонний клас ServiceCenter для зберігання автомобілів будь-якого типу. Також, забезпеч, щоб клас Car мав методи доступу до своїх полів (марка, модель, рік випуску).

Варіант 8

Створи програму для керування музичним магазином. Музичний магазин продає різні музичні інструменти та аксесуари. Кожен інструмент має назву, тип та ціну. Необхідно створити клас Instrument, який містить ці дані.

Крім того, створи шаблонний клас MusicStore, який представляє музичний магазин. Цей клас має методи для додавання інструменту до асортименту магазину, видалення інструменту за назвою, виведення списку всіх доступних інструментів та пошуку інструменту за типом.

Забезпеч універсальність класу MusicStore, використовуючи шаблонний параметр для типу інструменту. Це дозволить тобі працювати з різними типами інструментів (наприклад, гітари, барабани, клавішні тощо).

Для демонстрації роботи програми:

1. Створи об'єкт MusicStore для зберігання інструментів.

2. Додай до магазину декілька різних інструментів.

3. Виведи на екран список всіх доступних інструментів у магазині.

4. Знайди та виведи на екран всі інструменти певного типу.

5. Видали з магазину інструмент за назвою.

6. Знову виведи на екран оновлений список доступних інструментів у магазині.

Не забудь використовувати шаблонний клас MusicStore для зберігання інструментів будь-якого типу. Також, забезпеч, щоб клас Instrument мав методи доступу до своїх полів (назва, тип, ціна).

Варіант 9

Створи програму для керування спортивним клубом. Спортивний клуб має різні секції, кожна з яких спеціалізується на певному виді спорту. Кожна секція має назву, тренера та кількість учасників. Необхідно створити клас Section, який містить ці дані.

Крім того, створи шаблонний клас SportsClub, який представляє спортивний клуб. Цей клас має методи для додавання секції до клубу, видалення секції за назвою, виведення списку всіх доступних секцій та пошуку секції за кількістю учасників.

Забезпеч універсальність класу SportsClub, використовуючи шаблонний параметр для типу секції. Це дозволить тобі працювати з різними видами спорту (наприклад, футбол, баскетбол, плавання тощо).

Для демонстрації роботи програми:

1. Створи об'єкт SportsClub для зберігання секцій.

2. Додай до клубу декілька різних секцій.

3. Виведи на екран список всіх доступних секцій у клубі.

4. Знайди та виведи на екран всі секції з певною кількістю учасників.

5. Видали з клубу секцію за назвою.

6. Знову виведи на екран оновлений список доступних секцій у клубі.

Не забудь використовувати шаблонний клас SportsClub для зберігання секцій будь-якого типу. Також, забезпеч, щоб клас Section мав методи доступу до своїх полів (назва, тренер, кількість учасників).

Варіант 10

Створи програму для керування онлайн-курсами. Кожен курс має назву, викладача та кількість студентів. Необхідно створити клас Course, який містить ці дані.

Крім того, створи шаблонний клас OnlineLearningPlatform, який представляє платформу для онлайн-навчання. Цей клас має методи для додавання курсу до платформи, видалення курсу за назвою, виведення списку всіх доступних курсів та пошуку курсу за кількістю студентів.

Забезпеч універсальність класу OnlineLearningPlatform, використовуючи шаблонний параметр для типу курсу. Це дозволить тобі працювати з різними типами курсів (наприклад, програмування, мови, маркетинг тощо).

Для демонстрації роботи програми:

1. Створи об'єкт OnlineLearningPlatform для зберігання курсів.

2. Додай до платформи декілька різних курсів.

3. Виведи на екран список всіх доступних курсів на платформі.

4. Знайди та виведи на екран всі курси з певною кількістю студентів.

5. Видали з платформи курс за назвою.

6. Знову виведи на екран оновлений список доступних курсів на платформі.

Не забудь використовувати шаблонний клас OnlineLearningPlatform для зберігання курсів будь-якого типу. Також, забезпеч, щоб клас Course мав методи доступу до своїх полів (назва, викладач, кількість студентів).

Варіант 11

Розроби програму для керування готельним бронюванням. Готель має декілька типів номерів, кожен з яких має свою кількість ліжок, ціну та доступність. Необхідно створити клас Room, який містить ці дані.

Створи клас HotelBookingSystem, який буде відповідати за керування бронюванням номерів готелю. Клас має методи для додавання номеру до системи, видалення номеру за його ідентифікатором, виведення списку доступних номерів та пошуку вільних номерів за кількістю ліжок.

Забезпеч універсальність класу HotelBookingSystem, використовуючи шаблонний параметр для типу номеру. Це дозволить тобі працювати з різними типами номерів (наприклад, одномісні, двомісні, люкси тощо).

Для демонстрації роботи програми:

1. Створи об'єкт HotelBookingSystem для керування готельними номерами.
2. Додай до системи декілька різних типів номерів.
3. Виведи на екран список доступних номерів у готелі.
4. Знайди та виведи на екран всі вільні номери з певною кількістю ліжок.
5. Видали з системи номер за його ідентифікатором.
6. Знову виведи на екран оновлений список доступних номерів у готелі.

Не забудь використовувати шаблонний клас HotelBookingSystem для зберігання номерів будь-якого типу. Також, забезпеч, щоб клас Room мав методи доступу до своїх полів (кількість ліжок, ціна, доступність).

Варіант 12

Розроби програму для керування замовленнями в ресторані. Ресторан має різні страви у своєму меню, кожна з яких має назву, опис та ціну. Необхідно створити клас Dish, який містить ці дані.

Створи клас OrderManagementSystem, який буде відповідати за керування замовленнями у ресторані. Клас має методи для додавання страви до замовлення, видалення страви за її назвою, виведення списку всіх страв у замовленні та обчислення загальної суми замовлення.

Забезпеч універсальність класу OrderManagementSystem, використовуючи шаблонний параметр для типу страви. Це дозволить тобі працювати з різними типами страв (наприклад, перші страви, основні страви, десерти тощо).

Для демонстрації роботи програми:

1. Створи об'єкт OrderManagementSystem для керування замовленнями в ресторані.
2. Додай до замовлення декілька різних страв.
3. Виведи на екран список всіх страв у замовленні.
4. Видали з замовлення страву за її назвою.
5. Знову виведи на екран оновлений список страв у замовленні.
6. Обчисли загальну суму замовлення та виведи її на екран.

Не забудь використовувати шаблонний клас OrderManagementSystem для зберігання страв будь-якого типу. Також, забезпеч, щоб клас Dish мав методи доступу до своїх полів (назва, опис, ціна).

Варіант 13

Розроби програму для керування запозиченням та поверненням велосипедів у велопрокаті. Кожен велосипед має унікальний ідентифікатор, модель та стан (доступний або запозичений).

Створи клас Bike, який містить поля для зберігання цих даних. Також, додай методи для отримання та зміни стану велосипеда.

Створи клас BikeRentalSystem, який буде відповідати за керування запозиченням та поверненням велосипедів. Клас повинен мати методи для реєстрації нового велосипеда, запозичення велосипеда за його ідентифікатором,

повернення велосипеда за його ідентифікатором та виведення списку доступних велосипедів.

Забезпеч універсальність класу BikeRentalSystem, використовуючи шаблонний параметр для типу велосипеда. Це дозволить тобі працювати з різними типами велосипедів (наприклад, гірські велосипеди, шосейні велосипеди тощо).

Для демонстрації роботи програми:

1. Створи об'єкт BikeRentalSystem для керування запозиченням та поверненням велосипедів.
2. Зареєструй декілька нових велосипедів у системі.
3. Запозичи один з доступних велосипедів за його ідентифікатором.
4. Поверни запозичений велосипед за його ідентифікатором.
5. Виведи на екран список доступних велосипедів.

Не забудь використовувати шаблонний клас BikeRentalSystem для зберігання велосипедів будь-якого типу. Також, забезпеч, щоб клас Bike мав методи доступу до своїх полів (ідентифікатор, модель, стан).

Варіант 14

Створіть програму для керування готельним бронюванням. Кожне бронювання має включати ім'я гостя, номер кімнати, дату заїзду та виїзду. Необхідно створити клас Reservation, який містить ці дані.

Крім того, створіть шаблонний клас Hotel, який представляє колекцію бронювань. Цей клас має методи для додавання бронювання, видалення бронювання за номером кімнати, виведення списку всіх бронювань та пошуку бронювань за ім'ям гостя.

Забезпечте універсальність класу Hotel, використовуючи шаблонний параметр для типу бронювання. Це дозволить вам працювати з різними типами бронювань (наприклад, одномісні, двомісні, люксові тощо).

Для демонстрації роботи програми:

1. Створіть об'єкт Hotel для зберігання бронювань.
2. Додайте до готелю декілька бронювань різних типів, наприклад, одномісних, двомісних, люксових тощо.
3. Виведіть на екран список всіх бронювань у готелі.
4. Знайдіть та виведіть на екран усі бронювання певного гостя.
5. Видаліть з готелю бронювання за номером кімнати.
6. Знову виведіть на екран оновлений список бронювань у готелі.

Не забудьте використовувати шаблонний клас Hotel для зберігання бронювань будь-якого типу. Також, забезпечте, щоб клас Reservation мав методи доступу до своїх полів (ім'я гостя, номер кімнати, дата заїзду та виїзду).

Варіант 15

Створіть програму для керування списком контактів. Кожен контакт має включати ім'я, прізвище та номер телефону. Необхідно створити клас Contact, який містить ці дані.

Крім того, створіть шаблонний клас ContactList, який представляє колекцію контактів. Цей клас має методи для додавання контакту, видалення контакту за номером телефону, виведення списку всіх контактів та пошуку контактів за ім'ям або прізвищем.

Забезпечте універсальність класу ContactList, використовуючи шаблонний параметр для типу контакту. Це дозволить вам працювати з різними типами контактів (наприклад, особисті, службові, родинні тощо).

Для демонстрації роботи програми:

1. Створіть об'єкт ContactList для зберігання контактів.
2. Додайте до списку кілька контактів різних типів, наприклад, особистих, службових, родинних тощо.
3. Виведіть на екран список всіх контактів у списку.
4. Знайдіть та виведіть на екран усі контакти за певним ім'ям або прізвищем.
5. Видаліть зі списку контакт за номером телефону.
6. Знову виведіть на екран оновлений список контактів у списку.

Не забудьте використовувати шаблонний клас ContactList для зберігання контактів будь-якого типу. Також, забезпечте, щоб клас Contact мав методи доступу до своїх полів (ім'я, прізвище, номер телефону).

Варіант 16

Створіть програму для керування домашнім інвентарем. Кожен предмет має містити назву, опис та кількість. Необхідно створити клас Item, який містить ці дані.

Крім того, створіть шаблонний клас Inventory, який представляє колекцію предметів. Цей клас має методи для додавання предмету, видалення предмету за назвою, виведення списку всіх предметів та пошуку предметів за описом.

Забезпечте універсальність класу Inventory, використовуючи шаблонний параметр для типу предмету. Це дозволить вам працювати з різними типами предметів (наприклад, побутові, електроніка, спортивні речі тощо).

Для демонстрації роботи програми:

1. Створіть об'єкт Inventory для зберігання предметів.
2. Додайте до інвентаря кілька предметів різних типів, наприклад, побутових, електроніки, спортивних речей тощо.
3. Виведіть на екран список всіх предметів у інвентарі.
4. Знайдіть та виведіть на екран усі предмети з певним описом.
5. Видаліть з інвентаря предмет за назвою.
6. Знову виведіть на екран оновлений список предметів у інвентарі.

Не забудьте використовувати шаблонний клас Inventory для зберігання предметів будь-якого типу. Також, забезпечте, щоб клас Item мав методи доступу до своїх полів (назва, опис, кількість).

Варіант 17

Створіть програму для керування списком подій. Кожна подія має містити назву, дату та час проведення. Необхідно створити клас Event, який містить ці дані.

Крім того, створіть шаблонний клас EventManager, який дозволить додавати події, видаляти події за назвою, виводити список всіх подій та шукати події за датою.

Забезпечте універсальність класу EventManager, використовуючи шаблонний параметр для типу події. Це дозволить вам працювати з різними типами подій (наприклад, зустрічі, конференції, виставки тощо).

Для демонстрації роботи програми:

1. Створіть об'єкт EventManager для керування списком подій.
2. Додайте до списку кілька подій різних типів, наприклад, зустрічі, конференції, виставки тощо.
3. Виведіть на екран список всіх подій.
4. Знайдіть та виведіть на екран усі події, що відбудуться в певну дату.
5. Видаліть зі списку подію за назвою.

Знову виведіть на екран оновлений список подій.

Не забудьте використовувати шаблонний клас EventManager для керування подіями будь-якого типу. Також, забезпечте, щоб клас Event мав методи доступу до своїх полів (назва, дата, час проведення).

Варіант 18

Створіть програму для керування списком задач. Кожна задача має містити опис, пріоритет та статус (виконано/не виконано). Необхідно створити клас Task, який містить ці дані.

Крім того, створіть шаблонний клас TaskManager, який дозволить додавати задачі, видаляти задачі за описом, виводити список всіх задач та змінювати статус задачі.

Забезпечте універсальність класу TaskManager, використовуючи шаблонний параметр для типу задачі. Це дозволить вам працювати з різними типами задач (наприклад, робочі завдання, домашні обов'язки, особисті цілі тощо).

Для демонстрації роботи програми:

1. Створіть об'єкт TaskManager для керування списком задач.
2. Додайте до списку кілька задач різних типів, наприклад, робочі завдання, домашні обов'язки, особисті цілі тощо.
3. Виведіть на екран список всіх задач.
4. Змініть статус певної задачі на "виконано".
5. Видаліть зі списку задачу за описом.

6. Знову виведіть на екран оновлений список задач.

Не забудьте використовувати шаблонний клас TaskManager для керування задачами будь-якого типу. Також, забезпечте, щоб клас Task мав методи доступу до своїх полів (опис, пріоритет, статус).

Варіант 19

Створіть програму для керування списком покупок. Кожен товар має назву, кількість та ціну. Необхідно створити клас Product, який містить ці дані.

Крім того, створіть шаблонний клас ShoppingList, який дозволить додавати товари до списку покупок, видаляти товари за назвою, виводити список всіх товарів та обчислювати загальну суму покупки.

Забезпечте універсальність класу ShoppingList, використовуючи шаблонний параметр для типу товару. Це дозволить вам працювати з різними типами товарів (наприклад, продукти харчування, одяг, електроніка тощо).

Для демонстрації роботи програми:

1. Створіть об'єкт ShoppingList для керування списком покупок.
2. Додайте до списку декілька товарів різних типів, наприклад, продукти харчування, одяг, електроніка тощо.
3. Виведіть на екран список всіх товарів.
4. Видаліть зі списку товар за назвою.
5. Знову виведіть на екран оновлений список товарів.
6. Обчисліть загальну суму покупки і виведіть її на екран.

Не забудьте використовувати шаблонний клас ShoppingList для керування покупками будь-якого типу товару. Також, забезпечте, щоб клас Product мав методи доступу до своїх полів (назва, кількість, ціна).

Варіант 20

Створіть програму для керування списком друзів і їх контактною інформацією. Кожен друг має ім'я, прізвище та номер телефону. Необхідно створити клас Friend, який містить ці дані.

Крім того, створіть клас FriendList, який дозволяє додавати друзів до списку, видаляти друзів за ім'ям або прізвищем, виводити список всіх друзів та проводити пошук друзів за номером телефону.

Забезпечте універсальність класу FriendList, використовуючи шаблонний параметр для типу друзів. Це дозволить вам працювати з різними типами друзів (наприклад, колеги, сімейні члени, однокласники тощо).

Для демонстрації роботи програми:

1. Створіть об'єкт FriendList для керування списком друзів.
2. Додайте до списку декілька друзів з різними іменами та прізвищами.
3. Виведіть на екран список всіх друзів.
4. Видаліть зі списку друга за ім'ям або прізвищем.
5. Знову виведіть на екран оновлений список друзів.

6. Здійсніть пошук друга за номером телефону і виведіть результати на екран.

Не забудьте використовувати шаблонний клас FriendList для керування списком друзів будь-якого типу. Також, забезпечте, щоб клас Friend мав методи доступу до своїх полів (ім'я, прізвище, номер телефону).

Лабораторна робота №7 Контейнерні класи. Стандартна бібліотека шаблонів (STL) в C++.

Мета: ознайомитись Контейнерні класи та навчитись їх програмно реалізовувати мовою C++.

Теоретична частина

Контейнерний клас (або “клас-контейнер”) в мові C++ — це клас, призначений для зберігання і організації декількох об’єктів певного типу даних (користувацьких чи фундаментальних).

В мові програмування C++, контейнерні класи є класами, які призначені для зберігання та управління колекціями об'єктів. Вони надають зручний інтерфейс для додавання, видалення і доступу до елементів у колекції.

Контейнерні класи в C++ доступні завдяки бібліотеці стандартного шаблону (STL - Standard Template Library), яка включає в себе різні типи контейнерів, такі як вектори, списки, множини, асоціативні масиви тощо.

Основні типи контейнерів, які можна знайти в STL, включають наступні:

- Вектор (vector): Послідовний динамічний масив елементів.
- Список (list): Двоспрямований зв'язаний список елементів.
- Двійкове дерево (binary tree): Структура даних, де кожен вузол може мати не більше двох нащадків.
- Мапа (map): Асоціативний масив, що зберігає пари ключ-значення.
- Множина (set): Колекція унікальних елементів без поняття порядку.

Це лише кілька прикладів контейнерів, що надаються STL. Кожен контейнер має свої особливості і методи для додавання, видалення, пошуку та інших операцій, що можуть бути виконані з елементами колекції.

Використання контейнерних класів дозволяє забезпечити ефективне управління даними та спростити роботу з колекціями в мові програмування C++.

Наприклад використання контейнерного класу vector з бібліотеки STL в C++:

```
#include <iostream>
#include <vector>

int main() {
    // Створення об'єкту вектора для зберігання цілих чисел
    std::vector<int> numbers;

    // Додавання елементів до вектора
    numbers.push_back(10);
    numbers.push_back(20);
    numbers.push_back(30);

    // Виведення елементів вектора
    std::cout << "Елементи вектора: ";
```

```
        for (int i = 0; i < numbers.size(); ++i) {
            std::cout << numbers[i] << " ";
        }
        std::cout << std::endl;

        // Видалення елемента з вектора
        numbers.pop_back();

        // Оновлення значення елемента
        numbers[1] = 50;

        // Виведення оновлених елементів вектора
        std::cout << "Оновлені елементи вектора: ";
        for (int number : numbers) {
            std::cout << number << " ";
        }
        std::cout << std::endl;

        // Отримання розміру вектора
        std::cout << "Розмір вектора: " << numbers.size() << 
std::endl;

        return 0;
    }
```

У цьому прикладі спочатку створюється об'єкт вектора numbers, який зберігатиме цілі числа. За допомогою функції push_back() додаються числа 10, 20 і 30 до вектора. Потім елемент 30 видаляється за допомогою функції pop_back(), а елемент з індексом 1 (20) оновлюється на 50. Нарешті, елементи вектора виводяться на екран за допомогою циклу, а розмір вектора виводиться за допомогою функції size().

Вивід програми буде наступним:

```
Елементи вектора: 10 20 30
Оновлені елементи вектора: 10 50
Розмір вектора: 2
```

Завдання

Варіант 1

Розробіть систему керування магазином електроніки. У цій системі необхідно зберігати і керувати інформацією про продукти, які доступні у магазині. Для цього вам потрібно використовувати контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над списком продуктів.

- Додайте декілька продуктів до інвентаря, включаючи їхню назву, ціну та кількість на складі.
- Виведіть список продуктів у інвентарі.

- Оновіть кількість одного або декількох продуктів у списку.
- Виведіть оновлений список продуктів у інвентарі.
- Обчисліть загальну вартість усього інвентаря та виведіть її.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 2

Розробіть систему управління бібліотекою. У цій системі потрібно зберігати та керувати інформацією про книги, що знаходяться в бібліотеці. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над списком книг.

- Додайте декілька книг до бібліотеки, включаючи їх назву, автора, рік видання та ISBN.
- Виведіть список книг у бібліотеці.
- Видаліть одну або декілька книг зі списку за ISBN.
- Проведіть пошук книг за автором і виведіть результат.
- Проведіть пошук книг за роком видання і виведіть результат.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 3

Розробіть систему управління завданнями. У цій системі потрібно зберігати та керувати списком завдань. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над списком завдань.

- Додайте декілька завдань до списку, включаючи їх назву, опис та дедлайн.
- Виведіть список завдань.
- Видаліть одне або декілька завдань зі списку за назвою.
- Виведіть оновлений список завдань.
- Виведіть список завдань з певним дедлайном.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 4

Розробіть систему управління складом магазину автозапчастин. У цій системі потрібно зберігати та керувати інформацією про наявні автозапчастини. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над списком запчастин.

- Додайте декілька запчастин до складу, включаючи їх назву, виробника, ціну та кількість на складі.
- Виведіть список запчастин на складі.

- Видаліть одну або декілька запчастин зі складу за назвою.
- Оновіть ціну та кількість для певної запчастини.
- Виведіть оновлений список запчастин на складі.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 5

Розробіть систему управління студентськими курсами. У цій системі потрібно зберігати та керувати інформацією про курси та студентів, які до них записалися. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними. Використайте знання набуті в минулих заняттях для побудови програмного продукту.

- Додайте декілька курсів до списку, включаючи їх назву, викладача та кількість годин.
- Виведіть список курсів та кількості записаних студентів на кожний курс.
- Видаліть один або декілька курсів зі списку за назвою.
- Запишіть декілька студентів на певні курси.
- Виведіть оновлений список курсів та кількості записаних студентів на кожний курс.
- Виведіть список студентів, записаних на певний курс.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 6

Розробіть систему керування запасами продуктів у супермаркеті. У цій системі потрібно зберігати та керувати інформацією про наявні продукти, їх кількість та ціни. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними.

- Додайте декілька продуктів до списку, включаючи їх назву, категорію, ціну та кількість на складі.
- Виведіть повний список продуктів на складі.
- Видаліть один або декілька продуктів зі списку за назвою.
- Оновіть ціну та кількість для певного продукту.
- Виведіть оновлений список продуктів на складі.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 7

Розробіть систему керування готелями. У цій системі потрібно зберігати та керувати інформацією про доступні номери, резервації та клієнтів.

Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними.

- Додайте декілька кімнат до списку доступних кімнат, включаючи їх номер, категорію, ціну та доступність.
- Виведіть список доступних кімнат.
- Здійсніть резервацію для декількох кімнат, включаючи номер кімнати, дату початку та закінчення резервації та кількість гостей.
- Виведіть список всіх резервацій.
- Скасуйте резервацію для певної кімнати за номером.
- Виведіть оновлений список доступних кімнат.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 8

Розробіть систему керування поштовими відправленнями. У цій системі потрібно зберігати та керувати інформацією про відправлення, включаючи даних про відправника, одержувача, вагу та тип поштового відправлення. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними.

- Додайте декілька відправлень до списку, включаючи дані про відправника, одержувача, вагу та тип відправлення.
- Виведіть список всіх відправлень.
- Видаліть певне відправлення за номером.
- Оновіть вагу та тип для певного відправлення.
- Виведіть оновлений список всіх відправлень.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 9

Розробіть систему керування студентськими оцінками. У цій системі потрібно зберігати та керувати оцінками студентів за різними предметами. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними.

- Додайте декілька студентів до списку, включаючи їх ім'я, прізвище та початкові оцінки.
- Виведіть список оцінок всіх студентів.
- Видаліть певного студента за ім'ям та прізвищем.
- Додайте оцінку певному студенту.

- Видаліть оцінку у певного студента.
- Виведіть оновлений список оцінок всіх студентів.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 10

Розробіть систему керування замовленнями у ресторані. У цій системі потрібно зберігати та обробляти дані про замовлення страв, включаючи назви страв, їх кількість та ціну. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними.

- Додайте декілька страв до замовлення, вказавши їх назви та ціни.
- Виведіть загальну суму замовлення.
- Видаліть певну страву зі замовлення за назвою.
- Додайте інші страви до замовлення.
- Виведіть оновлену загальну суму замовлення.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 11

Розробіть систему керування фітнес-клубом. У цій системі потрібно зберігати та обробляти дані про членів клубу, включаючи їх особисту інформацію, абонементи та тренування. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними.

- Додайте декілька членів клубу та їхню особисту інформацію.
- Додайте абонементи для певних членів клубу.
- Додайте тренування та запишіть деяких членів клубу на тренування.
- Видаліть певного члена клубу та його абонементи.
- Видаліть тренування з програми клубу.
- Відмініть запис члена клубу на певне тренування.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 12

Розробіть систему керування музичним фестивалем. У цій системі потрібно зберігати та обробляти дані про виконавців, концерти та квитки на фестиваль. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними.

- Додайте декілька виконавців на фестиваль та їхню інформацію.

- Додайте концерти на фестиваль та квитки для певних концертів.
- Забронюйте квитки на фестиваль.
- Видаліть певного виконавця з фестивалю та його концерти.
- Видаліть концерти з програми фестивалю.
- Скасуйте бронювання квитків на фестиваль.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 13

Розробіть систему управління кінотеатром. У цій системі потрібно зберігати та обробляти дані про фільми, сеанси та квитки в кінотеатрі. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними.

- Додайте декілька фільмів до кінотеатру та їхню інформацію.
- Додайте сеанси фільмів до кінотеатру.
- Додайте квитки на певні сеанси.
- Забронюйте квитки в кінотеатрі.
- Видаліть певний фільм з кінотеатру та його сеанси.
- Видаліть сеанси зі списку сеансів кінотеатру.
- Скасуйте бронювання квитків в кінотеатрі.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 14

Розробіть систему управління туристичним агентством. У цій системі потрібно зберігати та обробляти дані про тури, клієнтів та бронювання. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними.

- Додайте декілька турів до агентства та їхню інформацію.
- Додайте клієнтів до агентства та їхню інформацію.
- Здійсніть бронювання турів для певних клієнтів.
- Скасуйте бронювання турів.
- Видаліть певний тур з агентства та всі бронювання, пов'язані з цим туром.
- Видаліть певного клієнта з агентства та всі його бронювання.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 15

Розробіть систему управління громадським парком. У цій системі потрібно зберігати та обробляти дані про відвідувачів, події та ресурси парку. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними.

- Додайте декілька відвідувачів до парку та їхню інформацію.
- Додайте події, що відбуваються у парку, та їхню інформацію.
- Додайте ресурси, доступні у парку, та їхню інформацію.
- Видаліть певного відвідувача з парку та всі дані, пов'язані з ним.
- Видаліть певну подію з парку та всі дані, пов'язані з нею.
- Видаліть певний ресурс з парку.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 16

Розробіть систему для управління агротехнічними операціями на фермі. Фермери можуть використовувати цю систему для планування та відстеження своїх сільськогосподарських дійств. Розробіть класи та функціонал для збереження та обробки даних про культури, польові операції та урожайність.

- Додайте декілька культур та їхню інформацію.
- Додайте польові операції, що виконує фермер, та їхню інформацію.
- Додайте урожайності культур та їхню інформацію.
- Видаліть певну культуру та всі дані, пов'язані з нею.
- Видаліть певну польову операцію за типом та датою.
- Видаліть певну урожайність культури.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 17

Розробіть систему керування медичним центром. У цій системі потрібно зберігати та обробляти дані про пацієнтів, лікарів, медичні записи та призначення лікування. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними.

- Додайте декілька пацієнтів та їхню особисту інформацію.
- Додайте лікарів до медичного центру.
- Додайте медичні записи для певних пацієнтів.
- Видаліть певного пацієнта та його медичні записи.
- Видаліть лікаря з медичного центру.

- Призначте лікування та оновіть діагноз для певного пацієнта.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 18

Розробіть систему керування таксі-службою. У цій системі потрібно зберігати та обробляти дані про таксі, водіїв та замовлення на перевезення пасажирів. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними.

- Додайте декілька таксі до системи.
- Додайте водіїв та їхню особисту інформацію.
- Розмістіть замовлення на перевезення пасажира для певних таксі.
- Скасуйте замовлення на перевезення пасажира.
- Отримайте список доступних таксі.
- Отримайте список замовлень на перевезення пасажира для певного водія.

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 19

Розробіть систему центру реєстрації автомобілів. У цій системі потрібно зберігати та обробляти дані про власників автомобілів, автомобілі та їх реєстраційні номери. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними.

- Додайте декілька власників автомобілів та їхню особисту інформацію.
- Додайте автомобілі для певних власників.
- Отримайте дані про власника автомобіля та автомобіль за реєстраційним номером.
- Видаліть певного власника автомобіля та його автомобілі.
- Видаліть автомобіль з реєстраційною номером.
- Зверніть увагу, що для збереження власників автомобілів та автомобілів можна

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Варіант 20

Розробіть систему керування центром зайнятості. У цій системі потрібно зберігати та обробляти дані про безробітних осіб, вакансії та

працевлаштування. Використовуйте контейнерні класи зі стандартної бібліотеки шаблонів (STL) для збереження та операцій над даними.

- Додайте декілька безробітних осіб та їхню особисту інформацію.
- Додайте вакансії для певних галузей.
- Знайдіть відповідності між безробітними особами та вакансіями.
- Виведіть результати збігів.
- Видаліть певну безробітну особу та вакансію.
- Зверніть увагу, що для збереження безробітних осіб та вакансій можна використовувати

Ви маєте розширити програму, додати додаткові методи та функціональні можливості, які вам здаються відповідними.

Список використаної літератури

1. B. Stroustrup: The C++ Programming Language (Fourth Edition). May 2013. Addison Wesley. Reading Mass. USA. May 2013. ISBN 0-321-56384-0. 1360 pages. Softcover, hardcover, and electronic versions.
2. Ira Pohl: Object-oriented programming using C++. 1997. Addison-Wesley. ISBN 978-0201895506. 543 pages.
3. B. Stroustrup: Programming -- Principles and Practice Using C++. December 2008. Addison-Wesley. ISBN 978-0321543721. 1264 pages. Softcover.
4. B. Stroustrup: Programming -- Principles and Practice Using C++ (Second Edition). May 2014. Addison-Wesley. ISBN 978-0321992789. 1312 pages. Softcover and electronic versions.
5. B. Stroustrup: A Tour of C++ (Second Edition). July 2018. Addison-Wesley. ISBN 978-0-13-499783-4. 240 pages. Softcover and electronic versions.
6. Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Conallen, Kelli A. Houston / Object-Oriented Analysis and Design with Applications (3rd Edition) - 2007/ 720 p. ISBN 978-5-8459-1401-9, 0-201-89551-X
7. Grady Booch James Rumbaugh Ivar Jacobson Б90 Язык UML. Руководство пользователя. 3-е изд.: Пер. с англ. Мухин Н. – М.: ДМК Пресс, 2016. – 496 с.: ил.
8. Rainer Grimm. C++ Core Guidelines. Addison-Welsey Professional. 2022. 403 c.
9. Bill Weinman. C++20 STL Cookbook. Packt Publishing. 2022. 450 c.
10. Ткачук В. М. Програмування на С++ : Лабораторний практикум / В. М. Ткачук. – Івано-Франківськ : Видавництво Прикарпатського національного університету імені Василя Стефаника, 2011. – 160 с.
11. Жуковський С.С., Вакалюк Т.А. Програмування мовою С++. Структурне програмування (лабораторний практикум). Навчальний посібник для студентів фізико-математичного факультету. – Житомир: Вид-во ЖДУ, 2011. – 92 с. (видання друге, перероблене та доповнене).
12. Грицюк Ю.І., Рак Т.Є. Г 85 Об'єктно-орієнтоване програмування мовою С++ : навчальний посібник. – Львів : Вид-во Львівського ДУ БЖД, 2011. – 404 с. – Статистика: іл. 18, табл. 12, бібліогр. 34. ISBN 978-966-3466-86-3BN 978-966-3466-86-3
13. Adrian Ostrowski, Piotr Gaczkowski. Software Architecture with C++. Packt Publishing. 2021. 522 c.

Інформаційні ресурси

14. Курс «Об’єктно-орієнтоване програмування» на сервері дистанційної освіта ЦНТУ. – URL: https://moodle.kntu.kr.ua/course/view.php?id=1046
15. Онлайн-курси Prometheus. – URL: https://prometheus.org.ua/

16. Онлайн-курси Coursera. – URL: https://www.coursera.org
17. Академія Cisco. – URL: https://www.netacad.com
18. Он-лайн ресурс з інформаційних технологій. – URL:https://habr.com
19. Он-лайн ресурс з інформаційних технологій. – URL:https://dou.ua/
20. Пошукова система. – URL:https://www.google.com/
21. Он-лайн ресурс перегляду відеоуроків.– URL:https://www.youtube.com
22. Он-лайн ресурс LEARN C++. – URL: https://www.learncpp.com/
23. Он-лайн ресурс aCode .– URL: https://acode.com.ua/