

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з дисципліни
«Сучасні методи програмування»
для студентів спеціальності 123 Комп'ютерна інженерія
усіх форм навчання.
Об'єктно-орієнтоване програмування мовою C++

Методичні вказівки до виконання лабораторних робіт з дисципліни «Сучасні методи програмування» для студентів спеціальності 123 Комп'ютерна інженерія усіх форм навчання. Об'єктно-орієнтоване програмування мовою C++ / Укл.: Р. К. Кудерметов, О. В. Польська, Н. А. Куликовська — Запоріжжя: НУ «Запорізька політехніка», 2024 — 70 с.

Укладачі: Р. К. Кудерметов, доцент, к.т.н.
О. В. Польська, ст. викладач
Н. А. Куликовська, ст. викладач

Рецензент: М. Ю. Тягунова, доцент, к.т.н.

Відповідальний
за випуск: О. В. Польська, ст. викладач

Затверджено:
на засіданні кафедри
«Комп'ютерні системи та мережі»
Протокол № 2 від 26 вересня 2024 р.

Рекомендовано до видання:
НМК факультету КНТ
Протокол № 2 від 28 жовтня 2024 р.

Зміст

1	Введення в ООП, елементи мови C++	4
1.1	Розробка класів об'єктів у статичній пам'яті	4
1.1.1	Програма: клас Food	4
1.2	Вказівники на об'єкти	6
1.2.1	Програма: клас Student	7
1.3	Багатофайловий проект	10
1.3.1	Багатофайлова програма: класи Student та Group	11
1.4	Зміст звіту	16
1.5	Контрольні питання	16
2	Дружні функції та класи. Перевантаження операцій	18
2.1	Дружні функції та класи	18
2.1.1	Програма: класи Student , Professor , Dean . .	18
2.1.2	Програма: класи Array , Sort	21
2.2	Перевантаження операцій	25
2.2.1	Програма: клас Folder	25
2.2.2	Програма: клас Matrix	28
2.3	Зміст звіту	37
2.4	Контрольні питання	38
3	Спадкування в C++	40
3.1	Прості приклади спадкування	40
3.1.1	Програма: класи Abstract , ResumeEU , ResumeUSA	43
3.2	Конструктори при спадкуванні	50
3.2.1	Програма: класи Text , Book , Novel тощо	50
3.3	Віртуальні функції та абстрактні класи	56
3.3.1	Програма: класи Bank , Client , BusinessLogic	57
3.4	Зміст звіту	68
3.5	Контрольні питання	69
	Рекомендована література	70

ЛАБОРАТОРНА РОБОТА 1

Введення в ООП, елементи мови C++

Мета роботи — вивчити структурований похідний тип клас мови C++, способи керування видимістю компонентів класу та доступу до них, визначення компонентних функцій класу, поняття конструктора, деструктора, вказівника на об'єкт класу, статичного члена класу.

1.1 Розробка класів об'єктів у статичній пам'яті

1.1.1 Програма: клас Food

У програмі (див. лістинг 1.1) демонструється синтаксис та прийом роботи з класом і його компонентами, такими як змінні-члени та функції-члени (методи) класу:

- у рядках 5-21 *визначено* клас **Food** (деякий *класифікатор* продуктів). Елементами цього класу є відкриті змінні-члени **name** і **price**, та закритий — **weight**;

- у класі визначено конструктор **Food(string,float,float)** (рядки 9-13), призначений для ініціалізації змінних-членів екземплярів класу;

- у класі також визначено відкритий метод **getWeight()** (рядки 15-17), за допомогою якого можна отримати доступ до закритого елемента **weight**;

- у методі **main()** продемонстровано способи створення екземплярів класу (об'єктів) (рядки 26, 27) та масиву екземплярів класу (рядок 31), а також доступ до змінних-членів (рядки 28, 34) та функцій-членів класу (рядки 29, 35).

Лістинг 1.1 Об'єкти статичної пам'яті

```
1  #include <iostream>
2
3  using namespace std;
4
```

```

5  class Food{
6  public:
7      string name;
8      float price;
9      Food (string s, float p, float w) {
10         name=s;
11         price = p;
12         weight = w;
13     }
14
15     float getWeight(){
16         return weight;
17     }
18
19     private:
20     float weight;
21 };
22 int main() {
23     float totalWeight = 0;
24     float totalPrice = 0;
25
26     Food orange1("Valencia", 21, 96);
27     Food orange2("Tarocco", 22, 105);
28     cout<<"totalPriceOranges = " << orange1.price
29         + orange2.price << endl;
30
31     cout<<"totalWeightOranges = " <<
32         orange1.getWeight() +
33         orange2.getWeight()<<endl;
34
35     Food apple[3]={Food("Golden",10.50,112),
36                     Food("McIntosh",11.99,100),
37                     Food("Jonathan",12,120)};
38
39     for(int i = 0; i < 3; i++){
40         totalPrice += apple[i].price;
41         totalWeight += apple[i].getWeight();
42     }
43
44     cout<<"totalPriceApples = " << totalPrice <<
45         endl;

```

```

39     cout<<"totalWeightApples = " << totalWeight;
40     return 0;
41 }

```

Завдання до програми

1. Реалізуйте та вивчіть програму з лістингу 1.1. Виконайте в програмі наступні завдання та продемонструйте її працездатність із внесеними змінами:

- додайте до головної функції `main()` фрагмент коду, за допомогою якого спробуйте отримати безпосередній доступ до закритого елементу класу `weight`. Зверніть увагу на повідомлення компілятора під час компіляції такого коду;

- додайте до програми фрагмент коду, що створює ще один вид екземплярів класу (наприклад, банани), і працює з їх змінними та функціями-членами. Реалізуйте підрахунок та вивід на екран загальної вартості та ваги продуктів;

- додайте до програми код, який застосовує операцію `sizeof` до класу, до екземпляра класу. Поясніть отриманий результат;

2. Розробіть та продемонструйте роботу власного прикладу програми, у якому має бути клас `Student` з такими компонентами:

- відкриті змінні-члени: ім'я та номер групи;
- закриті змінні-члени: вік та середній бал за минулу сесію;
- конструктор, що ініціалізує ім'я, номер групи та вік;
- відкриті функції-члени для доступу до середнього балу, що дозволяють змінювати та отримувати його значення.

1.2 Вказівники на об'єкти

Вказівники на об'єкти аналогічні вказівникам на змінні інших типів. Особливо вони необхідні, коли ми заздалегідь не знаємо, скільки об'єктів необхідно створити, тобто коли їх кількість визначається під час виконання програми. Крім того, об'єкти можуть виявитися більшими за розміром необхідної для них пам'яті. У таких випадках об'єкти створюються в динамічній пам'яті. Для цього

використовується операція **new**, за допомогою якої операційною системою виділяється ділянка пам'яті і повертається адреса початку виділеної ділянки пам'яті [1,2]¹. Операція **new** відіграє роль аналогічну функції **malloc()** у мові C.

Після того, як виділена пам'ять перестає бути необхідною для роботи програми, її рекомендується звільнити за допомогою операції **delete**, яка відіграє роль аналогічну функції **free()** у мові C.

1.2.1 Програма: клас Student

У програмі (див. лістинг 1.2) демонструються: застосування операції **new** для створення масиву об'єктів у динамічній пам'яті, доступ до компонентів об'єктів через вказівник, використання статичного члена класу. Деякі пояснення до програми наведені нижче:

- у рядках 4-23 визначено клас **Student**. Елементами цього класу є закриті змінні-члени **name**, **rating** та *статичний* член класу — змінна **numberStudents**, яка використовується для підрахунку загальної кількості створених об'єктів типу **Student**;

- дотримуючись принципу *інкапсуляції*, всі змінні-члени класу оголошені за замовчуванням закритими, а для доступу до них використовуються **set**- та **get**-функції, які створюються програмістом для контрольованого доступу до змінних-членів класу. У нашому випадку це функція **setData()** (рядки 9-14) і функції **getData()** (рядки 15-18) та **getNumberStudents()** (рядки 19-22);

- у рядку 25 значенням «нуль» ініціалізується статична змінна **numberStudents**, яка є спільною для всіх екземплярів класу (загальною для всього класу). Щоразу, коли створюється новий об'єкт і викликається функція **setData()**, то значення цієї змінної збільшується на одиницю (рядок 13);

- у головній функції **main()** визначається масив вказівників **studPtr** (рядок 28) і далі в циклі (рядки 31-37) кожен з них зв'язується за допомогою операції **new** з об'єктами типу **Student**, що створюються;

¹У розділі Рекомендована література наведено корисні для ознайомлення Інтернет-ресурси [3-6]

– доступ до змінних-членів цих об'єктів здійснюється за допомогою `set-` та `get-`функцій. Зверніть увагу, що оскільки з об'єктами пов'язані вказівники, то виклик цих функцій виконується шляхом розіменування вказівників (операція «стрілка» `->`);

– зауважте, що для ініціалізації змінних-членів класу у програмі використовується функція `setData()`, а не конструктор з параметрами. У контексті даної задачі це виявилось зручнішим [7]. У рядку 32 під час створення об'єктів спрацьовує вбудований у будь-який клас відкритий неявний конструктор без параметрів, який називається *конструктором за замовчуванням*.

Лістинг 1.2 Програма з об'єктами динамічної пам'яті

```

1  #include <iostream>
2  using namespace std;
3
4  class Student{
5      char name [40];
6      float rating;
7      static int numberStudents;
8  public:
9      void setData()
10     {
11         cout <<"Enter name: "; cin >> name;
12         cout <<"Enter rating: "; cin >> rating;
13         numberStudents++;
14     }
15     void getData()
16     {
17         cout << name << ": " << rating << endl;
18     }
19     int getNumberStudents()
20     {
21         return numberStudents;
22     }
23 };
24
25 int Student::numberStudents = 0;
26
```

```

27 int main() {
28     Student * studPtr [30];
29     int n = 0;
30     char ch;
31     do{
32         studPtr[n] = new Student;
33         studPtr[n]->setData();
34         n++;
35         cout << "Enter next (y/n)?";
36         cin >> ch;
37     } while(ch == 'y');
38
39     for(int i = 0; i < n; i++)
40     {
41         cout << i+1 <<". ";
42         studPtr[i] -> getData();
43     }
44     cout << "Number Students: " <<
         studPtr[0]->getNumberStudents();
45     for(int i = 0; i < n; ++i)
46         delete studPtr[i];
47     return 0;
48 }

```

Завдання до програми

1. Реалізуйте та вивчіть програму з лістингу 1.2. Виконайте в програмі наступні завдання та продемонструйте її працездатність із внесеними змінами:

- доопрацюйте програму так, щоб у ній підраховувалося середнє значення рейтингу групи студентів. Для цього визначте ще один статичний член класу, за допомогою якого підсумовуються значення рейтингів;

- за допомогою операції **sizeof** визначте обсяг пам'яті, що виділяється для масиву вказівників **studPtr[30]**, і обсяг пам'яті, який займає один з екземплярів класу. Поясніть отримані результати;

2. *Розробіть та продемонструйте роботу власного прикладу

програми. Для цього визначте у програмі додатковий клас, наприклад староста групи, і створіть екземпляр цього класу, користуючись прийомами, освоєними при реалізації програми з лістингу 1.1. Порівняйте рейтинг старости з середнім рейтингом групи студентів та виведіть відповідне повідомлення.

1.3 Багатофайловий проект

У даному розділі лабораторної роботи ми створимо *багатофайловий проект*. Великі програми зазвичай розбиваються на частини, що підвищує наочність програми, сприяє чіткому визначенню взаємодії окремих частин програми, дозволяє розподілити роботу зі створення програми у команді розробників. Інтегровані середовища розробки програм (Integrated Development Environment, IDE), здебільшого підтримують створення багатофайлових проектів. При цьому створюється спеціальний файл проекту, який містить інструкції компонування файлів, з яких складається багатофайловий проект.

При створенні багатофайлових об'єктно-орієнтованих проектів визначення класів переважно поміщають у заголовні файли (*.h), а реалізацію функцій-членів (методів) — у вихідні файли, наприклад (*.cpp). Це є ще одним аспектом ідеї інкапсуляції об'єктно-орієнтованого програмування. Користувачам програми в заголовних файлах надається інформація про вміст класу, тобто його інтерфейс, а саме, які змінні та функції містить клас, їх типи та сигнатури. Реалізація функцій класу може надаватися користувачам також в об'єктному коді (файли *.o) або у вигляді бібліотек (файли *.lib, *.dll). Для реалізації такого підходу функції-члени класу необхідно визначати поза тілом класу. Синтаксис такого визначення буде продемонстрований у лістингах нашого проекту.

Недоліком програми з лістингу 1.2 є і той факт, що ми оголошували фіксоване значення розміру масиву об'єктів **Student**. Тобто ми створювали однаковий розмір груп студентів при їх формуванні. Зрозуміло, що розміри навчальних груп можуть відрізнятись.

У даному проекті ми реалізуємо так званий *зв'язаний список*,

який є більш гнучкою системою зберігання даних у порівнянні з масивами. Ми формуватимемо групу студентів як зв'язані між собою об'єкти. Таким чином, групу студентів буде подано як об'єкт-список **Group**, а елементами цього списку будуть об'єкти типу **Student**. Зв'язок між об'єктами здійснюється за допомогою вказівників. Кожен із об'єктів списку студентів містить поле-вказівник **next**, значенням якого є адреса наступного елемента списку. Вказівник на список **first** є членом об'єкта-списку **Group** і містить адресу останнього, створеного за допомогою операції **new** елемента списку. Схема зв'язаного списку показана на рис. 1.1.

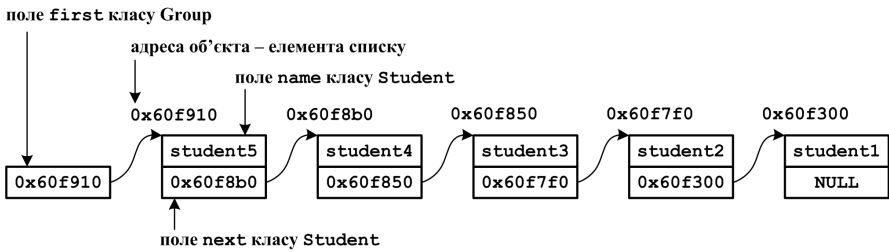


Рисунок 1.1 – Структура зв'язаного списку

1.3.1 Багатофайлова програма: класи **Student** та **Group**

Багатофайловий проект складається з п'яти файлів:

- заголовного файлу **student.h** з визначенням класу **Student** (лістинг 1.3);
- вихідного файлу **student.cpp** з реалізацією функцій-членів класу **Student** (лістинг 1.4);
- заголовного файлу **group.h** з визначенням класу **Group** (лістинг 1.5);
- вихідного файлу **group.cpp** з реалізацією функцій-членів класу **Group** (лістинг 1.6);
- файлу **main.cpp** головної функції **main()** (лістинг 1.7).

Розглянемо деякі деталі багатофайлової програми:

- у лістингу 1.3 визначено клас **Student** (рядки 4-13). Він містить змінну **name** (рядок 6) для зберігання імені студента, змінну-

вказівник **next** типу **Student** (рядок 8) для зберігання адреси наступного у списку студента, прототипи функцій доступу **setName()** і **getName()** (рядки 9 та 10) для ініціалізації та перегляду змінної **name**, конструктор без параметрів (рядок 11) та статичну змінну **snumber** (рядок 12) для підрахунку загальної кількості студентів у групі;

- лістинг 1.4 містить визначення конструктора (рядки 4-7), реалізацію функцій-членів класу **Student** та ініціалізацію статичної змінної **snumber** (рядок 19);

- у лістингу 1.5 визначено клас **Group**. Він містить вказівник **first** типу **Student** (рядок 3), змінну **number** (рядок 4) для зберігання назви групи, конструктор (рядок 6), деструктор (рядок 7), прототипи функцій-членів класу: **addStudent()** (рядок 8) для додавання студентів до списку та **display()** (рядок 9) для виведення на друк списку студентів;

- лістинг 1.6 містить реалізацію функцій-членів класу **Group**. Конструктор ініціалізує члени класу (рядки 4-9), встановлюючи назву групи **number** (рядок 7) та початкове значення вказівника **first** (рядок 8). Метод **addStudent()** (рядок 11) за допомогою операції **new** (рядок 13) створює новий об'єкт типу **Student**, викликає його функцію **setName()** (рядок 14), за допомогою якої реалізоване введення імені студента;

- у методі **addStudent()** (див. лістинг 1.6) вказівнику **next** новоствореного об'єкта присвоюється попереднє значення вказівника **first** (рядок 15), а в якості нового значення вказівника **first** присвоюється значення вказівника **newStudent** (рядок 16), тобто адреса щойно створеного об'єкта типу **Student**.

Таким чином, кожен новостворений елемент списку містить адресу попереднього елемента, а голова списку — вказівник **first** містить адресу останнього створеного елемента. Порядок організації зв'язаного списку за допомогою функції **addStudent()** пояснює зворотний порядок виведення на друк елементів списку за допомогою функції **display()** (рядок 19) та зворотний порядок вивільнення деструктором **~Group()** (рядок 32) пам'яті, що займають елементи списку.

Лістинг 1.3 Визначення класу Student (**student.h**)

```
1  #include <iostream>
2  using namespace std;
3
4  class Student{
5  private:
6      string name;
7  public:
8      Student* next;
9      void setName();
10     string getName();
11     Student();
12     static int snumber;
13 };
```

Лістинг 1.4 Реалізація методів класу Student (**student.cpp**)

```
1  #include <iostream>
2  #include "student.h"
3
4  Student::Student()
5  {
6      snumber++;
7  }
8  Student::setName()
9  {
10     cout << "Enter name: ";
11     cin >> name;
12 }
13
14 string Student::getName()
15 {
16     return name;
17 }
18
19 int Student::snumber = 0;
```

Лістинг 1.5 Визначення класу Group (group.h)

```
1 #include "student.h"
2 class Group{
3     Student* first;
4     string number;
5 public:
6     Group();
7     ~Group();
8     void addStudent();
9     void display();
10 };
```

Лістинг 1.6 Реалізація методів класу Group (group.cpp)

```
1 #include "group.h"
2 using namespace std;
3
4 Group::Group()
5 {
6     cout << "Enter name of group: ";
7     cin >> number;
8     first = NULL;
9 }
10
11 void Group::addStudent()
12 {
13     Student* newStudent = new Student;
14     newStudent -> setName();
15     newStudent -> next = first;
16     first = newStudent;
17 }
18
19 void Group::display()
20 {
21     Student * current = first;
22     cout << number << endl;
23     int i = 0;
24     while(current)
25     {
```

```

26         i++;
27         cout << i << ". " << current->getName()
           << endl;
28         current = current -> next;
29     }
30 }
31
32 Group::~Group()
33 {
34     Student * current = first;
35     while(current != NULL)
36     {
37         Student* temp = current;
38         current = current->next;
39         delete temp;
40     }
41     cout << "All elements are deleted" << endl;
42 }

```

Лістинг 1.7 Головна функція програми (main.cpp)

```

1  #include "group.h"
2
3  int main()
4  {
5      Group group;
6      char ch;
7      do{
8          group.addStudent();
9          cout << "Enter next (y/n)?";
10         cin >> ch;
11     } while(ch == 'y');
12     group.display();
13     cout << "Number students: " <<
        Student::snumber << endl;
14     return 0;
15 }

```

Завдання до проекту

1. Реалізуйте та вивчіть багатофайлову програму. Виконайте наступні завдання та продемонструйте працездатність проекту із внесеними змінами:

- створіть в IDE проект програми. Створіть усі необхідні файли багатофайлової програми та включіть їх у проект. Зауважте, що IDE має опції компіляції кожного файлу проекту окремо, а також компіляції та збирання (компонування) всього проекту. Відкомпілюйте та запустіть проект. Після введення назви групи та списку студентів програма має вивести на друк список групи студентів та їх загальну кількість;

- доопрацюйте програму так, щоб вона при додаванні елементів до списку виводила на друк адреси — значення вказівників **next** та **first**. Проаналізуйте отримані адреси елементів списку, об'єми пам'яті, які займають елементи списку, та «відстань» між ними. Дайте відповідь на запитання та обґрунтуйте відповіді:

- чи однаковий обсяг пам'яті займають об'єкти типу **Student**;
- елементи списку розміщуються в пам'яті один за одним чи іншим чином;

- додайте до класу **Student** змінну для зберігання рейтингу студента і забезпечте у програмі можливість підрахунку середнього значення рейтингу студентів групи;

2. *Розробіть та продемонструйте роботу власного прикладу багатофайлового проекту, в якому реалізуйте можливість створення кількох груп студентів (від трьох до десяти).

1.4 Зміст звіту

Звіт повинен містити: мету роботи, лістинги або фрагменти коду, що розроблені для власного прикладу програм, з необхідними поясненнями та відповіді на контрольні питання.

1.5 Контрольні питання

1. Чим відрізняються структура у мові C та клас у мові C++?

2. Як відрізняються властивості компонентів класів, визначених за допомогою ключового слова **struct** та **class**?
3. Чим визначається розмір пам'яті, який займає екземпляр класу?
4. Як отримати доступ до закритого елемента класу?
5. У чому відмінність функції, визначеної в тілі класу, від звичайної функції?
6. Як отримати доступ до елемента класу через вказівник?
7. Навіщо визначати конструктор і деструктор?
8. У чому відмінність визначення конструктора від визначення функції-члену класу?
9. Чи можна передати параметри в деструктор?
10. Чи може конструктор бути перевантаженим?
11. Скільки у класі може бути конструкторів за замовчуванням?
12. Скільки у класі може бути деструкторів?
13. Чи може бути конструктор закритим членом класу?
14. Чи можна один екземпляр класу привласнити іншому?

ЛАБОРАТОРНА РОБОТА 2

Дружні функції та класи. Перевантаження операцій

Мета роботи — вивчити дружні функції та дружні класи. Вивчити функції-оператори, за допомогою яких реалізується перевантаження операцій.

2.1 Дружні функції та класи

Для підвищення гнучкості доступу до членів класу та взаємодії між класами у мові C++ є механізм дружніх функцій та дружніх класів. Функція, яка оголошена дружньою для даного класу, надається доступ до закритих та захищених полів цього класу. Дружня функція може бути незалежною функцією чи функцією-членом іншого класу. Головні особливості дружньої функції:

- дружня функція не належить до класу, який надає їй свою дружбу. Тому на неї не поширюється дія специфікаторів доступу **public**, **protected**, **private**, а отже, вона може бути оголошена дружньою у будь-якій секції визначення тіла класу;
- у дружню функцію при її виклику не передається прихований вказівник **this** екземпляра класу, в якому ця функція оголошена дружньою. Тому для роботи з членами цього класу в дружню функцію повинні явно передаватися в якості параметру або екземпляр класу, або вказівник на нього, або його посилання.

2.1.1 Програма: класи **Student**, **Professor**, **Dean**

У програмі (див. лістинг 2.1) показаний приклад із двома дружніми функціями. Сценарій програми наступний: професор оцінює знання студента на ім'я **name** та встановлює його рейтинг **rating** і передає відомості щодо оцінки у деканат. Декан, ознайомившись із рейтингом студента, говорить йому «Добре!» або «Погано!». Проаналізуємо реалізацію програми:

- компілятору при обробці коду класів **Professor** та **Dean** необхідно «знати» про клас **Student**, тому в рядку 4 виконано *none*

редне оголошення класу `Student`;

- функція `setRating()` класу `Professor` (рядок 7) і функція `getRating()` класу `Dean` (11) оголошені дружніми (рядки 18 і 19) для класу `Student`;

- як ми вже говорили вище, в дружню функцію не передається прихований вказівник `this` на екземпляр класу, що надає дружбу іншим класам. Тому ми передаємо явно вказівник на екземпляр класу `Student` (рядки 34 і 39) і всередині дружніх функцій отримуємо доступ до членів класу `Student` шляхом розіменування вказівника (наприклад, рядок 36).

Лістинг 2.1 Приклад програми із дружніми функціями

```

1  #include <iostream>
2  using namespace std;
3
4  class Student;
5
6  class Professor{
7      public: void setRating(Student *, float);
8  };
9  class Dean{
10     public:
11         float getRating(Student *);
12         void message();
13     };
14
15     class Student{
16         string name;
17         float rating;
18         friend void Professor::setRating(Student *,
19             float);
19         friend float Dean::getRating(Student *);
20     public:
21         string getName();
22         Student();
23     };
24     Student::Student()
25     {

```

```

26     cout<<"Enter name: "; cin>>name;
27 }
28
29 string Student::getName()
30 {
31     return name;
32 }
33
34 void Professor::setRating(Student *st, float
    rt)
35 {
36     st->rating=rt;
37 }
38
39 float Dean::getRating(Student* st)
40 {
41     cout << st->getName();
42     if(st->rating >= 75) cout << ": Good!\n";
43     else cout << ": Bad!\n";
44     return st->rating;
45 }
46
47 int main() {
48
49     Student st1, st2, st3;
50     Professor prof;
51     Dean dn;
52
53     prof.setRating(&st1, 65);
54     prof.setRating(&st2, 75);
55     prof.setRating(&st3, 76);
56
57     dn.getRating(&st1);
58     dn.getRating(&st2);
59     dn.getRating(&st3);
60
61     return 0;
62 }

```

Завдання до програми

1. Реалізуйте та вивчіть програму з лістингу 2.1. Виконайте в програмі наступні завдання та продемонструйте її працездатність із внесеними змінами:

- забезпечте можливість введення рейтингу студента з клавіатури, для цього внесіть зміни в реалізацію функції `setRating()` класу `Professor` і у відповідні рядки коду. Результат виконання завдання може бути наступним (див. рис. 2.1);

2. *Розробіть та продемонструйте роботу власного прикладу програми, у якому реалізуйте наступний сценарій:

- за результатами сесії кожен студент отримує оцінки з 8 навчальних дисциплін. Для цього у конструктор класу `Student` можна внести можливість ініціалізації або введення з клавіатури оцінок студента;

- декан підраховує середній рейтинг студента і виносить рішення про можливість призначення стипендії. Для цього у класі `Dean` потрібно розробити функцію, яка є дружньою для класу `Student`, яка обчислює середній рейтинг студента за навчальний семестр та дозволяє декану прийняти рішення щодо стипендії студента;

- професор може переглянути середній рейтинг студента за семестр. Для цього у класі `Professor` потрібно розробити функцію, яка є дружньою для класу `Dean` і може виводити на екран дані студента, значення його середнього рейтингу і висновок щодо стипендії;

- при розробці коду зауважте, що може виникнути необхідність визначення деяких змінних-членів класів. Результат виконання завдання може бути наступним (див. рис. 2.2)

2.1.2 Програма: класи `Array`, `Sort`

У програмі (див. лістинг 2.2) показана реалізація *дружнього класу*. Деякі пояснення:

- клас `Sort` оголошений дружнім для класу `Array` (рядок 10), тому всі його функції-члени (в даному випадку це одна функція

`min()`) мають доступ до полів класу `Array`;

- у конструкторі класу `Array` за допомогою операції `new` створюється масив динамічної пам'яті, на який посилається вказівник `p*` (рядок 16) та виконується ініціалізація масиву випадковими числами в діапазоні від 0 до 99 (рядки 18-22);

- оскільки ми створюємо масив динамічної пам'яті, то для її вивільнення за допомогою операції `delete` необхідний явний деструктор (рядки 12 і 26-29);

- функція `min()` дружнього класу `Sort` (рядки 33, 36-43) має доступ до всіх полів класу `Array` і завдяки цьому визначає мінімальний елемент масиву, що належить екземпляру класу `Array` ;

- в функцію `min()` як аргумент передається вказівник на екземпляр класу, який надає свою дружбу (рядок 36).

```
Enter name: Василенко
Enter name: Петренко
Enter name: Степаненко
Enter rating for Василенко: 83
Enter rating for Петренко: 74
Enter rating for Степаненко: 96
Василенко: Good!
Петренко: Bad!
Степаненко: Good!
```

Рисунок 2.1 – Приклад виконання коду завдання 1 до лістингу 2.1

Лістинг 2.2 Приклад програми із дружнім класом

```
1  #include <iostream>
2  #include <ctime>
3  #include <cstdlib>
4  #include <iomanip>
5  using namespace std;
6  class Array {
7      int n;
8      int *p;
9  public:
10     friend class Sort;
11     Array(int);
12     ~Array();
13 };
```

```

14 Array::Array(int m):n(m)
15 {
16     p = new int [n];
17     srand(time(NULL));
18     for(int i=0; i < n; i++)
19     {
20         p[i] = rand()%100;
21         cout << setw(2) << p[i] << " ";
22     }
23     cout << endl;
24 }
25
26 Array::~~Array()
27 {
28     delete [] p;
29 }
30
31 class Sort{
32 public:
33     int min (Array *);
34 };
35
36 int Sort::min(Array *x)
37 {
38     int tmp = x-> p [0];
39     for(int i = 1; i < x->n; i++)
40         if(x->p[i] <tmp)
41             tmp = x-> p [i];
42     return tmp;
43 }
44
45 int main()
46 {
47     Array arr(10);
48     Sort srt;
49     cout << "min = " << srt.min(&arr);
50     return 0;
51 }

```

```

Enter name: Васильченко
Enter rating for course 1: 75
Enter rating for course 2: 76
Enter rating for course 3: 85
Enter rating for course 4: 70
Enter rating for course 5: 90
Enter rating for course 6: 80
Enter rating for course 7: 91
Enter rating for course 8: 73
Enter name: Петренко
Enter rating for course 1: 65
Enter rating for course 2: 63
Enter rating for course 3: 68
Enter rating for course 4: 75
Enter rating for course 5: 63
Enter rating for course 6: 65
Enter rating for course 7: 77
Enter rating for course 8: 70
Enter name: Степаненко
Enter rating for course 1: 88
Enter rating for course 2: 90
Enter rating for course 3: 95
Enter rating for course 4: 98
Enter rating for course 5: 87
Enter rating for course 6: 93
Enter rating for course 7: 90
Enter rating for course 8: 92

Student scholarship for Васильченко: Yes! Average rating = 80

Student scholarship for Петренко: No! Average rating = 68.25

Student scholarship for Степаненко: Yes! Average rating = 91.625

```

Рисунок 2.2 – Приклад виконання коду завдання 2 до лістингу 2.1

Завдання до програми

1. Реалізуйте та вивчіть програму з лістингу 2.2. Виконайте в програмі наступні завдання та продемонструйте її працездатність із внесеними змінами:

- доопрацюйте конструктор класу **Array** так, щоб у ньому здійснювалася перевірка виділення динамічної пам'яті для масиву та виведіть відповідне повідомлення;

2. Розробіть та продемонструйте роботу власного прикладу програми, у якому зробіть наступне:

- розробіть функцію-член класу **Sort**, яка забезпечить можливість сортування елементів масиву класу **Array** за зростанням (спаданням).

2.2 Перевантаження операцій

Можливість перевизначати операції у мові C++ дозволяє надати власний сенс звичайним операціям мови. Створюючи власні типи (класи) та екземпляри цих типів (об'єкти), за допомогою перевантаження операцій можна визначити необхідні операції над цими об'єктами. Таким чином, мову C++ можна розширювати новими типами і операціями. Крім того, код програми, в якій використовуються перевантажені операції, більш простий та інтуїтивно зрозумілий [1, 2].

2.2.1 Програма: клас Folder

У програмі (див. лістинг 2.3) демонструються найпростіші приклади перевантаження операцій: бінарної та унарної. Сценарій, який моделює програма, полягає в тому, що ми визначаємо кількість файлів у каталозі та його обсяг. Копіюючи у цей каталог інші каталоги, ми збільшуємо загальну кількість файлів та обсяг каталогу. Таким чином ми вводимо об'єкти типу каталог і за допомогою перевантажених операцій обробляємо ці об'єкти. В результаті бінарна операція додавання стає операцією копіювання в каталог іншого каталогу, а операція інкремент — операцією створення копії будь-якого каталогу в даному каталозі.

Короткий аналіз програми:

- у рядках 14-20 визначено функцію-оператор, яка перевантажує бінарну операцію підсумовування. Лівим операндом бінарної перевантаженої операції завжди є екземпляр класу, у якому викликається перевантажена операція, а правим — екземпляр класу, який передається у функцію-оператор як аргумент. Усередині функції створюється тимчасовий об'єкт `tmp` (рядок 16), а його змінній-члену `numberFiles` присвоюється сума змінної-члена `numberFiles` лівого операнда (екземпляра класу, в якому викликається дана функція-оператор) та одноіменної змінної-члена правого операнда (екземпляра `f`). Теж саме відбувається і зі змінними-членами `folderSize`;
- функція-оператор `operator+()` має тип `Folder`, тобто вона повертає об'єкт класу `Folder` (рядок 19). У даному випадку це

тимчасовий об'єкт `tmp`;

- у рядках 21-26 визначено функцію-оператор, що перевантажує постфіксну унарну операцію інкремент, яка імітує створення копії каталогу. Зрозуміло, що при цьому кількість файлів та обсяг каталогу повинні збільшитися вдвічі. Оскільки у постфіксній унарній операції відсутній правий операнд, то у функцію-оператор не передається жодного аргументу, тому в якості аргументу потрібно записати будь-який тип або неіснуючу змінну будь-якого типу (ми записали `int noname`). Для префіксної унарної операції аргумент має бути відсутнім;

- у даній перевантаженій унарній операції інкремента нам необхідно, щоб функція-оператор повертала об'єкт, який викликає цю функцію-оператор. Тому в якості значення, що повертається використовується розіменований вказівник `this` (рядок 25), тобто об'єкт, який викликав цю функцію-оператор;

- у головній функції у рядку 35 об'єкту `f3` присвоюється сума об'єктів `f1` і `f2`. Отже відбувається порозрядне копіювання об'єкта `tmp`, який повертається функцією-оператором `operator+`(), в об'єкт `f3`. Створену перевантажену бінарну операцію можна застосовувати, наприклад, як показано в рядку 39;

- використання унарної перевантаженої операції показано у рядку 37, причому, оскільки пріоритет операції інкременту вище пріоритету операції підсумовування, то спочатку виконається унарна операція, а потім – бінарна.

Лістинг 2.3 Програма з перевантаженими операціями класу `Folder`

```

1  #include <iostream>
2
3  using namespace std;
4
5  class Folder{
6      int numberFiles;
7      int folderSize;
8  public:
9      Folder(int n=0, int s=0)
10     {
```

```

11         numberFiles = n;
12         folderSize = s;
13     }
14     Folder operator+(Folder f)
15     {
16         Folder tmp;
17         tmp.numberFiles = numberFiles +
18             f.numberFiles;
19         tmp.folderSize = folderSize + f.folderSize;
20         return tmp;
21     }
22     Folder operator++(int noname)
23     {
24         numberFiles *= 2;
25         folderSize *= 2;
26         return *this;
27     }
28     void getProperties()
29     {
30         cout << numberFiles << " " << folderSize <<
31             endl;
32     }
33 };
34
35 int main() {
36     Folder f1(4,100), f2(10,300), f3;
37     f3 = f1 + f2;
38     cout << "f3 = f1+f2: "; f3.getProperties();
39     f3 = f3 + f1 ++;
40     cout << "f3 = f3 + f1++: "; f3.getProperties();
41     f1 = f1 + f2 + f3;
42     cout << "f1 = f1 + f2 + f3 :";
43     f1.getProperties();
44     return 0;
45 }

```

Завдання до програми

1. Реалізуйте та вивчіть програму з лістингу 2.3. Виконайте в програмі наступні завдання та продемонструйте її працездатність із внесеними змінами:

- запрограмуйте одну з перевантажених операцій поза тілом класу (синтаксис аналогічний визначенню функцій-членів класу поза тілом класу);

- поекспериментуйте із застосуванням даних перевантажених операцій у головній функції. Реалізуйте декілька додаткових виразів з перевантаженими операціями та поясніть результат;

2. *Розробіть та продемонструйте роботу власного прикладу програми, у якому створіть власні перевантажені операції:

- бінарну операцію віднімання, що імітує переміщення деякого каталогу з даного каталогу;

- унарну операцію декремента, що імітує знищення файлів у деякому каталозі. У цьому випадку число файлів та обсяг каталогу мають обнулитися, а сам каталог має залишитися.

2.2.2 Програма: клас `Matrix`

Дана програма¹ (див. лістинг 2.4) складніша за попередню, але в ній реалізоване корисне і часто використовуване на практиці перевантаження операцій. У програмі представлені перевантажені операції: бінарна операція для складання матриць та унарна — для транспонування матриці. Враховуючи, що матриці створюються як масиви в динамічній пам'яті, які підлягають знищенню після їх використання, то виникають деякі особливості застосування техніки перевантаження операцій. Зокрема, в програмі показано застосування так званого *конструктора копіювання* та необхідність *перевантаження операції присвоєння*. Програма має багато операторів налагоджувального друку для демонстрації деяких особливостей реалізації. Ці оператори можна після налагодження програми усунути або закоментувати. Нижче наведено аналіз програми та лістинг 2.5

¹Ця програма не входить до числа обов'язкових для заліку лабораторної роботи

з результатами роботи програми.

Для аналізу нам знадобиться знання конструктора копіювання. Під час передачі об'єкта у функцію, як відомо, всередині функції створюється локальна копія цього об'єкта, а по завершенні її роботи ця локальна копія знищується. Якщо об'єкт не використовує динамічну пам'ять, то жодних проблем, крім витрат пам'яті, нема. Якщо ж об'єкт використовує динамічну пам'ять, то при виході з функції ця ділянка пам'яті звільнюється, і об'єкт, який передавався у функцію, стає пошкодженим. Для уникнення такої ситуації у мові C++ використовується конструктор копіювання, у якому програміст сам регулює процес копіювання.

Синтаксис заголовка конструктора копіювання показаний у рядку 16. У конструктор передається *посилання на об'єкт* типу **Matrix**. Посилання слід розуміти як вказівник, який не потрібно розіменувати.

У програмі введено спеціальний лічильник — статичний член класу **counter** (рядок 13), за допомогою якого можна відстежити кількість і номери об'єктів (матриць), що створюються і знищуються.

Отже, розглянемо деякі особливості реалізації перевантаження операцій у даній програмі:

- для створення матриць використовується звичайний конструктор (рядки 15, 29-37), у який передаються розміри матриці. Матриця представляється як «двовимірний» масив динамічної пам'яті, який створюється за допомогою операції **new**. Для перевірки виділення пам'яті розроблено функцію **errMem()** (рядки 12, 82-89);

- конструктор копіювання (рядки 16, 38-51) отримує посилання на об'єкт (матрицю), копіює його масив у власний масив. Таким чином, динамічна пам'ять та вказівник на неї у вихідній матриці не піддаються змінам у функціях, в які вона передається за посиланням;

- деструктор **~Matrix()** (рядки 17, 53-59) вивільняє пам'ять, виділену для матриці в конструкторі та конструкторі копіювання;

- функція **fillMatrix()** присвоює елементам матриць випадкові дійсні числа в діапазоні від 0 до 1 (рядки 23, 61-66);

- для зручності у класі передбачена функція виводу матриць на екран `printMatrix()` (рядки 24, 68-80);

- перевантажена операція додавання (рядки 20, 108-116) сумує поелементно масиви двох об'єктів, тобто матриць. Усередині цієї функції-оператора створюється тимчасова матриця `tmp` (рядок 111), у елементи якої заносяться значення відповідної суми елементів матриць, що складаються. При виході з функції-оператора тимчасова матриця `tmp` буде знищена деструктором;

- перевантажена операція транспонування матриці (рядки 21, 118-126) розроблена як перевантаження унарної операції логічного заперечення (в такому випадку підійде будь-яка допустима унарна операція) і як у попередній програмі функція-оператор повертає розіменований вказівник `this`, тобто об'єкт, який викликав цю операцію. Під час виходу з функції-оператора тимчасова матриця `tmp` буде знищена деструктором;

- після обчислення суми двох об'єктів (матриць) результат може бути наданий іншому об'єкту (матриці) (рядок 146). Якщо цього застосувати звичайне присвоєння об'єкта, то вказівник на масив динамічної пам'яті об'єкта, що стоїть ліворуч від операції присвоєння, буде вказувати на неіснуючий масив, оскільки тимчасовий об'єкт `tmp` буде знищений деструктором під час виходу з функції. Тому в класі розроблена спеціальна *перевантажена операція присвоєння*, яка створює власний масив та покажчик на нього. У цей масив копіюються значення масиву матриці `tmp` перед її знищенням;

- перевантажена операція присвоєння (рядки 19, 91-106) повертає об'єкт, що викликав цю операцію, тобто лівий операнд операції. Це здійснюється за допомогою розіменування покажчика `this`. Рядок 94 необхідний для випадку, якщо програміст вирішить присвоїти об'єкт самому собі. Так само перевантажена операція присвоєння спрацьовує при виконанні перевантаженої операції для транспонування матриці (рядок 150).

Лістинг 2.4 Програма з перевантаженими операціями класу Matrix

```

1  #include <iostream>
2  #include <iomanip>
3  #include <ctime>
4  #include <cstdlib>
5
6  using namespace std;
7
8  class Matrix{
9      int n;
10     int m;
11     double **a;
12     void errMem(void*);
13     static int count;
14 public:
15     Matrix (int, int);
16     Matrix (const Matrix & matrix);
17     ~Matrix();
18
19     Matrix operator = (const Matrix & matrix);
20     Matrix operator + (const Matrix & matrix);
21     Matrix operator! ();
22
23     void fillMatrix();
24     void printMatrix();
25 };
26
27 int Matrix::count = 0;
28
29 Matrix::Matrix(int nn, int mm):n(nn), m(mm)
30 {
31     cout << "Construtor, create object " <<
32         ++count << endl;
33     a = new double * [n]; errMem(a);
34     for(int i = 0; i < n; i++)
35     {
36         a[i] = new double[m]; errMem(a[i]);
37     }
38 }

```

```

38 Matrix::Matrix(const Matrix& matrix)
39 {
40     cout << "Copy-constructor, create object " <<
        ++count << endl;
41     n = matrix.n; m = matrix.m;
42     a = new double * [n]; errMem(a);
43     for(int i =0; i<n; i++)
44     {
45         a[i] = new double[m]; errMem(a[i]);
46     }
47
48     for(int i = 0; i<n; i++ )
49         for(int j = 0; j < m; j++)
50             a[i][j] = matrix.a[i][j];
51 }
52
53 Matrix::~Matrix()
54 {
55     cout << "Destructor, delete object " <<
        count-- << endl;
56     for(int i=0; i<n; i++)
57         delete a[i];
58     delete [] a;
59 }
60
61 void Matrix::fillMatrix()
62 {
63     for(int i=0; i<n; i++)
64         for(int j=0; j<m; j++)
65             a[i][j] = (double)rand()/RAND_MAX;
66 }
67
68 void Matrix::printMatrix()
69 {
70     cout << endl;
71     cout.setf(ios::fixed);
72     cout.precision(3);
73     for(int i=0; i<n; i++)
74     {
75         for(int j = 0; j<n; j++)

```

```

76         cout <<a[i][j]<<" ";
77     cout << endl;
78 }
79 cout << endl;
80 }
81
82 void Matrix::errMem(void *p)
83 {
84     if(!p)
85     {
86         cout<<"Memory usage error!"<<endl;
87         exit(1);
88     }
89 }
90
91 Matrix Matrix::operator = (const Matrix& matrix)
92 {
93     cout << "operator = \n";
94     if(this == &matrix) return *this;
95     n = matrix.n; m = matrix.m;
96     a = new double * [n]; errMem(a);
97     for(int i =0; i<n; i++)
98     {
99         a[i] = new double[m]; errMem(a[i]);
100     }
101     for(int i = 0; i<n; i++)
102         for(int j = 0; j < m; j++)
103             a[i][j] = matrix.a[i][j];
104
105     return *this;
106 }
107
108 Matrix Matrix::operator+(const Matrix& matrix)
109 {
110     cout << "operator + \n";
111     Matrix tmp(n,m);
112     for(int i = 0; i<n; i++)
113         for(int j = 0; j<m; j++)
114             tmp.a[i][j] = a[i][j] + matrix.a[i][j];
115     return tmp;

```

```

116 }
117
118 Matrix Matrix::operator !()
119 {
120     cout << "operator ! \n";
121     Matrix tmp(n,m);
122     for(int i = 0; i<n; i++)
123         for(int j = 0; j<m; j++)
124             tmp.a[i][j] = a[j][i];
125     return tmp;
126 }
127
128 int main() {
129     srand(time(NULL));
130
131     Matrix m1(3,3);
132     m1.fillMatrix();
133     cout << "\nm1:\n";
134     m1.printMatrix();
135
136     Matrix m2(3,3);
137     m2.fillMatrix();
138     cout << "\nm2:\n";
139     m2.printMatrix();
140
141     Matrix m3(3,3);
142     cout << "\nm3:\n";
143     m3.fillMatrix();
144     m3.printMatrix();
145
146     m3 = m1+m2;
147     cout << "\nm3 = m1 + m2: \n";
148     m3.printMatrix();
149
150     m2 = !m1;
151     cout << "\nm2 = !m1: \n";
152     m2.printMatrix();
153
154     return 0;
155 }

```

Тепер зробимо кілька коментарів щодо роботи головної функції (рядок 128). Для цього розглянемо, що виведеться на друк в результаті її роботи (див. лістинг 2.5):

- створено три матриці `m1`, `m2` та `m3` (рядки 1, 8 і 16). Для цього тричі спрацював конструктор. Матриці заповнені випадковими числами;

- рядок коду `m3 = m1+m2` у функції `main()` спрацював наступним чином:

- викликано перевантажену операцію додавання (рядок 24) і створено об'єкт 4 (рядок 25), тобто матриця `tmp`;

- при виході з перевантаженої операції додавання спрацювала перевантажена операція присвоєння (26) і спрацював конструктор копіювання, який створив об'єкт 5 (рядок 27);

- потім непотрібні надалі об'єкти 5 і 4 було знищено (рядки 28 і 29);

- рядок коду `m2 = !m1` у функції `main()` викликав перевантажену операцію для транспонування матриці. При цьому виконалась аналогічна послідовність створення допоміжних об'єктів, виклику перевантаженої операції присвоєння, знищення допоміжних об'єктів;

- після роботи перевантажених операцій на друк виведено результуючі матриці `m3` (рядки 31-35) та `m2` (рядки 44-48);

- наприкінці головної функції тричі спрацював деструктор, що знищив матриці `m3`, `m2` та `m1`.

Лістинг 2.5 Результат роботи програми з лістингу 2.4

```

1  Construtor, create object 1
2  m1:
3
4  0.138 0.671 0.254
5  0.671 0.287 0.246
6  0.214 0.480 0.562
7
8  Construtor, create object 2
9
10 m2:
```

```
11
12 0.018 0.017 0.987
13 0.464 0.518 0.464
14 0.110 0.781 0.268
15
16 Construtor, create object 3
17
18 m3:
19
20 0.830 0.012 0.569
21 0.968 0.917 0.055
22 0.915 0.317 0.236
23
24 operator +
25 Construtor, create object 4
26 operator =
27 Copy-construtor, create object 5
28 Destructor, delete object 5
29 Destructor, delete object 4
30
31 m3 = m1 + m2:
32
33 0.156 0.688 1.241
34 1.135 0.805 0.710
35 0.324 1.261 0.829
36
37 operator!
38 Construtor, create object 4
39 operator =
40 Copy-construtor, create object 5
41 Destructor, delete object 5
42 Destructor, delete object 4
43
44 m2 = !m1:
45
46 0.138 0.671 0.214
47 0.671 0.287 0.480
48 0.254 0.246 0.562
49
50 Destructor, delete object 3
```

```
51 Destructor, delete object 2
52 Destructor, delete object 1
```

Завдання до програми

1. Реалізуйте та вивчіть програму з лістингу 2.4. Виконайте в програмі наступні завдання та продемонструйте її працездатність із внесеними змінами:

- для класу **Matrix** розробіть функції-члени (або конструктори), що забезпечують можливість:
 - створення нульової матриці (всі елементи якої дорівнюють нулю);
 - створення одиничної матриці (яка має елементи головної діагоналі такі, що дорівнюють одиниці, а всі інші — нулю);
 - введення значень елементів матриці з клавіатури;
 - поекспериментуйте із застосуванням даних перевантажених операцій у головній функції, створюючи в ній різну кількість матриць. Реалізуйте декілька додаткових виразів з перевантаженими операціями та поясніть результат;

2. *Розробіть та продемонструйте роботу власного прикладу програми, у якому зробіть наступне:

- створіть з цієї програми багатофайловий проект, помістивши визначення класу в заголовний файл, а реалізацію функцій-членів — у вихідні файли;
- у проекті додатково розробіть перевантажені операції:
 - множення матриць;
 - заміни значення всіх елементів матриці на протилежні.

2.3 Зміст звіту

Звіт повинен містити: мету роботи, лістинги або фрагменти коду, що розроблені для власного прикладу програм, з необхідними поясненнями та відповіді на контрольні питання.

2.4 Контрольні питання

1. Пояснити призначення дружніх функцій та класів;
2. Пояснити дію специфікаторів доступу `public`, `protected` та `private` на дружню функцію?
3. Чому в дружню функцію необхідно передавати об'єкт чи посилання або вказівник на об'єкт?
4. Пояснити необхідність попереднього оголошення класів при створенні визначень класів з дружніми функціями;
5. Що таке перевантаження операцій у мові C++?
6. Який з операндів бінарної перевантаженої операції є об'єктом, що викликає, для функції-оператора, що реалізує цю перевантажену операцію?
7. Чому не можна за допомогою перевантаження операції змінити зміст операції над базовими типами?
8. Який з операндів бінарної перевантаженої операції передається у функцію-оператор в якості аргумента?
9. Чи може в якості одного з операндів бінарної перевантаженої операції бути змінна базового типу? Якщо так, то наведіть приклад.
10. Чи може в якості операнда унарної перевантаженої операції бути змінна базового типу? Якщо так, то якого типу?
11. Чим відрізняються префіксна та постфіксна форми унарної перевантаженої операції?
12. Чи може перевантажена операція бути перевантаженою? Обґрунтуйте відповідь.
13. Рядок 14 (ліст. 2.3) можна записати в наступних формах (поясніть кожен):

```
Folder operator+(Folder&f)
Folder operator+(const Folder&f)
const Folder operator+(const Folder&f)
const Folder operator+(const Folder&f) const
```

14. Яким чином можна повернути об'єкт із функції? Чи є можливість повернути із функції посилання на об'єкт?
15. Що таке посилання? Навести приклади використання.

16. Що означають наступні рядки коду і коли їх необхідно застосовувати?

```
void func(AnyClass& obj){...}  
void func(const AnyClass&obj){...}
```

17. Для чого необхідні конструктор копіювання та конструктор копіювання за замовчуванням? У чому їхня відмінність?

18. Навіщо буває потрібно створювати перевантажену операцію присвоювання?

19. Чи може бути перевантажена функція дружньою та не бути функцією-членом класу (або бути функцією-членом іншого класу)? Якщо так, то наведіть приклади.

ЛАБОРАТОРНА РОБОТА 3

Спадкування в C++

Мета роботи — вивчити механізм спадкування класів, віртуальні функції та абстрактні класи.

3.1 Прості приклади спадкування

Однією з цілей ООП є повторне використання вже існуючих класів. Це дозволяє заощаджувати час, сили, гроші. Іноді виникають ситуації, коли клас вже створений, налагоджений і навіть багато разів використовувався в різних додатках (програмах), але для нового випадку його використання необхідно додати ще якісь властивості класу до вже існуючих. У таких випадках ООП дозволяє побудувати новий об'єкт, використовуючи характеристики вже існуючого об'єкта. Процес створення нового класу, який використовує деякі властивості існуючих класів, називається *спадкуванням*. Клас, чий властивості успадковуються, називають *базовим* або *батьківським*. Клас, який успадковує властивості базового класу, називають *похідним* або *дочірнім*.

На рис. 3.1 показані UML-діаграми класів та різні види спадкування. Нагадаємо деякі позначення мови UML та діаграми класів:

- клас позначається графічно у вигляді прямокутника, який може бути поділений на три секції:
- назва (ім'я) класу;
- атрибути (змінні-члени, поля) класу;
- операції, які може виконувати клас (функції-члени класу, методи);
- перед атрибутами та операціями класу можуть бути вказані специфікатори доступу: мінус (-) — **private**, плюс (+) — **public**, решітка (#) — **protected** ;
- обов'язковою є тільки секція з назвою класу, решта секцій заповнюється, якщо необхідно акцентувати увагу на деяких елемен-

тах класу;

- за потреби для полів класу можуть бути зазначені їх типи (`int`, `char`, `string` тощо);
- відношення узагальнення (спадкування) між класами позначається графічно у вигляді безперервної лінії та незабарвленого трикутної стрілки, спрямованої від похідного до базового класу;
- для деяких елементів діаграми можуть бути створені примітки як зображення аркуша із загнутим кутком.

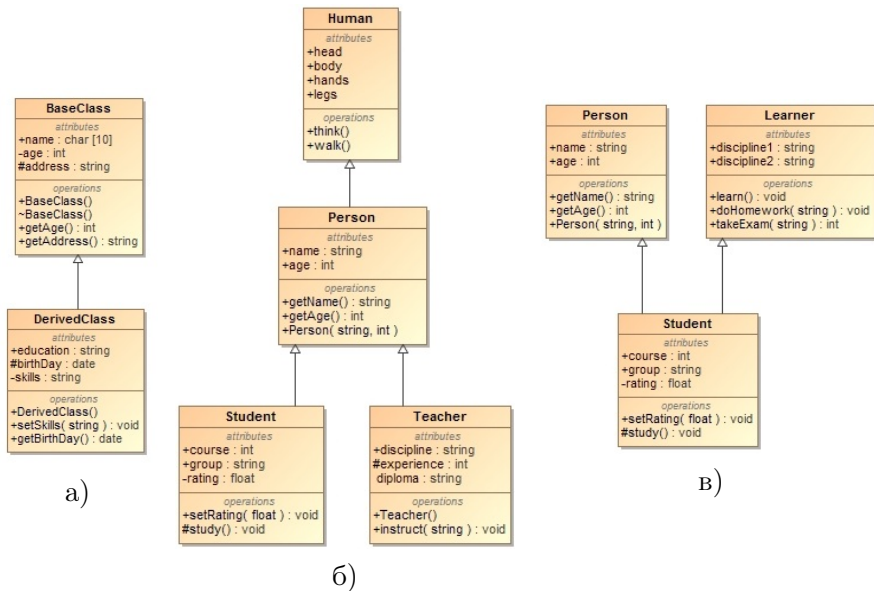


Рисунок 3.1 – Діаграми класів: а) просте спадкування; б) опосередковане множинне спадкування; в) пряме множинне спадкування

Як модельний приклад для вивчення спадкування візьмемо наступний сценарій. Нехай ви створюєте вебдодаток, який у online-режимі формує резюме для клієнта вашого сервісу. Розглянемо тільки бізнес-логіку вашого додатка, тобто процес формування вмісту резюме. Правила складання резюме для різних країн мають свої особливості та традиції, але вміст всіх резюме має багато спільного. З точки зору програмування, всі резюме можна визначити як клас:

клас резюме. Для того, щоб не програмувати знову вже існуючі та загальні для всіх резюме поля, а тільки розширювати клас залежно від країни та призначення резюме (працевлаштування, навчання) будемо використовувати механізм спадкування класів.

Отже, розглянемо діаграму класів, наведену на рис. 3.2, та відзначимо деякі особливості резюме для різних країн. Звичайно, нашою метою не є всебічний та докладний виклад усіх тонкощів складання та оформлення резюме, тому ми виділимо лише деякі з них.

Перш за все, резюме є інструментом маркетингу, в якому чітко та коротко викладено інформацію щодо вашої освіти та практичного досвіду. Зазвичай, спільними полями для всіх резюме є контактна інформація, така як: ім'я, адреса, номер телефону, електронна адреса. Але це ще не є резюме, бо такі атрибути можуть бути складовими деяких інших сутностей, наприклад, візитівки. Тому введемо поки що простий клас **Abstract**, що містить змінні-члени **name**, **phone**, **address** та **e-mail** та функції-члени для керування даними. Якщо цей клас розширити інформацією про освіту, досвід роботи та деякими іншими даними, то ми отримаємо резюме.

Є суттєва різниця між резюме в Європі та резюме в США. Головна з них — резюме в США дуже коротке та не містить подробиць особистого характеру (вік, сімейний стан, хоббі), але бажаною є наявність даних про досягнуті успіхи (у навчанні, роботі) та спеціальні вміння (наприклад, знання унікальної мови). Крім того, резюме оформлюється на одному аркуші спеціального розміру (див. примітку на рисунку).

Дещо розширене резюме в Європі називається *curriculum vitae* (CV) (лат. «шлях життя»). А в США *curriculum vitae* складається лише для осіб, діяльність яких пов'язана з наукою, освітою та медициною. Разом із загальноєвропейським стандартом, CV у різних країнах мають свої відмінності. Наприклад, у резюме для Франції необхідна кольорова фотографія, схвалюються відомості про особисте життя (сімейний стан, кількість дітей). Для Франції потрібно вказувати вік, а для Німеччини день народження та бажано написати про хоббі. У резюме для Іспанії необхідні паспортні дані (но-

мер). Для Італії — фотографію та інформацію про хоббі можна не наводити, але обов’язково дати згоду (autorizzo) про використання ваших персональних даних (інакше резюме взагалі не буде розглянуто). У Великій Британії бажано вказати двох осіб, які можуть рекомендувати вас, а надання фотографії не бажано.

З урахуванням вищевикладених особливостей клас **CurriculumVitae** успадковує властивості класу **ResumeEU** та розширює його шляхом додавання нових властивостей (наприклад, поле **skills**) (див. рис. 3.2). Разом з тим клас **CurriculumVitae** є батьківським для класів **FranceCV**, **GermanyCV** тощо, які розширюють його полями з урахуванням особливостей, прийнятих для тієї чи іншої країни. Клас **USACV** (*curriculum vitae* для США) виділено окремо, він успадковує безпосередньо (прямо) два класи, розширюючи резюме для США та з огляду на поля європейського **CV**.

Таким чином, замість того, щоб створювати класи **CV** для різних країн, щоразу повторюючи створення полів та функцій роботи з ними, ми створюємо ці класи, поступово збільшуючи (уточнюючи, специфікуючи) їх зміст та використовуючи, в той же час, раніше створені поля та функції батьківських класів.

3.1.1 Програма: класи **Abstract**, **ResumeEU**, **ResumeUSA**

У нашій першій програмі даної лабораторної роботи ми вивчимо основний синтаксис спадкування, реалізувавши просте спадкування двох класів, виділених з нашої моделі. Діаграму класів нашої програми наведено на рис. 3.3, код програми — в лістингу 3.1. У програмі спеціально введено деякі оператори виведення на друк, для дослідження послідовності спрацьовувань конструкторів та деструкторів. Ці оператори згодом можна прибрати або закоментувати.

Розглянемо особливості коду програми:

- у класі **Abstract** (рядок 4) визначено змінні-члени класу: відкритий — **name**, закритий — **phone** та захищений — **e_mail**. Для доступу до закритого та захищеного змінних-членів визначено функції **getEmail()** і **getPhone()** (рядки 19 і 20);

- конструктор **Abstract** призначений для введення значень змінних-членів класу;

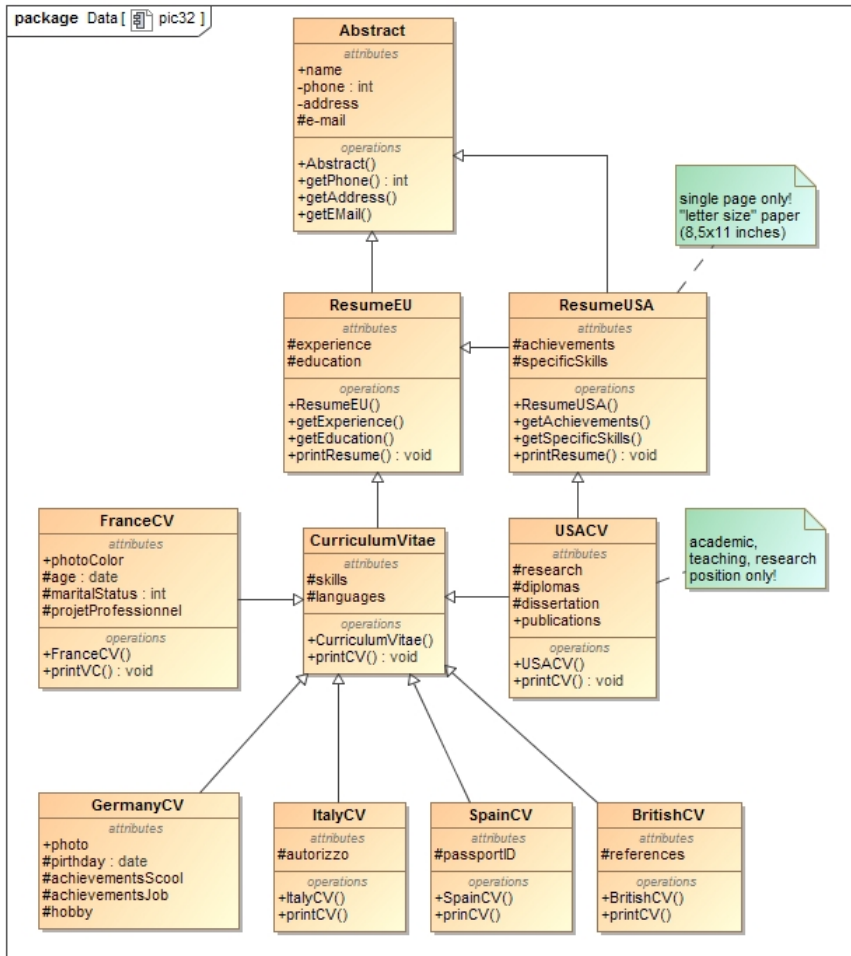


Рисунок 3.2 – Діаграма класів Curriculum Vitae

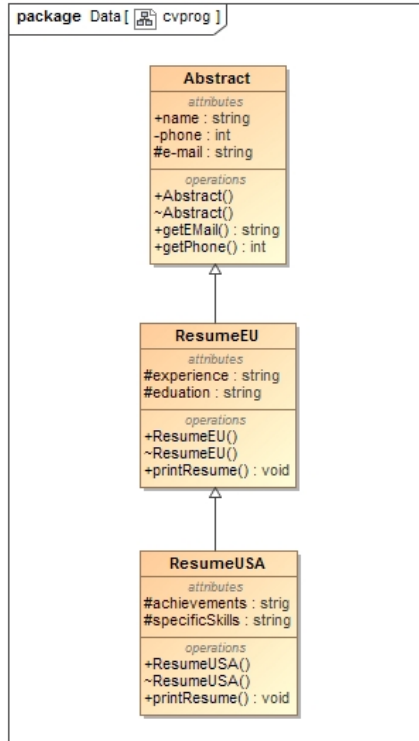


Рисунок 3.3 – Діаграма класів для програми з лістингу 3.1

– класи (**ResumeEU** і **ResumeUSA**) визначено аналогічно, тільки в них відсутні **get**-функції, але є функції виведення даних на друк;

– синтаксис спадкування бачимо у рядку 23 — після імені похідного класу **ResumeEU** стоїть двокрапка і потім специфікатор доступу **protected** та ім'я базового класу **Abstract**. У цьому випадку використання специфікатора доступу **protected** означає, що змінні-члени **name** і **e_mail**, а також функції-члени класу **Abstract** будуть захищеними, тобто вони можуть бути використані всередині похідного класу, але будуть недоступними поза класом (принцип інкапсуляції);

– внаслідок успадкування всі змінні-члени та функції-члени класу **Abstract** стають відповідними елементами класу **ResumeEU**;

– клас **ResumeEU** «багатіший» за клас **Abstract**, оскільки він має властивості класу **Abstract** та власні властивості (змінні-члени **experience** та **education**). До успадкованих елементів клас **ResumeUA** має такий самий доступ, як і до своїх власних. Це видно з функції друку (рядки 38-44);

– клас **ResumeUSA** успадковує всі властивості класу **ResumeEU**, а отже опосередковано і всі властивості класу **Abstract**. У цьому ми переконуємося, коли запустимо програму на виконання;

– зверніть увагу на рядок 67. Назви функцій виведення на друк у класів **ResumeEU** та **ResumeUSA** однакові. Для вказівки, яку саме функцію викликати в даному випадку, використовується операція розширення видимості (**::**), яка стоїть після імені класу, чия функція (це може бути член класу) повинна бути підставлена компілятором.

Лістинг 3.1 Приклад простого спадкування

```

1  #include <iostream>
2  using namespace std;
3
4  class Abstract{
5  public: string name;
6  private: int phone;
7  protected: string e_mail;
8  public:
```

```

9      Abstract(){
10          cout << "constructor Abstract\n";
11          cout << "Enter name: ";   cin >> name;
12          cout << "Enter phone: ";  cin >> phone;
13          cout << "Enter e-mail: "; cin >> e_mail;
14      }
15
16      ~Abstract(){
17          cout << "desructor Abstract\n";
18      }
19      string getEMail() {return e_mail;}
20      int  getPhone() {return phone;}
21 };
22
23 class ResumeEU : protected Abstract{
24 protected:
25     string experience;
26     string education;
27 public:
28     ResumeEU(){
29         cout << "constructor ResumeEU\n";
30         cout << "Enter experience: "; cin >>
            experience;
31         cout << "Enter education: ";  cin >>
            education;
32     }
33
34     ~ResumeEU(){
35         cout << "destructor ResumeEU\n";
36     }
37
38     void printResume(){
39         cout << "name\t\t" << name<<endl;
40         cout << "phone\t\t" << getPhone()<<endl;
41         cout << "e-mail\t\t" << e_mail<<endl;
42         cout << "experience\t" << experience <<
            endl;
43         cout << "education\t" << education << endl;
44     }
45 };

```

```

46 class ResumeUSA : protected ResumeEU{
47 protected:
48     string achievements;
49     string specificSkills;
50 public:
51     ResumeUSA()
52     {
53         cout << "constructor ResumeUSA\n";
54         cout << "Enter your achievements: ";
55         //e.g. Worked on special projects
56         cin >> achievements;
57         cout << "Enter your specificSkills: ";
58         //e.g. Programmer on language Lisp
59         cin >> specificSkills;
60     }
61     ~ResumeUSA()
62     {
63         cout << "destructor ResumeUSA\n";
64     }
65     void printResume()
66     {
67         ResumeEU::printResume();
68         cout << "achievements\t" << achievements <<
            endl;
69         cout << "specificSkills\t" <<
            specificSkills << endl;
70     }
71 };
72
73 int main() {
74     cout << "\nInput ResumeEU:\n";
75     ResumeEU myResumeEU;
76     cout << "\nOutput ResumeEU:\n";
77     myResumeEU.printResume();
78     cout << "\nInput ResumeUSA:\n";
79     ResumeUSA myResumeUSA;
80     cout << "\nOutput ResumeUSA:\n";
81     myResumeUSA.printResume();
82     return 0;
83 }

```

Результат роботи програми з лістингу 3.1 (з точністю до назв) показаний на лістингу 3.2.

Лістинг 3.2 Результат роботи програми з лістингу 3.1

```
Input ResumeEU:
constructor Abstract
Enter name: Petrenko
Enter phone: 0971256789
Enter e-mail: petrenko@gmail.com
constructor ResumeEU
Enter experience: programmer
Enter education: master_of_science
```

```
Output ResumeEU:
name      Petrenko
phone     0971256789
e-mail    petrenko@gmail.com
experience programmer
education master_of_science
```

```
Input ResumeUSA:
constructor Abstract
Enter name: Angolenko
Enter phone: 0993334152
Enter e-mail: angolenko@ukr.net
constructor ResumeEU
Enter experience: admin
Enter education: computer-engineering-ZNTU
constructor ResumeUSA
Enter you achievements: Worked_on_special_projects
Enter you specificSkills:
    programmer_on_language_Lisp
```

```
Output ResumeUSA:
name      Angolenko
phone     0993334152
e-mail    angolenko@ukr.net
experience admin
education computer-engineering-ZNTU
```

```

achievements    Worked_on_special_projects
specificSkills  programmer_on_language_Lisp
destructor      ResumeUSA
destructor      ResumeEU
desructor       Abstract
destructor      ResumeEU
desructor       Abstract

```

Завдання до програми

1. Реалізуйте та вивчіть програму з лістингу 3.1. Продемонструйте працездатність програми;
2. Розробіть та продемонструйте роботу власного прикладу програми, що моделює фрагмент діаграми на рис. 3.2, який включає класи `Abstract`, `ResumeEU`, `CurriculumVitae` та `FranceCV`.

3.2 Конструктори при спадкуванні

У цьому розділі ми розглянемо, як ініціалізуються члени базових і похідних класів при спадкуванні.

У програмі основна увага приділена варіантам конструкторів при опосередкованому спадкуванні і прямому множинному спадкуванні, а також їх комбінації. У цій програмі також показано форми доступу, використання та захисту членів успадкованих класів. Реалізація програми та наведені нижче завдання до неї спрямовані на закріплення навичок організації ієрархії класів.

3.2.1 Програма: класи `Text`, `Book`, `Novel` тощо

У проекті (див. лістинги 3.3-3.8) окремо представлені класи програми, між якими є відносини спадкування. Лістинг 3.9 містить код головної функції програми. Для демонстрації порядку викликів конструкторів та деструкторів вони мають оператори виведення на консоль відповідних повідомлень.

ЛІСТИНГ 3.3 Клас Text

```
1  class Text{
2      int words;
3      string language;
4  protected:
5      void showInfo(){
6          cout<<"words: "<<words<<" , language: "
              "<<language<<endl;
7      }
8  public:
9      Text(int ws, string lg):words(ws),
              language(lg){
10         cout << "\nConstructor of Text\n";
11     }
12     ~Text(){
13         cout << "Destructor of Text\n";
14     }
15 };
```

ЛІСТИНГ 3.4 Клас Book

```
1  class Book{
2      int pages;
3      string title;
4  protected:
5      void showInfo(){
6          cout << "title: " << title << " , pages: " <<
              pages << endl;
7      }
8  public:
9      Book(int pg, string tl): pages(pg), title(tl){
10         cout << "\nConstructor of Book\n";
11     }
12     ~Book(){
13         cout << "Destructor of Book\n";
14     }
15 };
```

ЛІСТИНГ 3.5 Клас Novel

```

1  class Novel:protected Book, protected Text{
2      string author;
3  public:
4      Novel(string a, int p, string t, int w, string
        l) : author(a), Book(p,t), Text(w,l){
5          cout << "\nConstructor of Novel\n";
6      }
7      ~Novel(){
8          cout << "Destructor of Novel\n";
9      }
10     void showInfo(){
11         cout << "Novel:\n";
12         cout << "author: " << author << endl;
13         Book::showInfo();
14         Text::showInfo();
15     }
16 };

```

ЛІСТИНГ 3.6 Клас TechnicalText

```

1  class TechnicalText : protected Text{
2      int terms;
3      int charts;
4  protected:
5      void showInfo(){
6          cout << "terms:" << terms << ", charts: " <<
            charts << endl;
7          Text::showInfo();
8      }
9  public:
10     TechnicalText(int tr, int ch, int ws, string
        lg): terms(tr),charts(ch),Text(ws,lg){
11         cout << "\nConstructor of TechnicalText\n";
12     }
13     ~TechnicalText(){
14         cout << "Destructor of TechnicalText\n";
15     }
16 };

```

Лістинг 3.7 Клас ExerciseBook

```

1  class ExerciseBook : protected Book{
2      string owner;
3  public:
4      ExerciseBook(string o, int p, string t):owner
        (o), Book(p,t){
5          cout << "\nConstructor of ExerciseBook";
6      }
7      ~ExerciseBook(){
8          cout << "Destructor of ExerciseBook\n";
9      }
10     void showInfo(){
11         cout << "\nExercise Book:\n";
12         cout << "owner: " << owner << endl;
13         Book::showInfo();
14     }
15 };

```

Лістинг 3.8 Клас Manual

```

1  class Manual:protected Book, protected
    TechnicalText{
2      int instructions;
3  public:
4      Manual(int i,int p, string t, int tr, int ch,
        int ws, string lg):
5          instructions(i), Book(p, t), TechnicalText(tr,
        ch, ws, lg){
6          cout << "\nConstructor of Manual\n";
7      }
8      ~Manual(){
9          cout << "Destructor of Manual\n";
10     }
11     void showInfo(){
12         cout << "Manual:\n";
13         Book::showInfo();
14         cout <<"instructions: "<<instructions<<endl;
15         TechnicalText::showInfo();
16     }
17 };

```

Лістинг 3.9 Головна функція програми

```

1  int main() {
2      ExerciseBook eb("Ivan Student", 96, " C++");
3      eb.showInfo();
4      Novel nv("J. Barnes",102, "Metroland", 51120,
5              "en");
6      nv.showInfo();
7      Manual man(222,22,"Phone",45, 5 , 1000, "en");
8      man.showInfo();
9      return 0;
10 }
```

В результаті правильної реалізації програми та її виконання на консолі повинні вивестись повідомлення, показані на лістингу 3.10.

Лістинг 3.10 Результат роботи програми

```

Constructor of Book

Constructor of ExerciseBook
Exercise Book:
owner: Ivan Student
title:  C++, pages: 96

Constructor of Book

Constructor of Text

Constructor of Novel
Novel:
author: J. Barnes
title: Metroland, pages: 102
words: 51120, language: en

Constructor of Book

Constructor of Text

Constructor of TechnicalText
```

```

Constructor of Manual
Manual:
title: Phone, pages: 22
instructions: 222
terms:45, charts: 5
words: 1000, language: en

```

```

Destructor of Manual
Destructor of TechnicalText
Destructor of Text
Destructor of Book
Destructor of Novel
Destructor of Text
Destructor of Book
Destructor of ExerciseBook
Destructor of Book

```

Завдання до програми

1. Реалізуйте та вивчіть програмний проект. Виконайте в проєкті наступні завдання та продемонструйте його працездатність із внесеними змінами:

- вивчіть програми проєкту і перш ніж його реалізовувати, побудуйте UML-діаграму класів даного проєкту, вказавши в секціях кожного класу змінні-члени та функції-члени класів, а також їх видимість (+, -, #);

- реалізуйте програму, зібравши наведені коди класів в один файл або у вигляді багатобайтового проєкту, та продемонструйте її роботу;

- поясніть порядок виконання конструкторів та деструкторів об'єктів в результаті роботи програми;

- поясніть, як ініціалізуються члени класів, що утворюють ієрархію класу **Manual**;

- поясніть призначення та мету обраного в програмі розміщення специфікаторів доступу. Для довідки правила визначення прав доступу до членів класів при спадкуванні наведені в табл. 3.1;

2. *Розробіть та продемонструйте роботу власного прикладу програми, в якому розробіть додатковий клас, що є спадкоємцем класу Novel.

Таблиця 3.1 – Права доступу при спадкуванні

Специфікатор у базовому класі	Специфікатор перед базовим класом	Права доступу у похідному класі
public	public	public
protected	public	protected
private	public	недоступний
public	protected	protected
protected	protected	protected
private	protected	недоступний
public	private	private
protected	private	private
private	private	недоступний

3.3 Віртуальні функції та абстрактні класи

Віртуальні функції, разом із механізмом спадкування та вказівниками на об'єкти класів, забезпечують реалізацію *динамічного поліморфізму*, який ще називають *пізнім зв'язуванням*. Динамічний поліморфізм можна розуміти як можливість змінювати функціональність програми під час її виконання, тобто поліморфізм під час роботи програми. На відміну від динамічного, *статичний поліморфізм* або *раннє зв'язування* — це зміна поведінки програми за рахунок перевантаження функцій та/або операцій, які, як ми знаємо, реалізуються на етапі компіляції програми.

Динамічний поліморфізм — це потужний механізм об'єктно-орієнтованого програмування, який дозволяє створювати надзвичайно гнучкі програми та застосунки.

3.3.1 Програма: класи Bank, Client, BusinessLogic

У цій програмі демонструється застосування та реалізація динамічного поліморфізму. Сценарій, який моделює програма, є наступним. Є два банки та клієнти, які здійснюють з цими банками операції отримання кредиту та відкриття депозиту. Залежно від поточної ситуації відсоткові ставки кредиту та депозиту змінюються. Клієнт, збираючись узяти кредит та/або відкрити депозит, вибирає банк із найбільш привабливими для себе поточними відсотковими ставками за кредитом та/або депозитом.

Програма включає такі класи (див. UML-діаграму на рис. 3.4):

- **Bank** — встановлює процентні ставки за кредитом та депозитом;
- **Bank1Department** та **Bank2Department** — класи, що імітують служби (підрозділи) першого та другого банків, відповідно. Ці служби безпосередньо оформлюють кредити та депозити;
- **Client** — клас, екземпляри якого є клієнтами банків. Клас включає змінні-члени, що містять ім'я клієнта, суми коштів, які клієнт збирається отримати в кредит і вкласти в депозит, вказівники на екземпляри оформлених кредитів та депозитів. У класі є необхідні функції-члени для доступу до членів даних та конструктор з параметрами;
- **BusinessLogic** — клас, що дозволяє клієнту вибрати вигідний у поточній ситуації банк для кредиту та депозиту. Клас містить змінні-члени — вказівники на створювані службами банку кредити та депозити. Значення цих вказівників надаються відповідним вказівникам класу **Client**. Безпосередній вибір банку та виду послуги виконується у функції `choice()`;
- класи **CreditBank1**, **DepositBank1**, **CreditBank2** та **DepositBank2** типізують конкретні об'єкти оформлених кредитів та депозитів, відповідних банків;
- абстрактний клас **AbstractBankDepartment** є базовим для класів **Bank1Department** та **Bank2Department**, в яких перевантажуються його *чисто віртуальні функції* `createCredit()` і `createDeposit()`;
- абстрактний клас **AbstractService** є базовим для чотирьох

класів (**CreditBank1** тощо), які представляють кредити та депозити банків. У цих класах перевантажується його чисто віртуальна функція `getInfo()`.

Абстрактні класи даного прикладу та їх спадкоємці реалізують так званий *патерн* (шаблон) проектування «*Абстрактна фабрика*» [8]. На UML-діаграмі (рис. 3.4) такі класи виділені кольором. Ідея цього шаблону проектування полягає в тому, щоб програма була незалежною від процесу створення нових об'єктів та їх типів. У нашому прикладі базовий клас **AbstractBankDepartment** є поліморфним і є Абстрактною фабрикою. Він містить фабричні методи створення всіх типів об'єктів (у разі кредитів і депозитів) – чисто віртуальні функції `createCredit()` і `createDeposit()`.

Вказівник цього базового класу може приймати значення адрес похідних класів **Bank1Department** та **Bank2Department**, які можна назвати конкретними фабриками. Залежно від поточного стану клієнт може вибрати в якому банку брати кредит або відкривати депозит. В такому разі до вказівника базового класу заноситься адреса відповідної конкретної фабрики, яка створює конкретний об'єкт – кредит або депозит. Об'єкти кредитів та депозитів мають загальний базовий абстрактний клас **AbstractService**, тому конкретні фабрики не залежать від типів об'єктів, що створюються.

Нижче наведено список файлів (див. лістинги 3.11-3.24), які містять коди перерахованих класів і призначені для створення багатофайлового проекту. На UML-діаграмі показані основні відносини між класами, включаючи спадкування, залежності (із стереотипами «use» та «create») та композиції.

Лістинг 3.11 Головна функція програми

```

1  #include "businesslogic.h"
2  int main() {
3      static const int n = 5;
4      srand(time(0));
5
6      string names[n] =
          {"Abramob", "Borisov", "Verbina",
           "Grishin", "Dubovik"};

```

```

7      int psc[n] = {20000, 21000, 40000, 15000,
      25000}; //principal sum for credit
8      int psd[n] = {41000, 33000, 22000, 55000,
      35000}; //principal sum for deposit
9
10     Client* client[n];
11
12     for(int i =0; i < n; i++)
13         client[i] = new
            Client(names[i],psc[i],psd[i]);
14     int mc1 = 30; //minimal rate of credit of the
            bank-1 (30..38)
15     int md1 = 18; //minimal rate of deposit of
            the bank-1 (18..26)
16     int mc2 = 29; //minimal rate of credit of the
            bank-2 (29..37)
17     int md2 = 17; //minimal rate of deposit of
            the bank-2 (17..25)
18
19     Bank b1(mc1,md1);
20     Bank b2(mc2,md2);
21
22     BusinessLogic bs;
23
24     for(int i = 0; i < n; i++)
25     {
26         bs.choice(b1, b2, client[i]);
27         client[i]-> credit -> getInfo();
28         client[i]-> deposit -> getInfo();
29     }
30
31     for(int i = 0; i < n; i++)
32     {
33         delete client[i] -> credit;
34         delete client[i] -> deposit;
35         delete client[i];
36     }
37     return 0;
38 }

```

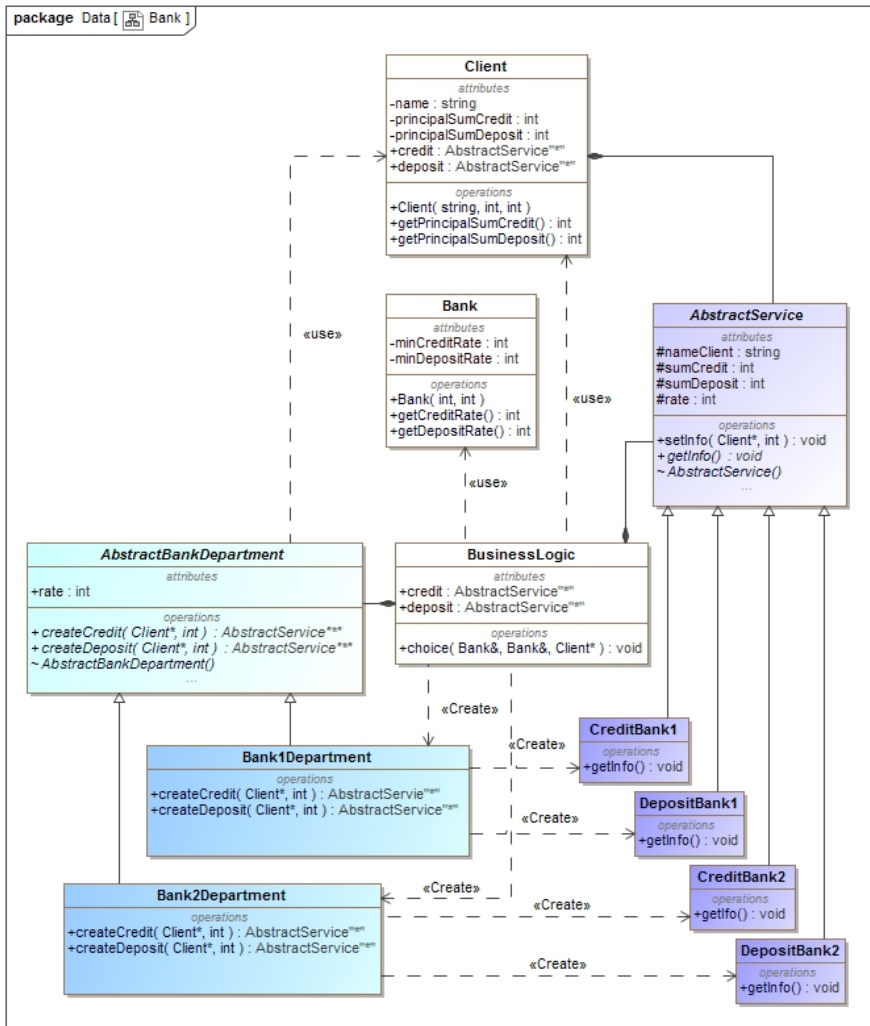


Рисунок 3.4 – Діаграма класів програми

ЛІСТИНГ 3.12 Файл client.h

```
1  #if !defined(CLIENT_H)
2  #define CLIENT_H
3
4  #include <iostream>
5
6  using namespace std;
7
8  class AbstractService;
9
10 class Client{
11     string name;
12     int principalSumCredit;
13     int principalSumDeposit;
14
15 public:
16     Client(string, int, int);
17
18     string getName();
19     int getPrincipalSumCredit();
20     int getPrincipalSumDeposit();
21
22     AbstractService* credit;
23     AbstractService* deposit;
24 };
25 #endif
```

ЛІСТИНГ 3.13 Файл client.cpp

```
1  #include "client.h"
2
3  Client::Client(string nm, int psc, int
    psd):name(nm),
4      principalSumCredit(psc),
5      principalSumDeposit(psd) { }
6
7  string Client::getName(){return name;}
8
9  int Client::getPrincipalSumCredit(){
```

```

9     return principalSumCredit;
10 }
11
12 int Client::getPrincipalSumDeposit(){
13     return principalSumDeposit;
14 }

```

Лістинг 3.14 Файл bank.h

```

1  #if !defined(BANK_H)
2  #define BANK_H
3  #include <cstdlib>
4  #include <ctime>
5
6  class Bank{
7  protected:
8      int minCreditRate;
9      int minDepositRate;
10 public:
11     Bank(int, int);
12     int getCreditRate();
13     int getDepositRate();
14 };
15 #endif

```

Лістинг 3.15 Файл bank.cpp

```

1  #include "bank.h"
2
3  Bank::Bank(int mc, int md):minCreditRate(mc),
4      minDepositRate(md) { }
5
6  int Bank::getCreditRate(){
7      return minCreditRate + rand()%9;
8  }
9
10 int Bank::getDepositRate(){
11     return minDepositRate + rand()%9;
12 }

```

Лістинг 3.16 Файл abstractbankdep.h

```
1  #if !defined(A_BANKDEP_H)
2  #define A_BANKDEP_H
3  #include "abstractservice.h"
4
5  class AbstractBankDepartment{
6  public:
7      int rate;
8      virtual AbstractService* createCredit(Client*,
9          int) = 0;
10     virtual AbstractService*
11         createDeposit(Client*, int) = 0;
12     virtual ~AbstractBankDepartment();
13 };
14 #endif
```

Лістинг 3.17 Файл abstractbankdep.cpp

```
1  #if !defined(BANK_DEPS_H)
2  #define BANK_DEPS_H
3  #include "abstractbankdep.h"
4  #include "concreteservice.h"
5
6  class Bank1Department: public
7      AbstractBankDepartment{
8  public:
9      AbstractService* createCredit(Client*, int);
10     AbstractService* createDeposit(Client*, int);
11 };
12
13 class Bank2Department: public
14     AbstractBankDepartment{
15 public:
16     AbstractService* createCredit(Client*, int);
17     AbstractService* createDeposit(Client*, int);
18 };
19 #endif
```

ЛІСТИНГ 3.18 Файл concretebankdep.cpp

```
1  #include "concretebankdep.h"
2
3  AbstractService*
4      Bank1Department::createCredit(Client* c, int
5      rate){
6      CreditBank1* cb1 = new CreditBank1;
7      cb1->setInfo(c,rate);
8      return cb1;
9  }
10
11 AbstractService*
12     Bank1Department::createDeposit(Client* c, int
13     rate){
14     DepositBank1* db1 = new DepositBank1;
15     db1->setInfo(c,rate);
16     return db1;
17 }
18
19 AbstractService*
20     Bank2Department::createCredit(Client* c, int
21     rate){
22     CreditBank2* cb2 = new CreditBank2;
23     cb2->setInfo(c,rate);
24     return cb2;
25 }
26
27 AbstractService*
28     Bank2Department::createDeposit(Client* c, int
29     rate){
30     DepositBank2* db2 = new DepositBank2;
31     db2->setInfo(c,rate);
32     return db2;
33 }
```

ЛІСТИНГ 3.19 Файл abstractservice.h

```
1  #if !defined(SERVICE_H)
2  #define SERVICE_H
```

```

3  #include <iostream>
4  #include "client.h"
5
6  using namespace std;
7
8  class AbstractService{
9  protected:
10     string nameClient;
11     int sumCredit;
12     int sumDeposit;
13     int rate;
14 public:
15     virtual void getInfo() = 0;
16     virtual ~AbstractService();
17     void setInfo(Client*, int);
18 };

```

Лістинг 3.20 Файл abstractservice.cpp

```

1  #include "abstractservice.h"
2
3  AbstractService::~~AbstractService(){ }
4
5  void AbstractService::setInfo(Client* c, int r){
6     nameClient = c -> getName();
7     sumCredit = c -> getPrincipalSumCredit();
8     sumDeposit = c -> getPrincipalSumDeposit();
9     rate = r;
10 }

```

Лістинг 3.21 Файл concreteservice.h

```

1  #if !defined(C_SERVICE_H)
2  #define C_SERVICE_H
3  #include "abstractservice.h"
4
5  class CreditBank1: public AbstractService{
6  public:
7     void getInfo();
8 };

```

```

9  class CreditBank2: public AbstractService{
10 public:
11     void getInfo();
12 };
13
14 class DepositBank1: public AbstractService{
15 public:
16     void getInfo();
17 };
18
19 class DepositBank2: public AbstractService{
20 public:
21     void getInfo();
22 };
23 #endif

```

ЛІСТИНГ 3.22 Файл concreteservice.cpp

```

1  #include "concreteservice.h"
2
3  void CreditBank1::getInfo(){
4      cout << nameClient << ". Credit of Bank1: ";
5      cout << "sum " << sumCredit << " rate " <<
        rate << "%\n";
6  }
7  void CreditBank2::getInfo(){
8      cout << nameClient << ". Credit of Bank2: ";
9      cout << "sum " << sumCredit << " rate " <<
        rate << "%\n";
10 }
11 void DepositBank1::getInfo(){
12     cout << nameClient << ". Deposit of Bank1: ";
13     cout << "sum " << sumDeposit << " rate " <<
        rate << "%\n";
14 }
15 void DepositBank2::getInfo(){
16     cout << nameClient << ". Deposit of Bank2: ";
17     cout << "sum " << sumDeposit << " rate " <<
        rate << "%\n";
18 }

```

Лістинг 3.23 Файл businesslogic.h

```
1 #if !defined(BUSINESS_H)
2 #define BUSINESS_H
3 #include "bank.h"
4 #include "concretebankdep.h"
5 class BusinessLogic{
6     AbstractService* credit;
7     AbstractService* deposit;
8 public: void choice(Bank&, Bank&, Client*);
9 };
10 #endif
```

Лістинг 3.24 Файл businesslogic.cpp

```
1 #include "businesslogic.h"
2 void BusinessLogic::choice(Bank& b1, Bank& b2,
3     Client* c){
4     int c1 = b1.getCreditRate();
5     int d1 = b2.getDepositRate();
6     int c2 = b1.getCreditRate();
7     int d2 = b2.getDepositRate();
8     Bank1Department* ptrB1 = new Bank1Department;
9     Bank2Department* ptrB2 = new Bank2Department;
10    if (c1 <= c2){
11        credit = ptrB1 -> createCredit(c,c1);
12    }
13    else{
14        credit = ptrB2 -> createCredit(c,c2);
15    }
16    if(d1 >= d2){
17        deposit = ptrB1 -> createDeposit(c,d1);
18    }
19    else{
20        deposit = ptrB2 -> createDeposit(c,d2);
21    }
22    c->credit = credit;
23    c->deposit = deposit;
24    delete ptrB1;
25    delete ptrB2;
26 }
```

Завдання до програми

1. Реалізуйте та вивчіть програмний проект (див. лістинги 3.11-3.24). Виконайте в проекті наступні завдання та продемонструйте його працездатність із внесеними змінами:

- вивчіть UML-діаграму класів програми (рис. 3.4) та реалізуйте програму у вигляді багатофайлового проекту;
- визначте у всіх класах деструктори з виразами для виведення на друк повідомлень про їхню роботу, поясніть порядок їх спрацьовування, особливо звернувши увагу на роботу віртуальних деструкторів;
- додайте до програми віртуальні функції для обчислення доходів банку від наданого кредиту та клієнта від депозиту за певний проміжок часу, значення якого має вводитися з клавіатури. Обґрунтуйте вибір класу, до якого поміщені ці функції. Результати роботи цих функцій повинні виводитися за допомогою функції `getInfo()`;

2. *Розробіть та продемонструйте роботу власного прикладу програми, в якому:

- додайте можливість виконання банками ще однієї операції, наприклад, іпотеки чи інвестування. Для цього спочатку доопрацюйте UML-діаграму згідно з передбачуваними змінами програми, а потім внесіть ці зміни до програми. Реалізація повинна відповідати шаблону проектування Абстрактна фабрика;
- реалізуйте² сукупність клієнтів у вигляді *зв'язаного списку* з можливістю створювати необмежену кількість клієнтів та вводити дані клієнта з консолі.

3.4 Зміст звіту

Звіт повинен містити: мету роботи, лістинги або фрагменти коду, що розроблені для власного прикладу програм, з необхідними поясненнями та відповіді на контрольні питання.

²Цей пункт завдання не входить до обов'язкових для заліку лабораторної роботи

3.5 Контрольні питання

1. Що таке спадкування в ООП? Як називаються класи, чії властивості успадковуються, та класи, які успадковують?
2. Які компоненти базового класу можна успадковувати?
3. Який порядок спрацьовування конструкторів та деструкторів класів, які перебувають у спадкуванні?
4. Як отримати доступ до компонентів конкретного класу, якщо класи знаходяться у спадкуванні, а компоненти мають однакові ідентифікатори? Для чого при спадкуванні класів використовують специфікатор доступу `protected`?
5. Що таке непряме/пряме множинне спадкування? Навести приклади.
6. На прикладах показати механізм ініціалізації членів вищих в ієрархії успадкування класів при непрямому і прямому множинному успадкуванні.
7. Як подолати невизначеність, що виникає при ромбоподібному спадкуванні із застосуванням (або без використання) віртуальних класів? Навести приклади.
8. Функції-члени якого класу будуть доступними через вказівник базового класу, якщо вони оголошені невіртуальними (або віртуальними)?
9. Що таке поліморфізм? В чому відмінність між динамічним та статичним поліморфізмом. Як вони реалізуються в мові C++ і чим викликана необхідність їх застосування?
10. Для чого використовується віртуальні функції, віртуальний деструктор, віртуальні класи? Що таке чисто віртуальна функція?
11. Що таке абстрактний клас? Чи може він містити членидані та невіртуальні функції? Якщо так, то навести приклад.
12. Чи можна у похідному класі, не перевизначати успадковану чисто віртуальну функцію? Якщо так, то навести приклад такого спадкування та головну функцію, що використовує цей клас.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

- [1] Horton I., Weert P. Beginning C++23: From Beginner to Pro. 7-th ed. / I. Horton, P. Weert.– Apress, 2023.
- [2] Gregoire M. Professional C++. 6-th ed. / M. Gregoire. – Wrox, 2024.
- [3] Tutorials: C++ Language. URL: <https://cplusplus.com/doc/tutorial/>.
- [4] International Standard ISO/IEC 14882:2020(E). Programming Language C++. URL: <https://isocpp.org/std/the-standard>.
- [5] Standard C++ Library reference. URL: <https://cplusplus.com/reference/>.
- [6] Get Started! URL: <https://isocpp.org/get-started>.
- [7] Мартін Р. Чистий код. Створення і рефакторинг за допомогою Agile / Р. Мартин; [пер. з англ. Г. Якубовська]. – Харків : “Ранок” : Фабула, 2024. – 448 с.
- [8] Head First. Патерни проєктування / Е. Фрімен, Е. Робсон, К. Сієрра, Б. Бейтс; [пер. з англ. Г. Якубовська]. – Харків : ВД : Фабула, 2022. – 672 с.