МІНІСТЕРСТВО ОСВІТИ, НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС
"ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ"
НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

МЕТОДИЧНІ ВКАЗІВКИ
до комп’ютерного практикуму

для студентів напрямів підготовки
6.040303 «Системний аналіз»,
6.040302 «Інформатика»,
6.050101 «Комп’ютерні науки»

Затверджено Методичною радою НТУУ «КПІ»

Назарчук І.В.

Київ
НТУУ “КПІ”
2011

Об’єктно-орієнтоване програмування: Методичні вказівки до комп’ютерного практикуму для студентів напрямів підготовки 6.040303 «Системний аналіз», 6.040302 «Інформатика», 6.050101 «Комп’ютерні науки» / Уклад.: І.В. Назарчук. – К. НТУУ “КПІ”, 2011. – 72 с.

Гріф надано Методичною радою НТУУ «КПІ»
(Протокол № від 2011р.)

ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

МЕТОДИЧНІ ВКАЗІВКИ
до комп’ютерного практикуму

для студентів напрямів підготовки
6.040303 «Системний аналіз»,
6.040302 «Інформатика»,
6.050101 «Комп’ютерні науки»

Укладач: Назарчук Ірина Василівна

Відповідальний редактор: В.Д. Романенко, д.т.н, професор

Рецензент: Д.Г. Діденко, к.т.н.

ВСТУП

Дисципліна «Об’єктно - орієнтоване програмування» входить до нормативної частини навчального плану підготовки бакалаврів напрямів 6.040302 «Інформатика» та 6.050101 «Комп’ютерні науки» і належить до циклу професійної та практичної підготовки, а також до вибіркової частини навчального плану підготовки бакалаврів напряму 6.040303 «Системний аналіз» і належить до циклу дисциплін за вибором ВНЗ. Забезпечується дисциплінами «Програмування та алгоритмічні мови», «Структури даних та алгоритми», «Теорія програмування» та забезпечує дисципліну «Бази даних та інформаційні системи», а також усі спеціальні курси, що потребують комп’ютерного моделювання.

Мета робіт комп’ютерного практикуму – формування вмінь і навичок, необхідних для раціонального використання засобів об’єктно - орієнтованого програмування.

Основні завдання — ознайомлення з існуючим інструментарієм створення складних програмних систем для різноманітних предметних областей, формування практичних навичок розробки, створення та налагодження програмного забезпечення з використанням сучасних об’єктно-орієнтованих мов програмування та стандартних бібліотек.

У методичних вказівках подано теоретичний матеріал та індивідуальні завдання на комп’ютерні практикуми з базових тем курсу. Мета кожної роботи полягає в опануванні тем, винесених в підзаголовок. Порядок виконання, список контрольних запитань та список рекомендованої літератури надаються в описі кожної роботи. Мови програмування – C++ та C#. До кожної роботи слід скласти звіт єдиного вигляду, наведеного в додатку.

КОМП'ЮТЕРНИЙ ПРАКТИКУМ № 1. КОНСТРУЮВАННЯ ОБ'ЄКТІВ КОНКРЕТНИХ КЛАСІВ

(до теми «Абстрагування та інкапсуляція»)

Мета роботи: Навчитись правильно описувати абстракції предметної області з використанням принципу інкапсуляції мовою *C++*.

Основні теоретичні відомості

Об'єктна декомпозиція, абстракція, опис абстракції. В основі об'єктно-орієнтованого підходу до програмування лежить об'єктна декомпозиція: система розділена на елементи, кожний з яких належить до певної абстракції предметної області і характеризується своєю власною поведінкою, а поведінка системи забезпечується взаємодією цих автономних елементів.

У мовах програмування конкретним виявом певної абстракції є тип. В об'єктно-орієнтованих мовах існують механізми, що дозволяють створювати нові типи – типи користувача. В *C++*, наприклад, це класи.

Синтаксис опису класу. Конструкція *class X{ ... };* називається визначенням класу, вона утворює новий тип.

Повний опис класу складається з трьох частин: оголошень полів даних, прототипів функцій-членів, визначень функцій-членів. Наприклад:

```
class Point                          // опис класу
{
   int x,y;                          //данні стану
public:
   void SetX(int);                   //прототип функції-члена
   int GetX(){return x;};            //прототип і визначення функції-
};                                   члена //всередині опису класу
void  Point:: SetX(int a){x=a;};     //визначення функції-члена за
```

```
                                        //межами класу
void main()
{
Point a, b;    . . .;}                  //об'єкти
```

Поля даних – екземпляри змінних стану, які характеризують об'єкт класу. Оголошуються всередині опису класу і усі екземпляри об'єктів даного класу будуть мати однакову множину змінних стану.

Функції-члени – це методи, які обробляють повідомлення, що передаються об'єктам класів (визначають відповідальність абстракції). Прототипи (оголошення) цих функцій повинні знаходитись всередині опису класу, визначення (опис) їх може бути і всередині і за межами класу. Якщо опис функцій дається за межами класу, необхідно вказувати ім'я класу разом з оператором дозволу області видимості ***«::»,*** оскільки різні класи можуть мати функції-члени з однаковими назвами.

В оголошенні класу можуть використовуватись помітки *public, private* та *protected,* які визначають права доступу до відповідних розділів тіла класу (відкритий, закритий, захищений). Порядок розташування та кількість цих поміток в описі класу – довільна. За умовчанням тип доступу *private.*

Відкритий *(public)* розділ опису класу утворює відкритий інтерфейс для його об'єктів: значення закритих даних класу можна змінити тільки скориставшись відкритими функціями-членами цього класу (змінити свій стан об'єкт може тільки опрацювавши допустиме для нього повідомлення – тим методом, який оголошений у відкритій частині класу).

Оскільки клас *(class)* уводить новий тип даних (у наведеному прикладі *Point*), оголошення об'єкту цього класу *(a,b)*– це оголошення змінних такого типу *(Point),* аналогічне декларуванню змінних стандартних типів перед їх використанням.

Класифікація функцій – членів. Кожний клас має свій набір функцій-членів (методів), які характеризують поведінку об'єктів цього класу. Усі ці функції можна умовно віднести до таких категорій:

- Конструктори – відповідають за створення та ініціалізацію об'єкта;
- деструктор – руйнує об'єкт;
- конструктори копіювання – для копіювання об'єктів;
- селектори – для доступу до закритих (або захищених) частин об'єкта без зміни їх значень;
- модифікатори – для зміни стану об'єкта.

За умови відсутності функцій-членів для нового класу, компілятор надає йому чотири методи, які називаються канонічними методами. Це конструктор умовчання, конструктор копій, деструктор, оператор надання (присвоювання).

Назва конструктора збігається з назвою класу, вказувати тип значення, що повертається не потрібно. В класі може бути декілька конструкторів, вони повинні розрізнятися тільки за кількістю та типами параметрів. Наприклад, у розглянутому раніше класі *Point* можуть бути такі конструктори:

```
Point();
Point(int);
Point(int, int);
Point(char*);
```

Якщо об'єкт потрібно ініціалізувати як копію існуючого, використовують конструктор копіювання. Для класу *Point* конструктор копіювання матиме наступний вигляд:

```
Point :: Point ( const Point & )
```

Конструктори ініціалізують об'єкт, інколи з захопленням певних ресурсів (пам'ять, файл), які потрібно звільнити після їх використання. Функція, яка гарантує правильне знищення об'єкту називається деструктором.

Назва деструктора збігається з іменем класу, перед яким ставлять «тильду», наприклад:

~Point();

Функції-селектори призначені для перевірки стану об'єкта, вони повертають значення полів, розташованих в захищених або закритих частинах класу. Наприклад,

int GetX(){return x;};

Функції-модифікатори призначені для зміни стану об'єкта, вони можуть не повертати значення *(void)* або повертати посилання на модифікований об'єкт, наприклад:

*Point& SetX(int a) { x=a; return *this;};*

Запис ***this*** означає об'єкт, для якого викликана функція-член.

Статичні данні-члени – глобальні змінні, які діють в межах одного класу. Існує лише одна копія такої змінної, яка належить класу (кожний об'єкт має власну копію нестатичного поля). Приклад оголошення:

class Point

{ . . . static int count;. . .};. . .

int Point:: count=0; //обов'язкова ініціалізація статичної змінної

Статичні функції-члени працюють тільки з членами класу, а не конкретного об'єкта класу.

Доступ до статичних членів класу відбувається так само, як і до звичайних нестатичних. Крім того, замість ідентифікатора об'єкта можна використати ім'я самого класу, наприклад: *Point:: set_count(5);*

***Inline*–функцій** – це маленькі функції, звернення до яких компілятор перетворює на їх код, що значно підвищує ефективність роботи програми. Такі функції можна повністю описати в середині опису класу, наприклад

int GetX(){return x;};

або визначити за межами опису класу з ключовим словом *inline*, наприклад

inline int Date::get_year(){ return y;}

Порядок виконання роботи

Спільні вимоги до усіх варіантів.

Визначити тип *Date* як клас, що містить:

- закриті поля цілого типу з інформацією про день, місяць та рік;
- три конструктори - умовчання з використанням поточної дати, конструктор з трьома параметрами, конструктор копіювання;
- деструктор (порожній);
- три функції-селектори для доступу окремо до кожного поля;
- функцію для виведення на екран інформації про дату у форматі dd.mm.yy;
- функції-модифікатори окремо для кожного з полів з повертанням посилання на модифікований об’єкт.

Інші типи описати відповідно варіанту. Якщо це не оговорено окремо, у кожному класі, що розробляється, передбачити:

- конструктори умовчання, з параметрами та копіювання;
- за наявності динамічних полів, у тому числі символьних рядків довільної довжини, у конструкторах передбачити виділення пам’яті;
- деструктор (порожній);
- функції-селектори для доступу окремо до кожного поля;
- функцію для виведення на екран інформації про поточний стан об’єкту у вигляді рядка зі значеннями кожного з полів через пропуск; якщо клас містить масив, інформація про кожний елемент масиву подається окремим рядком;
- функції-модифікатори окремо для кожного з полів з повертанням посилання на модифікований об’єкт.

У тестовому прикладі створити три об’єкти заданого відповідно варіанту типу (для демонстрації виклику кожного конструктора). Значення для параметрів конструктора з параметрами уводити з клавіатури.

Варіант1.

Тип «Освітній рівень» визначити як перерахування *(enum)* із значеннями полів «бакалавр», «спеціаліст», «магістр».

Тип «Людина» визначити як клас, що містить:

- закриті поля з ім’ям та прізвищем, реалізовані у вигляді символьних рядків довільної довжини; з датою народження типу *Date*;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Іспит» визначити як клас, що містить:

- закрите поле з назвою іспиту як символьний рядок фіксованої довжини;
- закриті поля з оцінкою та типу *Date* з датою складання іспиту;
- функції реалізувати відповідно загальним вимогам.

Тип «Студент» визначити як клас, що містить:

- закриті поля типу «Людина» та типу «Освітній рівень»;
- інформацію про здані іспити оформити як динамічний масив (вказівник і розмірність, поля закриті);
- передбачити функцію для додавання чергового іспиту до списку зданих;
- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації з прізвищем та середнім балом;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти типу «Студент» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. До одного з об’єктів додати один екзамен і вивести оновлену інформацію про нього у двох формах.

Варіант2.

Тип «Тип датчика» визначити як перерахування (enum) із значеннями полів «акселерометр», «проксіметр», «температури».

Тип «Датчик» визначити як клас, що містить:

- закрите поле з назвою одиниць вимірювання у вигляді символьного рядка довільної довжини;
- закриті дійсні поля, що визначають діапазон вимірювання та поточне значення фізичної величини;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Пристрій» визначити як клас, що містить:

- закриті поля типу «Датчик», «Тип датчика», цілого типу з номером місця кріплення та типу *Date* з датою калібрування датчика;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Вимірювальний канал» визначити як клас, що містить:

- статичне поле цілого типу з загальною кількістю створених каналів;
- закрите поле цілого типу з порядковим номером каналу; заповнюється автоматично у момент створення чергового об’єкту даного типу;
- інформацію про усі пристрої, що входять до складу даного каналу, оформити як динамічний масив (вказівник і розмірність);
- передбачити функцію для додавання одного пристрою до складу даного вимірювального каналу;
- передбачити функцію з виведенням скороченої інформації з назвою класу та порядковим номером об’єкта;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти типу «Вимірювальний канал» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. До одного з об’єктів додати один пристрій і вивести оновлену інформацію про цей об’єкт.

Варіант3.

Тип «Призначення» визначити як перерахування (enum) із значеннями полів «під забудову», «сільськогосподарського призначення», «зарезервована».

Тип «Власник» визначити як клас, що містить:

- закриті поля з ім’ям та прізвищем, реалізовані у вигляді символьних рядків довільної довжини та датою народження типу *Date*;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Опис» визначити як клас, що містить:

- закриті поля цілого типу із значеннями рівня ґрунтових вод та типу ґрунту;
- закриті поля «геодезична прив’язка» – як динамічний масив координат неправильного многокутника та кількість контрольних точок, що утворюють даний многокутник;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Земельна ділянка» визначити як клас, що містить:

- закриті поля типу «Власник» , «Опис» та «Призначення»;
- закрите поле дійсного типу зі значенням ринкової вартості ділянки;
- передбачити функцію виведення скороченої інформації з прізвищем власника та вартістю ділянки;
- інші функції реалізувати відповідно загальним вимогам.

Тип «Населений пункт» визначити як клас, що містить:

- статичне поле цілого типу з загальною кількістю населених пунктів;
- закрите поле цілого типу з порядковим номером населеного пункту; заповнюється автоматично у момент створення чергового об’єкту;
- інформацію про усі ділянки, що входять до складу даного населеного пункту, оформити як динамічний масив (вказівник і розмірність, поля закриті);
- передбачити функцію для зміни кількості земельних ділянок у складі даного населеного пункту;

- передбачити функцію з виведенням скороченої інформації з назвою класу та порядковим номером об’єкта;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти типу «Населений пункт» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. З одного населеного пункту видалити одну земельну ділянку, після чого вивести повну інформацію про цей пункт.

Варіант 4.

Тип «Тип приміщення» визначити як перерахування (enum) із значеннями полів «клітка», «вольєр», «акваріум», «тераріум».

Тип «Тварина» визначити як клас, що містить:

- закриті поля з назвою, країною походження та іменем, реалізовані у вигляді символьних рядків довільної довжини;
- закрите поле типу *Date* з датою народження тварини;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Одиниця обліку» визначити як клас, що містить:

- закриті поля типу «Тварина», типу *Date* з датою надходження та ціле з вартістю утримання тварини;
- функції реалізувати відповідно загальним вимогам.

Тип «Приміщення» визначити як клас, що містить:

- закрите поле типу «Тип приміщення»;
- закриті поля цілого типу з номером, розміром та вартістю прибирання приміщення;
- інформацію про тварин, що утримуються в даному приміщенні, оформити як динамічний масив типу «Одиниця обліку» (вказівник і розмірність, поля закриті);
- передбачити функцію для додавання чергової тварини в даному приміщенні;

- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації з номером та вартістю утримання тварин даного приміщення;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти типу «Приміщення» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. До одного з об’єктів додати одну тварину і вивести оновлену інформацію.

Варіант 5.

Тип «Форма проведення» визначити як перерахування (enum) зі значеннями полів «автобусна екскурсія», «прогулянка пішки», «кінна прогулянка», «рейсовим транспортом».

Тип «Організатор» визначити як клас, що містить:

- закриті поля з ім’ям та прізвищем, реалізовані у вигляді символьних рядків довільної довжини;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Екскурсія» визначити як клас, що містить:

- закриті поля типу «Організатор», «Форма проведення» та ціле з вартістю;
- закрите поле з місцем проведення екскурсії, реалізоване у вигляді символьного рядка;
- функції реалізувати відповідно загальним вимогам.

Тип «Тур» визначити як клас, що містить:

- закрите поле типу *Date* з датою початку;
- інформацію про екскурсії даного тура, оформити як динамічний масив (вказівник і розмірність, поля закриті);
- передбачити функцію для зміни кількості екскурсій у даному турі;
- крім стандартної функції виведення повної інформації, передбачити функцію виведення скороченої інформації з датою та вартістю тура;

- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти типу «Тур» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. До одного з об’єктів додати одну екскурсію і вивести оновлену інформацію.

Варіант6.

Тип «Періодичність» визначити як перерахування (enum) із значеннями полів «щотижня», «щомісяця», «щокварталу».

Тип «Автор» визначити як клас, що містить:

- закриті поля з ім’ям та прізвищем, реалізовані у вигляді символьних рядків довільної довжини та датою народження типу *Date*;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Стаття» визначити як клас, що містить:

- закриті поля типу «Автор», символьний рядок з назвою, цілі з кількістю сторінок та гонораром;
- функції реалізувати відповідно загальним вимогам.

Тип «Журнал» визначити як клас, що містить:

- закриті поля типу «Періодичність», «Автор» та символьний рядок з назвою;
- інформацію про включені до даного видання статті оформити як динамічний масив (вказівник і розмірність, поля закриті);
- передбачити функцію для внесення чергової статті до даного видання ;
- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації з назвою журналу та загальною кількістю сторінок;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти типу «Журнал» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. До одного з об’єктів додати одну статтю і вивести оновлену інформацію про дане видання в обох формах.

Варіант7.

Тип «Наукові досягнення» визначити як перерахування (enum) із значеннями полів «тези до доповіді», «стаття у фаховому виданні», «доповідь на міжнародній конференції», «стаття у міжнародному науковому журналі».

Тип «Студент» визначити як клас, що містить:

- закриті поля з ім’ям та прізвищем, реалізовані у вигляді символьних рядків довільної довжини, цілого типу з роком зарахування на навчання;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Замовник» визначити як клас, що містить:

- закриті поля з назвою організації та темою дослідження як символьні рядки довільної довжини, ціле поле з вартістю;
- функції реалізувати відповідно загальним вимогам.

Тип «Публікація» визначити як клас, що містить:

- закриті поля типу «Студент» та «Наукові досягнення»;
- функції реалізувати відповідно загальним вимогам.

Тип «Дослідження» визначити як клас, що містить:

- закриті поля типу «Замовник» та *Date* з датою підписання договору;
- інформацію про публікації оформити як динамічний масив (вказівник і розмірність, поля закриті);
- передбачити функцію для додавання чергової публікації;
- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації з темою дослідження та кількістю публікацій;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти типу «Дослідження» відповідно загальним вимогам, вивести інформацію про них у повному та скороченому вигляді. Додати публікацію і вивести оновлену інформацію.

Варіант 8.

Тип «Учасник змагань» визначити як клас, що містить:

- закриті поля з ім’ям та прізвищем, реалізовані у вигляді символьних рядків довільної довжини, з датою народження типу *Date*;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Виступ» визначити як клас, що містить:

- закрите поле логічного типу з типом змагань(командні, індивідуальні);
- статичне поле цілого типу з кількістю проведених виступів;
- закриті поля типу «Учасник змагань», цілі з порядковим номером поточного виступу (формується автоматично) та значенням результату;
- функції реалізувати відповідно загальним вимогам.

Тип «Змагання» визначити як клас, що містить:

- закрите поле з назвою змагання, реалізоване у вигляді символьного рядка довільної довжини;
- інформацію про проведені виступи оформити як динамічний масив (вказівник і розмірність, поля закриті);
- передбачити функцію для додавання чергового виступу до вже проведених;
- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації з назвою змагань та прізвищем переможця;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти типу «Змагання» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. До одного з об’єктів додати один виступ і вивести оновлену інформацію.

Варіант9.

Тип «Послуга» визначити як перерахування (enum) із значеннями полів «прибирання», «миття вікон», «догляд за дитиною», «дрібний ремонт», «змішана».

Тип «Виконавець» визначити як клас, що містить:

- закриті поля з ім’ям та прізвищем, реалізовані у вигляді символьних рядків довільної довжини, з датою народження типу *Date*;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Замовник» визначити як клас, що містить:

- закриті поля типу «Послуга» та з адресою замовника, реалізоване у вигляді символьного рядка довільної довжини;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Замовлення» визначити як клас, що містить:

- закриті поля типу «Виконавець», «Замовник» та *Date*;
- статичний масив вартостей послуг та закрите ціле поле з вартістю даної послуги (заповнюється автоматично);
- функції реалізувати відповідно загальним вимогам.

Тип «Бюро послуг» визначити як клас, що містить:

- закрите поле з назвою як символьний рядок фіксованої довжини;
- інформацію про усі виконані замовлення оформити як динамічний масив (вказівник і розмірність, поля закриті);
- передбачити функцію для внесення чергового замовлення до даного масиву;
- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації з назвою фірми та загальною вартістю замовлень;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти типу «Бюро послуг» відповідно загальним вимогам, вивести інформацію про них у повному та скороченому

вигляді. До одного з об’єктів додати замовлення і вивести оновлену інформацію.

Варіант10.

Тип «Робота» визначити як перерахування (enum) із значеннями полів «годування», «прибирання приміщення», «медогляд», «випас».

Тип «Тварина» визначити як клас, що містить:

- закриті поля з назвою та ім’ям тварини, реалізовані у вигляді символьних рядків довільної довжини, цілого типу з роком народження;
- закрите поле логічного типу зі статтю тварини;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Наряд» визначити як клас, що містить:

- закриті поля типу «Тварина», «Робота» та ціле з вартістю даної роботи;
- функції реалізувати відповідно загальним вимогам.

Тип «Догляд» визначити як клас, що містить:

- закриті поля з прізвищем власника, реалізоване у вигляді символьного рядка довільної довжини та з датою виконання робіт типу *Date*;
- інформацію про обслуговування тварин даного власника оформити як динамічний масив нарядів (вказівник і розмірність, поля закриті);
- передбачити функцію для додавання чергового наряду на обслуговування;
- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації з датою та сумарною вартістю робіт;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти типу «Догляд» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. До одного з об’єктів додати один наряд на обслуговування і вивести оновлену інформацію про даний робочий день в обох формах.

Варіант 11.

Тип «Твір» визначити як перерахування (enum) зі значеннями полів «інструментальний», «вокальний», «віршований», «прозовий».

Тип «Виконавець» визначити як клас, що містить:

- закриті поля з ім'ям та прізвищем, реалізовані у вигляді символьних рядків довільної довжини;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Виступ» визначити як клас, що містить:

- закриті поля типів «Виконавець»та «Твір», цілого типу з тривалістю виступу та символьний рядок з назвою твору;
- функції реалізувати відповідно загальним вимогам.

Тип «Концерт» визначити як клас, що містить:

- закрите поле з назвою фірми-організатора, реалізоване у вигляді символьного рядка довільної довжини, та з датою концерту типу *Date*;
- інформацію про включені до даного концерту виступи оформити як динамічний масив (вказівник і розмірність, поля закриті);
- передбачити функцію для внесення одного додаткового виступу;
- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації з назвою організатора та загальною тривалістю концерту;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об'єкти типу «Концерт» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. До одного з об'єктів додати один виступ і вивести оновлену інформацію.

Варіант12.

Тип «Розміщення» визначити як перерахування (enum) зі значеннями полів «стіна», «стіл», «підлога».

Тип «Фонди» визначити як клас, що містить:

- закриті поля з назвою та адресою, реалізовані у вигляді символьних рядків довільної довжини;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Витвір мистецтва» визначити як клас, що містить:

- закриті поля типу символьний рядок з назвою, цілого типу з роком створення та три дійсного типу з габарітними розмірами;
- функції реалізувати відповідно загальним вимогам.

Тип «Експонат» визначити як клас, що містить:

- закриті поля типів «Витвір мистецтва», «Фонди» та «Розміщення»;
- закрите поле цілого типу з вартістю експонату;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Виставковий зал» визначити як клас, що містить:

- закрите поле з назвою як символьний рядок;
- інформацію про наявні експонати оформити як динамічний масив (вказівник і розмірність, поля закриті);
- передбачити функцію для додавання чергового експонату до експозиції;
- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації про назву залу та часовий проміжок створення експонатів даної експозиції;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти типу «Виставковий зал» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. До одного з об’єктів додати один експонат і вивести оновлену інформацію про даний зал в обох формах.

Варіант 13.

Тип «Категорія» визначити як перерахування (enum) зі значеннями полів «холодні закуски», «перші страви», «другі страви», «десерти», «напої».

Тип «Повар» визначити як клас, що містить:

- закриті поля з ім’ям та прізвищем, реалізовані у вигляді символьних рядків довільної довжини;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Страва» визначити як клас, що містить:

- закриті поля типу символьного рядка з назвою, цілі з вартістю та тривалістю приготування, та типів «Категорія» і «Повар»;
- функції реалізувати відповідно загальним вимогам.

Тип «Замовлення» визначити як клас, що містить:

- закриті поля з назвою кафе та поточною датою (тип *Date*);
- інформацію про включені до даного замовлення страви оформити як динамічний масив (вказівник і розмірність, поля закриті);
- передбачити функцію для внесення до замовлення нової страви;
- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації з назвою кафе, датою та часом очікування даного замовлення;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти «Замовлення» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. До одного з об’єктів додати нову страву, вивести оновлену інформацію.

Варіант 14.

Тип «Клієнт» визначити як перерахування (enum) зі значеннями полів «чоловік», «жінка, фарбування», «жінка, стрижка», «жінка, укладка», «дитина».

Тип «Перукар» визначити як клас, що містить:

- закриті поля з ім’ям та прізвищем, реалізовані у вигляді символьних рядків довільної довжини;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Зачіска» визначити як клас, що містить:

- закрите поле з назвою як символьний рядок фіксованої довжини;

- закриті поля типів «Клієнт» та «Перукар», символьний рядок з назвою, цілого типу з вартістю та логічного «потреба у додаткових послугах»;
- функції реалізувати відповідно загальним вимогам.

Тип «Перукарня» визначити як клас, що містить:

- закриті поля з номером перукарні та типу *Date* з поточною датою;
- статичне поле з вартістю додаткових послуг;
- інформацію про виконані на даний момент зачіски оформити як динамічний масив (вказівник і розмірність, поля закриті);
- передбачити функцію для внесення нової зачіски до складу виконаних;
- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації з номером перукарні, датою та сумарною вартістю виконаних за день робіт;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти типу «Перукарня» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. До одного з об’єктів додати одну зачіску і вивести оновлену інформацію.

Варіант 15.

Тип «Категорія» визначити як перерахування (enum) зі значеннями полів «легковий сімейний», «спортивний», «кабріолет», «джип».

Тип «Автомобіль» визначити як клас, що містить:

- закриті поля з назвою фірми-виробника та маркою автомобіля, реалізовані у вигляді символьних рядків довільної довжини;
- закриті поля цілого типу з роком виробництва та вартістю;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Транспортний засіб» визначити як клас, що містить:

- закриті поля типів «Категорія» та «Автомобіль», типу *Date* з датою початку прокату, цілі поля з вартістю та тривалістю прокату, символьний рядок з номером автомобіля;

- функції реалізувати відповідно загальним вимогам.

Тип «Прокат» визначити як клас, що містить:

- закрите поле з назвою фірми з прокату транспортних засобів;
- інформацію про уже оформлені замовлення оформити як динамічний масив транспортних засобів (вказівник і розмірність, поля закриті);
- передбачити функції (або функцію) для зміни кількості внесених у масив транспортних засобів на один;
- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації з назвою фірми, датою та сумарною вартістю автомобілів, що знаходяться у прокаті;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти типу «Прокат» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. До одного з об’єктів додати транспортний засіб і вивести оновлену інформацію.

Варіант 16.

Тип «Категорія» визначити як перерахування (enum) зі значеннями полів «драма», «оперета», «опера», «балет».

Тип «Трупа» визначити як клас, що містить:

- закрите поле з назвою у вигляді символьного рядка довільної довжини;
- закриті поля цілого типу з кількістю акторів та сумою, що підлягає оплаті за роботу акторів;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Приміщення» визначити як клас, що містить:

- закриті поля з назвою та адресою у вигляді символьних рядків довільної довжини, цілого типу з сумою орендної плати та кількістю місць;
- статичне поле цілого типу з вартістю роботи оркестру;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Вистава» визначити як клас, що містить:

- закрите поле з назвою як символьний рядок;
- закриті поля типів «Категорія», «Приміщення» та «Трупа», типу *Date* з датою вистави, ціле з вартістю оренди, логічне «потреба в оркестрі»;
- функції реалізувати відповідно загальним вимогам.

Тип «Репертуар» визначити як клас, що містить:

- закрите поле з назвою місяця;
- інформацію про включені до репертуару вистави оформити як динамічний масив (вказівник і розмірність, поля закриті);
- передбачити функції (або функцію) для зміни кількості внесених до репертуару вистав на одну;
- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації з назвою класу, назвою місяця та сумарною кількістю вистав;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти типу «Репертуар» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. У одному з об’єктів видалити одну виставу і вивести оновлену інформацію.

Варіант 17.

Тип «Доставка» визначити як перерахування (enum) зі значеннями полів «посередник», «постачальник», «власними засобами».

Тип «Городина» визначити як клас, що містить:

- закриті поля з назвою та країною походження у вигляді символьних рядків довільної довжини, поле цілого типу з номером сезону визрівання;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Партія товару» визначити як клас, що містить:

- закриті поля типів «Городина» та «Доставка», цілого типу з кількістю, ціною одиниці та вартістю транспортування, типу *Date* з датою поставки;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Склад» визначити як клас, що містить:

- закриті поля цілого типу з номером та вартістю обслуговування приміщення;
- інформацію про прийняті на склад партії товару оформити як динамічний масив (вказівник і розмірність, поля закриті);
- передбачити функції (або функцію) прийому на склад та списання зі складу партії товару;
- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації з назвою класу та сумарною вартістю товару на складі;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти типу «Склад» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. До одного з об’єктів додати партію товару і вивести оновлену інформацію.

Варіант 18.

Тип «Варіант розпилювання» визначити як перерахування (enum) зі значеннями полів «брус», «дошка необрізана», «дошка обрізна», «рейка».

Тип «Деревина» визначити як клас, що містить:

- закриті поля з породою дерева у вигляді символьного рядка довільної довжини та цілого типу з вологістю та щільністю деревини;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Пиломатеріал» визначити як клас, що містить:

- закриті поля типів «Деревина» та «Варіант розпилювання»;
- закриті поля типу *Date* з датою поставки та цілого типу з маркуванням, кількістю та вартістю за одиницю;

- усі функції реалізувати відповідно загальним вимогам.

Тип «Майстерня» визначити як клас, що містить:

- статичне поле цілого типу, що містить інформацію про кількість ініціалізованих об’єктів типу «Майстерня»;
- закрите поле цілого типу з порядковим номером майстерні;
- інформацію про отримані пиломатеріали оформити як динамічний масив (вказівник і розмірність, поля закриті);
- передбачити функцію для зміни кількості отриманих до майстерні пиломатеріалів;
- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації з номером майстерні та сумарною вартістю отриманих пиломатеріалів;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти типу «Майстерня» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. До одного з об’єктів додати одиницю пиломатеріалу і вивести оновлену інформацію про цей об’єкт в обох формах.

Варіант 19.

Тип «Класифікація» визначити як перерахування (enum) зі значеннями полів «м’яка іграшка», «лялька», «модель техніки», «конструктор».

Тип «Іграшка» визначити як клас, що містить:

- закриті поля з назвами іграшки та фірми-виробника у вигляді символьних рядків довільної довжини, поле типу «Класифікація»;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Партія іграшок» визначити як клас, що містить:

- закриті поля типу «Іграшка», типу *Date* з датою поставки, цілого типу з ціною та кількістю екземплярів;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Магазин іграшок» визначити як клас, що містить:

- статичне поле цілого типу, що містить інформацію про кількість магазинів (об’єктів, що уже ініціалізовані);
- закрите поле цілого типу з порядковим номером магазину;
- інформацію про отримані партії іграшок оформити як динамічний масив (вказівник і розмірність, поля закриті);
- передбачити функцію (функції) для зміни кількості іграшок в магазині: додається партія іграшок, продається одна іграшка;
- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації з номером магазину та сумарною кількістю іграшок у магазині;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об’єкти типу «Магазин» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. Для одного з об’єктів додати партію іграшок, вилучити один екземпляр із цієї партії, вивести оновлену інформацію про даний магазин в обох формах.

Варіант 20.

Тип «Секція» визначити як перерахування (enum) зі значеннями полів «малювання», «танцювальний», «моделювання», «м’яка іграшка».

Тип «Керівник» визначити як клас, що містить:

- закриті поля з ім’ям та прізвищем, реалізовані у вигляді символьних рядків довільної довжини, з датою народження типу *Date*;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Гурток» визначити як клас, що містить:

- закрите поле з назвою у вигляді символьного рядка довільної довжини;
- закриті поля типів «Секція» та «Керівник», цілого типу з розміром оплати, кількістю занять на місяць та кількістю учнів;
- усі функції реалізувати відповідно загальним вимогам.

Тип «Будинок дитячої творчості» визначити як клас, що містить:

- закрите поле з адресою як символьний рядок;
- інформацію про діючі гуртки оформити як динамічний масив (вказівник і розмірність, поля закриті);
- передбачити функції (або функцію) для зміни кількості гуртків;
- крім стандартної функції виведення повної інформації, передбачити функцію з виведенням скороченої інформації з назвою класу, адресою та сумарною кількістю учнів, що відвідують гуртки будинку;
- інші функції реалізувати відповідно загальним вимогам.

У тестовому прикладі створити об'єкти типу «Будинок дитячої творчості» відповідно загальним вимогам і вивести інформацію про них у повному та скороченому вигляді. В одному із цих об'єктів видалити один гурток, додати інший і вивести оновлену інформацію про даний об'єкт в обох формах.

Контрольні запитання:

1. Що таке об'єктна декомпозиція? Дати визначення поняттям «повідомлення», «метод», «поведінка об'єкту», «передача повідомлення» о
2. Які механізми забезпечують захист об'єкта?
3. Які методи компілятор може надати класу за умовчанням?
4. Як відбувається створення, ініціалізація, знищення об'єктів?
5. У яких випадках відбувається виклик конструктора копій? Чим відрізняється поверхневе і глибоке копіювання об'єктів? Як працює конструктор копії за умовчанням?
6. Що таке функції-селектори та функції-модифікатори? Що таке «посилання на себе»?
7. Що таке і де використовуються inline-функції?
8. Пояснити призначення і використання статичних членів класу.

Література: [1, С. 55–76];[2, С. 270–297];[5, С. 96–100];[6, С. 16–18, 44– 56]; [7, С. 155–166]; [8, С. 301–332]; [15, С. 16–25]

КОМП'ЮТЕРНИЙ ПРАКТИКУМ № 2. РЕАЛІЗАЦІЯ УСПАДКУВАННЯ

(до теми «Поліморфізм та ієрархічність»)

Мета роботи: Навчитись правильно описувати ієрархії об'єктів та ієрархії класів мовою *C++*.

Основні теоретичні відомості

Ієрархічність та її різновиди. Для подолання складності системи будують як ієрархічні. Ієрархія – це впорядкування абстракцій, розташування їх по рівнях. Розрізняють два різновиди ієрархій: ієрархія класів або спадковість і ієрархія об'єктів або агрегація.

Спадковість – це механізм отримання нового класу із існуючих шляхом запозичення структурної або функціональної частини одного або декількох інших. Відповідно розрізняють одиночне та множинне успадкування. Спадковість створює таку ієрархію абстракцій, у якій підкласи (похідні класи, класи-нащадки) успадковують будову і поведінку одного або декількох суперкласів (базових класів, батьківських класів). Кажуть, що спадковість описується відношенням «is-a» і породжує ієрархію «узагальнення-спеціалізація».

Агрегація – це такий різновид ієрархії, який передбачає використання об'єктів одного класу в оголошенні іншого класу. Агрегацію (або включення) можна описати відношенням «part of» або «бути частиною». Клас, що містить поля – об'єкти інших класів, називається агрегатом або контейнером.

Одиночна спадковість. Реалізація в C++ . Ідея спадковості одна з основних в об'єктно-орієнтованому програмуванні. З погляду фізичної реалізації побудова нового класу на базі старого дає можливість повторного використання старого коду і додавання нових властивостей, зміни деяких

аспектів поведінки за рахунок заміни коду деяких методів (функцій) і прав доступу. Синтаксис оголошення наступний:

class Student

{char name, *group;..}*

class Starosta: public Student{int level; . . . }

Клас *Starosta* – похідний від класу *Student*, а *Student* – базовий для *Starosta.*

Клас *Starosta* крім своїх власних членів (*level*) містить і члени класу *Student.*

Виведення похідного класу із базового робить його підтипом базового. Тому вказівник на базовий клас можна використовувати як вказівник на похідний (але не навпаки). Наприклад:

*void f(Student * p, Starosta *q)*

*{Student *KA01 [10];*

KA01[0] = p;

KA02[1] = q;…}

Щоб клас можна було використовувати як базовий, його потрібно повністю визначити до визначення похідного, недостатньо тільки оголосити.

Для членів похідного класу доступні усі відкриті *(public)* та захищені *(protected)* члени базового класу напряму, як власні. Закриті поля *(private)* базового класу входять до складу похідного, але недоступні напряму – тільки через відкриті методи базового класу.

В С++ існує три різновиди успадкування: відкрите *(public)*, захищене *(protected)* та за умовчанням закрите *(private)*

За допомогою цього механізму похідний клас може змінити рівень доступу до успадкованої від базового класу частини своїх об’єктів.

Загальне правило: за допомогою похідного класу елементи базового класу не можна зробити більш відкритими (тільки більш закритими).

Конструктори похідних класів. В оголошенні любого конструктора похідного класу потрібно враховувати наступне.

- Конструювання об'єкту похідного класу обов'язково відбувається з викликом конструктора базового класу.
- Якщо явний виклик конструктора базового класу не передбачено у конструкторі похідного, за умовчання буде викликано конструктор без параметрів базового класу.
- Для організації виклику потрібної версії конструктора базового класу, можна скористатись списком ініціалізації, наприклад:

*Student :: Student (char*a, char* b): name (a), group (b) {…}*

*Starosta :: Starosta (char*a, char* b, int c): Student (a,b), level (c) {…}*

Об'єкти створюються згори вниз: спочатку базовий клас, тоді члени похідного класу, потім сам похідний клас. Знищуються – в протилежному порядку. Про це особливо важливо пам'ятати. коли в деструкторі необхідно явно звільняти ресурси.

Похідний клас може бути одночасно базовим для іншого класу. Такий набір зв'язаних класів називається ієрархією класів.

Переозначення функцій в похідних класах. Якщо в похідному класі створена функція з таким самим ім'ям, що і в базовому класі, то має місце заміщення методу. Кажуть, що в цьому випадку метод похідного класу приховує метод або методи (якщо їх декілька з однаковими іменами і різними сигнатурами) базового класу.

Множинне успадкування. Допустиме у *C++* множинне успадкування *(multiple inheritance)* дає можливість отримати похідний клас від декількох базових. Для опису ієрархії множинного успадкування можна використати орієнтований ациклічний граф.

Синтаксис заголовка класу розширюється, щоб можна було використати список базових класів з атрибутами. Наприклад:

class A: public B, public C{…}

Створення об’єкта похідного класу відбувається з викликом конструкторів базових класів у тій послідовності, в якій визначена спадковість, навіть якщо у списку ініціалізації вони описані у зворотному порядку. Для розглянутого приклада: *B, C, A.*

Порядок виконання роботи

Відповідно варіанту до тексту першої лабораторної роботи потрібно внести наступні зміни:

- визначити базовий та похідний класи відповідно таблиці 2.1.
- переписати оголошення похідного класу відповідно синтаксису для відкритого успадкування;
- переробити оголошення й реалізацію конструкторів похідного класу, у конструкторі з параметрами обов’язково передбачити використання конструктора з параметрами базового класу;
- у реалізації функції виведення на екран інформації в похідному класі використати відповідну функцію базового класу;
- до конструкторів та деструкторів базового, похідного та одного з агрегованих класів долучити виведення відповідної контрольної інформації для можливості відстеження порядку створення та знищення об’єктів;

У тестовому прикладі створити по одному об’єкту кожного з класів для контролю етапів утворення відповідних об’єктів й викликати функції виведення інформації. Пояснити отримані результати.

Таблиця 2.1. Визначення базових та похідних класів відповідно варіанту

Варіант	**Базові класи**	**Похідні класи**
1	«Людина»	«Студент»
2	«Датчик»	«Пристрій»
3	«Власник», «Опис»	«Земельна ділянка»
4	«Тварина»	«Одиниця обліку»
5	«Організатор»	«Екскурсія»

Варіант	Базові класи	Похідні класи
6	«Автор»	«Стаття»
7	«Замовник»	«Дослідження»
8	«Учасник»	«Змагання»
9	«Виконавець», «Замовник»	«Замовлення»
10	«Тварина»	«Наряд»
11	«Виконавець»	«Виступ»
12	«Фонд», «Витвір мистецтва»	«Експонат»
13	«Повар»	«Страва»
14	«Перукар»	«Зачіска»
15	«Автомобіль»	«Транспортний засіб»
16	«Трупа»	«Вистава»
17	«Городина»	«Партія товару»
18	«Деревина»	«Пиломатеріал»
19	«Іграшка»	«Партія іграшок»
20	«Керівник»	«Гурток»

Контрольні запитання:

1. Що таке успадкування? Описати структуру об’єкта похідного класу.
2. Описати правила доступу для різних типів успадкування.
3. Чим відрізняються два види ієрархії? Навести приклади із свого варіанта.
4. Як відбувається конструювання об’єктів при множинному успадкуванні? При ромбообразному успадкуванні?

Література: [1, С. 55–82];[2, С. 349–356];[5, С. 222–228];[[6, С. 16–18, 145–188]; [7, С. 281–285]; [8, С. 301–314];[9, 471–670];[15, С. 26–32]

КОМП'ЮТЕРНИЙ ПРАКТИКУМ № 3. ПЕРЕВАНТАЖЕННЯ ФУНКЦІЙ, ПЕРЕВАНТАЖЕННЯ ОПЕРАЦІЙ, ВИКОРИСТАННЯ МЕХАНІЗМУ ВІРТУАЛЬНИХ ФУНКЦІЙ

(до теми «Поліморфізм та ієрархічність»)

Мета роботи: Навчитись коректно користуватися перевантаженням функцій, перевантаженням операцій та механізмом віртуальних функцій в *C++*.

Основні теоретичні відомості

Перевантаження функцій. Перевантаження функцій означає, що для передачі повідомлень об'єктам різних класів можна скористатись функціями з однаковими іменами і що кожний об'єкт буде реагувати відповідним чином.

Така техніка використовується для базових операцій C++. Наприклад, існує одна назва і одна позначка для додавання: «+», але його можна використовувати для додавання цілих, дійсних, для інкременту вказівників. Ця ідея поширюється і на функції користувача. Фактично це різні функції і перевантаження імен має значення тільки для зручності запису.

Під час виклику функції *f* процес пошуку потрібної із множини перевантажених відбувається як пошук найбільшої відповідності типів формальних і фактичних параметрів.

Перевантаження операторів. Перевантаження операторів – можливість зміни семантичного навантаження стандартних операторів відповідно потребам нового класу або нових потреб існуючого класу.

В C++ допустимі перевантаження таких операторів:

+, -, /,*, %, ^, &, !, ~, =, <, >, +=, - =, * =, / =, % =, ^ =, & =, / =, <<, >>, ==, ! =, < =, > =, &&, //, ++, --, -> *, [], (), *new, new*[], *delete, delete*[].

Не можуть бути перевизначені оператори:

::, ., .*, :, *sizeof()* та непередбачені в синтаксисі мови.

Назва операторної функції починається з ключового слова ***operator***, за яким іде символ оператора, наприклад

*operator **

Операторна функція оголошується і може бути викликана як звичайна функція. Використання її як оператора – скорочена форма явного виклику. Наприклад в описі класу «комплексне число» оголошено оператор += :

```
class Complex
{   double re, im;
public:
complex (double r, double i): re(r), im(i){};
complex& operator += (complex&);
};
```

Тоді операторна функція може бути використана двома способами:

```
Complex a(1,2), b(4,-5);
a. operator += (b);          // явний виклик
a+=b;                        // скорочена форма
```

У перевантажених операторах зберігається пріоритет і порядок виконання у виразах, закріплені за даними операторами.

Унарний оператор можна визначити або у вигляді функції-члена без аргументів, або у вигляді функції-не-члена з одним аргументом.

Оператор індексації використовується для класів-контейнерів і надає можливість доступу до елемента колекції, яку містить такий клас, напряму, за індексом. Такий оператор має повертати посилання на тип елемента колекції, якщо його бажано використовувати і зліва і справа від оператора присвоєння.

Бінарний оператор можна визначити або у вигляді функції-члена з одним аргументом, або у вигляді функції-не-члена з двома аргументами.

Для ілюстрації сказаного, можна розглянути наступний приклад. Нехай в деякому класі *X* існує ціле поле *some*. Операцію «+=« можна визначити як суму полів *some* двох об'єктів *X*:

class X

{…friend X& operator += (X&, const X&);…};

X& operator += (X&a, const X&b) { a.some+=b.some; return a;}

Або з використанням функції-члена:

class X{. . . X& operator +=(const X&);. . . };

X& X :: operator += (const X&b){ some += b.some; return this;}*

Оператор присвоєння відноситься до бінарних операторів і в С++ може бути перевизначений як звичайний бінарний оператор, але тільки як член класу.

Дружні класи і дружні функції. Оскільки деякі операторні функції можуть бути зовнішніми, виникає проблема доступу до прихованих частин класу. В цьому випадку використовують так звані дружні функції.

Дружні функції повинні бути оголошені в середині опису класу, з яким вони дружні, за допомогою префікса *friend.* Наприклад:

class A{int q; . . . friend void func (A, int);. . .};

void func (A a1, int i) {a1. q =i;};

Якщо усі функції-члени одного класу (*X*) є дружніми для другого класу (*Y*), говорять, що клас *X* – дружній класу *Y*. Тоді цей факт оголошують так:

class Y{. . . friend class X;. . . };

Перетворення типів. У арифметичних виразах можуть бути присутні об'єкти різних типів, тому виникає проблеми перетворення типів.

Для перетворення визначеного раніше типу до типу, який визначає користувач, можна скористатись конструктором з одним параметром (конструктор перетворення). Наприклад:

complex (double r) {re = r; im = 0;}

Для перетворення у зворотному порядку, використовують спеціальну нестатичну функцію-член класу *operator тип () {…};*

Наприклад:

*complex :: operator double () {return (sqrt(re*re+im*im));};*

Перевантаження операторів уведення – виведення. Оскільки існує можливість створення типів даних користувача, логічно мати можливість форматного уведення і виведення значень такого типу, скориставшись формою запису відповідних стандартних операторів. Це бінарні операції «<<« – «помістити в потік» і «>>« – «взяти із потоку». Вони можуть бути визначені тільки як зовнішні функції.

Операторну функцію «<<« потрібно перевизначити так, щоб вона мала два аргументи: типу *ostream &* і користувацького типу і повертала *ostream &* (посилання на потік використовується, щоб не копіювати об'єкт потоку).

Для розглянутого раніше типу *complex:*

class complex {. . . friend ostream & operator << (ostream & , complex x). . .};

ostream & operator << (ostream & o, complex x)

{ return o <<x.re << '+' <<' i' <<x. im; };

Аналогічно можна перевизначити оператор уведення з потоку.

У деяких випадках для перевантаження цих операторів зручно користуватись функціями-членами класів *ostream* та *istream* такими як *put, get, getline, write, read* та *ignore*.

Пізнє зв'язування. Якщо у класах, зв'язаних відносинами успадкування, існують функції з однаковими сигнатурами, це означає, що об'єктам цих класів можуть бути передані однакові повідомлення. Може виникнути ситуація, коли на етапі компіляції відомо, яке повідомлення потрібно передати, але тільки під час виконання стане відомо об'єкту якого класу. У цьому випадку використовують механізм пізнього зв'язування.

В С++ пізнє зв’язування реалізоване за допомогою функцій-членів, які називаються віртуальними функціями.

Віртуальну функцію оголошують у базовому класі за допомогою префікса *virtual*, а потім переозначають у похідних класах.

Віртуальні функції покладаються на додаткову структуру даних, яка підтримує зв’язок між різними версіями функцій. Це таблиця віртуальних функцій (*virtual method table-**vtbl***) – таблиця вказівників на віртуальні функції, яка конструюється для кожного класу окремо. Всі екземпляри класу містять вказівник на цю таблицю.

Такий механізм дозволяє асоціювати і зв’язати повідомлення з методом під час виконання програми автоматично. Програміст тільки визначає дії, які повинен виконати об’єкт, отримавши повідомлення.

Щоб досягти пізнього зв’язування для об’єкта, необхідно скористатись вказівником або посиланням на нього. Для відкритих похідних класів вказівники і посилання на об’єкти цих класів сумісні з вказівниками і посиланнями на об’єкти базового класу. Обрана функція-член залежить від класу, на об’єкт якого вказана, а не від типу вказівника.

Наприклад:

```
class Animal{      int age;
public:  . . .  virtual void Ask();   };
class Dog: public Animal{    char*  breed;
public: . . . void Ask();   };
class Cat: public Animal{      char* color;
public: . . . void Ask();    };
void Animal:: Ask(){cout<<«I’m animal ! «;};
void Dog:: Ask(){cout<<«I’m “<<breed <<’!’;};
void Cat:: Ask(){cout<<«I’m “<<color<<«  cat! «;};
void main()
{Animal *a[3];
```

Animal a; Cat c(“white»); Dog d(“Doberman»);

a[0] = &a; a[1] = &c; a[2] = &d;

for (int i; i=0; i<3) a[i]->Ask(); };

Результат:

I’m animal! I’m white cat! I’m Doberman!

Якщо в класі присутня хоча б одна віртуальна функція, його деструктор потрібно визначити як віртуальний.

Порядок виконання роботи

Відповідно варіанту до тексту другої лабораторної роботи потрібно внести наступні зміни:

- перевантажити указані арифметичні та логічні оператори, оператор індексації та оператори форматного уведення-виведення для одного з класів відповідно варіанту;
- перевірку коректності за бажанням можна реалізувати без генерування виключних ситуацій;
- визначити оператор присвоювання для класів, для яких це доцільно;
- функцію, що виводить деяку скорочену інформацію про об’єкт, визначити як віртуальну.

Реалізувати тестовий приклад, у якому передбачити:

- демонстрацію роботи кожного з перевантажених операторів;
- демонстрацію роботи кожної з віртуальних функцій так, щоб був задіяний віртуальний механізм.

Варіант 1.Визначити:

- арифметичний оператор «+» для класу *Date*;

- логічні оператори «<»та «>» для визначення кращого за загальним рейтингом студента – для класу «Студент»; порівняння коректне для об’єктів одного освітнього рівня;
- логічні оператори «==» та «!=» для перевірки збігу двох об’єктів класу «Людина»;
- оператор індексації для доступу до інформації про іспит – для класу «Студент»;
- оператори форматного уведення-виведення – для класів *Date* та «Людина».

Варіант 2.Визначити:

- арифметичний оператор «-» для класу *Date;*
- логічні оператори «<» та «>» для порівняння об’єктів класу «Пристрій» за поточними значеннями фізичної величини (коректне для датчиків одного типу);
- логічні оператори «==»та «!=» для порівняння двох об’єктів класу «Пристрій» за збіжністю значень усіх полів;
- оператор індексації для доступу до інформації про пристрій – для класу «Вимірювальний канал»;
- оператори форматного уведення-виведення – для класів «Датчик» та «Пристрій».

Варіант 3.Визначити:

- арифметичний оператор «-» для класу *Date;*
- логічні оператори «<» та «>» для порівняння об’єктів класу «Земельна ділянка» за значеннями їх вартості;
- логічні оператори «==» та «!=» для перевірки збігу двох об’єктів класу «Власник» за значеннями усіх полів;

- оператор індексації для доступу до інформації про земельну ділянку – для класу «Населений пункт»;
- оператори форматного уведення-виведення – для класів *Date* та «Власник».

Варіант 4.Визначити:

- арифметичний оператор «-» для класу *Date*;
- логічні оператори «<» та «>» для порівняння об’єктів класу «Одиниця обліку» за значеннями вартості утримання;
- логічні оператори «==» та «!=» для перевірки збігу двох об’єктів класу «Тварина» за значеннями усіх полів;
- оператор індексації для доступу до інформації про тварину як одиницю обліку – для класу «Приміщення»;
- оператори форматного уведення-виведення – для класів *Date*, «Тварина» та «Одиниця обліку».

Варіант 5.Визначити:

- арифметичний оператор «+» для класу «Екскурсія» за значенням вартості за умови збігу місця і форми проведення;
- логічні оператори «<» та «>» для порівняння об’єктів класу «Екскурсія» за значеннями вартості проведення;
- логічні оператори «==» та «!=» для перевірки збігу двох об’єктів класу «Організатор» за значеннями усіх полів;
- оператор індексації для доступу до інформації про окрему екскурсію – для класу «Тур»;
- оператори форматного уведення-виведення – для класів «Організатор» та «Екскурсія».

Варіант 6. Визначити:

- арифметичний оператор «-» для класу *Date*;
- логічні оператори «<» та «>» для визначення більшої за обсягом статті – для класу «Стаття»;

- логічні оператори «==» та «!=» для перевірки збігу двох об’єктів класу «Автор»;
- оператор індексації для доступу до інформації про статтю – для класу «Журнал»;
- оператори форматного уведення-виведення – для класів *Date* та «Автор».

Варіант 7. Визначити:

- арифметичний оператор «+» для класу *Date*;
- логічні оператори «<»та «>» за вартістю робіт – для класу «Дослідження»;
- оператор індексації для доступу до інформації про публікацію – для класу «Дослідження»;
- логічні оператори «==»та «!=» для перевірки збігу двох об’єктів класу «Студент»;
- оператори форматного уведення-виведення – для класів «Студент» та «Публікація».

Варіант 8. Визначити:

- арифметичний оператор «+» для класу *Date*;
- логічні оператори «<»та «>» за результатом – для класу «Виступ»;
- оператор індексації для доступу до інформації про виступ – для класу «Змагання»;
- логічні оператори «==»та «!=» для перевірки збігу двох об’єктів класу «Учасник змагання»;
- оператори форматного уведення-виведення – для класів «Учасник» та «Виступ».

Варіант 9. Визначити:

- арифметичний оператор «+» для класу «Замовлення» як додавання за вартістю; коректний за умови збігу виконавців, значення послуги може змінитись на «змішана»;
- логічні оператори «<»та «>» за вартістю – для класу «Замовлення»;
- оператор індексації для доступу до інформації про замовлення – для класу «Бюро послуг»;
- логічні оператори «==»та «!=» для перевірки збігу двох об'єктів класу «Виконавець»;
- оператори форматного уведення-виведення – для класів «Замовник» та *Date*.

Варіант 10. Визначити:

- арифметичний оператор «+» для класу *Date*;
- логічні оператори «<»та «>» для класу «Догляд» за вартістю;
- логічні оператори «==»та «!=» для перевірки збігу двох об'єктів класу «Тварина»;
- оператор індексації для доступу до інформації про наряд – для класу «Догляд»;
- оператори форматного уведення-виведення – для класів *Date* та «Тварина».

Варіант 11. Визначити:

- арифметичний оператор «+» для класу «Виступ» за тривалістю, коректно за умову одного виконавця; назва твору складатиметься з назв двох творів (конкатенація);
- логічні оператори «<»та «>» для класу «Виступ» за тривалістю;

- логічні оператори «==»та «!=» для перевірки збігу двох об’єктів класу «Виконавець»;
- оператор індексації для доступу до інформації про виступ – для класу «Концерт»;
- оператори форматного уведення-виведення – для класів *Date* та «Виконавець».

Варіант 12. Визначити:

- арифметичний оператор «+» для класу *Date*;
- логічні оператори «<»та «>» для класу «Експонат» за вартістю;
- логічні оператори «==»та «!=» для перевірки збігу двох об’єктів класу «Витвір мистецтва»;
- оператор індексації для доступу до інформації про експонат – для класу «Виставковий зал»;
- оператори форматного уведення-виведення – для класів «Фонд» та «Витвір мистецтв».

Варіант 13. Визначити:

- арифметичний оператор «+» для класу *Date*;
- логічні оператори «<»та «>» для класу «Страва» за вартістю;
- логічні оператори «==»та «!=» для перевірки збігу двох об’єктів класу «Повар»;
- оператор індексації для доступу до інформації про страву – для класу «Замовлення»;
- оператори форматного уведення-виведення – для класів «Страва» та «Повар».

Варіант 14. Визначити:

- арифметичний оператор «+» для класу *Date*;
- логічні оператори «<»та «>» для класу «Зачіска» за вартістю;

- логічні оператори «==»та «!=» для перевірки збігу двох об’єктів класу «Перукар»;
- оператор індексації для доступу до інформації про зачіску – для класу «Перукарня»;
- оператори форматного уведення-виведення – для класів «Перукар» та «Зачіска».

Варіант 15. Визначити:

- арифметичний оператор «+» для класу *Date*;
- логічні оператори «<»та «>» для класу «Транспортний засіб» за вартістю прокату;
- логічні оператори «==»та «!=» для перевірки збігу двох об’єктів класу «Автомобіль»;
- оператор індексації для доступу до інформації про транспортний засіб – для класу «Прокат»;
- оператори форматного уведення-виведення – для класів «Автомобіль» та «Транспортний засіб».

Варіант 16. Визначити:

- арифметичний оператор «+» для класу *Date*;
- логічні оператори «<»та «>» для класу «Приміщення « за кількістю місць глядачів;
- логічні оператори «==»та «!=» для перевірки збігу двох об’єктів класу «Приміщення»;
- оператор індексації для доступу до інформації про виставу – для класу «Репертуар»;
- оператори форматного уведення-виведення – для класів «Приміщення» та *Date*.

Варіант 17. Визначити:

- арифметичний оператор «+» для класу «Партія товару» за умови збігу полів типу «Городина»; реалізувати як об'єднання двох партій;
- логічні оператори «<»та «>» для класу «Партія товару» за вартістю;
- логічні оператори «==»та «!=» для перевірки збігу двох об'єктів класу «Городина»;
- оператор індексації для доступу до інформації про партію товару – для класу «Склад»;
- оператори форматного уведення-виведення – для класів «Городина» та *Date*.

Варіант 18. Визначити:

- арифметичний оператор «+» для класу «Пиломатеріал» за умови збігу полів типу «Деревина»; реалізувати як об'єднання;
- логічні оператори «<»та «>» для класу «Пиломатеріал» за вартістю;
- логічні оператори «==»та «!=» для перевірки збігу двох об'єктів класу «Деревина»;
- оператор індексації для доступу до інформації про пиломатеріал – для класу «Майстерня»;
- оператори форматного уведення-виведення – для класів «Деревина» та «Пиломатеріал».

Варіант 19. Визначити:

- арифметичний оператор «+» для класу «Партія іграшок» для однакових іграшок;
- логічні оператори «<»та «>» для класу «Партія іграшок» за кількістю;
- логічні оператори «==»та «!=» для перевірки збігу двох об'єктів класу «Іграшка»;
- оператор індексації для доступу до інформації про партію іграшок – для класу «Магазин іграшок»;

- оператори форматного уведення-виведення – для класів «Іграшка» та «Партія іграшок» .

Варіант 20. Визначити:

- арифметичний оператор «+» для класу *Date*;
- логічні оператори «<»та «>» для класу «Гурток» за кількістю;
- логічні оператори «==»та «!=» для перевірки збігу двох об’єктів класу «Керівник»;
- оператор індексації для доступу до інформації про гурток – для класу «Будинок дитячої творчості»;
- оператори форматного уведення-виведення – для класів «Керівник» та «Гурток».

Контрольні запитання:

1. Що таке перевантаження функцій? Операцій?
2. Навести принцип визначення кількості параметрів операторної функції для перевантаження унарних та бінарних операторів.
3. Як повинен бути організований оператор?
4. Як реалізоване перевантаження операцій індексації, потокового уведення-виведення, логічних операцій, перетворення типів?
5. Наведіть порівняльну характеристику віртуальних та звичайних функцій.
6. Поясніть призначення дружніх класів та дружніх функції.

Література: [2, С. 309–320, 326 – 329, 358–363]; [6, С. 76–100, 122–140, 191–217]; [7, С. 192–210, 289–297];[8, С. 225–330, 380–400]; [9, С. 471–670]; [15, С. 32–70]

КОМП'ЮТЕРНИЙ ПРАКТИКУМ № 4. ВИКОРИСТАННЯ МОЖЛИВОСТЕЙ ПАРАМЕТРИЧНОГО ПОЛІМОРФІЗМУ, МЕХАНІЗМУ ВИКЛЮЧНИХ СИТУАЦІЙ ТА ЗАСОБІВ СТАНДАРТНОЇ БІБЛІОТЕКИ ШАБЛОНІВ

(до тем: «Параметричний поліморфізм», «Опрацювання виняткових ситуацій», «Стандартні бібліотеки»)

Мета роботи: Навчитись коректно користуватися параметризованими класами, параметризованими функціями та механізмом виключних ситуацій. Ознайомитись з основними засобами бібліотеки STL і вміти їх правильно застосовувати.

Основні теоретичні відомості

Абстракцію можна значно вдосконалити, якщо скористатись конструкцією параметризованих класів.

Клас, тип даних в якому заданий як параметр, називається параметризованим або шаблонним (шаблоном). Процес створення конкретного класу (екземпляра шаблону) на основі параметризованого класу шляхом підстановки замість формального параметра-типу фактичного типу – називається інстанціюванням (екземплярізацією).

Перед створенням об'єктів параметризований клас має бути інстанційованим.

Опис шаблону класу в C++. Опис шаблону починається ключовим словом *template*, наприклад

```
template <class T>  class X
{  T  var;  . . .
public:. . .
T  func1();
void func2(T val){ var=val;};    }
template <class T> T X<T>::  func1(){. . . return var;};
```

T – формальний тип, на місці якого після інстанціювання буде використано фактичний тип.

Функції-члени, описані за межами класу, мають бути описані як шаблони функцій.

Функції-члени, описані в середині представлення шаблона, вважаються *inline*, як і в звичайному класі.

Інстанціювання. Інстанціюванням шаблону *(template instantiation)* називається процес генерації оголошення класу за шаблоном класу та його аргументом. Аналогічно функція генерується (інстанціюється) із шаблона функції і аргументу шаблона. Версія шаблона для конкретного аргументу називається спеціалізацією. Наприклад: *X <char> obj(256);*

Процес інстанціювання відбувається під час компіляції. Згенеровані класи і функції – це звичайні класи і функції і підпорядковані усім правилам, які існують для звичайних класів і функцій.

Шаблони функцій. Синтаксис опису шаблонної функції:

template <class T> тип Function (список параметрів)

Такі функції можуть приймати як параметри і звичайні об’єкти, і об’єкти, задані в параметризованій формі.

Для типу, що використовується як фактичний параметр-тип, повинні бути допустимі використані в шаблоні операції.

Існує можливість виведення (*deduction*) типу аргументів шаблона функції за типами аргументів під час її виклику.

В шаблоні функції, як і в шаблоні класу, може бути декілька аргументів. Аргументи можна використати для вибору алгоритму.

Параметризовані класи найчастіше використовуються для представлення так званих контейнерних класів. Клас, що містить набір елементів деякого типу, називається класом-контейнером або просто контейнером. Оскільки кількість різновидів таких узагальнених контейнерів, які найчастіше

використовуються на практиці обмежена, всі вони зібрані в стандартній бібліотеці шаблонів *STL (Standard Template Library).*

Організація стандартної бібліотеки. Стандартна бібліотека визначена в просторі імен *std* і кожний засіб стандартної бібліотеки стає доступним через відповідний стандартний заголовковий файл, наприклад:

 # include <iostream>

Щоб усі імена із *std* оголосити глобальними потрібно скористатись *using*-оголошенням *using namespace std;*

До складу стандартної бібліотеки входять такі основні засоби:

контейнери *(<bitset>, <set>, <map>, <stack>, <vector>, <list>, <deque>, <queue>. . .);*

ітератори *(<iterator>. . .);*

алгоритми *(<algorithm>, <cstdlib>,<numeric>. . .);*

основні утіліти *(<utility>, <memory>, <functional>, <ctime>);*

діагностика *(<stdexcept>, <cassert>, <cerino>. . .);*

рядки *(<string>, <stdlib>,. . .);*

уведення-виведення *(<iostream>, <ios>, <istream>, <ostream>);*

локалізація *(<local>, . . .);*

числа*(<complex>, <valarray>, . . .).*

Підхід, прийнятий в стандартній бібліотеці C++ базується на таких поняттях, як послідовність, ітератор і алгоритм. Контейнер розглядається як послідовність елементів, до кожного з яких можна звертатись за допомогою ітератора.

Ітератор має багато спільних властивостей з вказівником: він посилається на поточний елемент послідовності і надає операцію, що примушує його посилатися на наступний. Для нього обов’язково визначені дві основні операції: «отримати доступ до елемента через ітератор» («*» – розіменування) і «примусити ітератор посилатись на наступний елемент» («++» – інкремент).

Початок і кінець послідовності визначається функціями *begin()* та *end()*. Початок – це ітератор, що вказує на перший елемент послідовності, кінець – ітератор, який посилається на наступний за останнім елемент.

Вбудовані масиви також можна вважати контейнерами, для них ітераторами слугуватимуть звичайні вказівники.

Стандартні контейнери поділяються на дві основні групи: послідовні та асоціативні.

Послідовні впорядковані порядком розташування елементів. До них відносяться базові контейнери вектор *(vector<T>)*, список *(list<T>)*, двобічна черга *(deque<T>)* та адаптери базових контейнерів: черга *(queue<T>)*, стек *(stack <T>)*, черга з пріоритетом *(priority_queue<T>)*.

Асоціативні контейнери містять ключі для пошуку елементів. Для них обов'язково визначена операція порівняння для елементів, що зберігаються. Це множини *(set<T>)*, мультімножини*(multiset<T>)*, відображення*(map <T>)* та мультівідображення*(multimap <T>)*.

Усі стандартні контейнери визначають методи, які маніпулюють контейнерами та їх елементами. Крім того, бібліотека надає стандартні алгоритми, які можна використовувати узагальнено з різними контейнерами. Основні визначені в *<algorithm>* .

Умовно ці алгоритми можна підрозділити на такі категорії:

- алгоритми сортування;
- алгоритми, які не модифікують послідовність;
- алгоритми, які модифікують послідовність;
- числові алгоритми (заголовковий файл <*numeric*>).

Об'єкт-функція. Часто в алгоритми потрібно передавати як параметри невеликі функції. Оскільки такі функції застосовують до кожного елементу контейнера, це значно знижує ефективність програми. Тому вказівник на функцію бажано замінити об'єктом деякого класу з перевантаженим оператором *()*. Якщо *operator()* оголошений як вбудований, він перетвориться

на *inline*-функцію, що за великої кількості звертань дасть значну перевагу у часі. Такі об’єкти називаються об’єктами-функціями. Бібліотека STL надає певний перелік стандартних об’єктів-функцій, можна розробити свої.

Приклад використання можливостей STL. Відсортувати за зменшенням елементи послідовності з використанням функції порівняння

```
#include "stdafx.h"
#include <iostream>
#include <vector>
#include <functional>
#include <algorithm>
using namespace std;
    // унарна функція для виведення елементів
template<class T>
class Print: public unary_function<T, void>
{   public:
void operator()(T& arg1)
{   cout<< arg1<<" ";   }   };
    // функція порівняння (оскільки стандартна функція сортування забезпечує сортування за збільшенням, у даному випадку необхідно визначити об’єкт порівняння)
bool ReverseCompare(int arg1, int arg2);
      // відобразити всі елементи в контейнері
template<class Container>
void ShowElements(Container& c, char* text);
int main(int argc, char* argv[])
{
typedef vector<int> VectorInt;
typedef VectorInt::iterator Itor;
      // створити вектор цілих
```

```
VectorInt vInt1(7);
Itor first1 = vInt1.begin();
Itor last1  = vInt1.end();
generate(first1, last1, rand);
ShowElements(vInt1, "Random number sequence");
        //  відсортувати послідовність за зменшенням
sort(first1, last1, ReverseCompare);
ShowElements(vInt1, "Sorted in descending order");
return 0;
}
        // функція порівняння
bool ReverseCompare(int arg1, int arg2)
{   return (arg1 > arg2);     }
        //  відобразити усі елементи контейнера
template<class Container>
void ShowElements(Container& c, char* text)
{ Print<Container::value_type> DoPrint;
cout<<text<<":\n";
for_each(c.begin(), c.end(), DoPrint);
cout<<«\n\n»;}
```

Обробка виняткових ситуацій. За умови існування декількох модулів, обробка помилок поділена на два етапи:

- генерування інформації про виникнення помилкової ситуації, яка не може бути розв’язана локально (генерування виключення);
- опрацювання помилок, виявлених в інших місцях (перехоплення і опрацювання виключення).

Виключеннями називають нештатні ситуації, які переривають програму користувача з повідомленням про помилку, яке видає система. Механізм

перехоплення виключних ситуацій реалізовано за допомогою трьох операторів: *try, catch* і *throw.*

- в блок *try* вставляється програмний код, у якому можуть виникнути помилки;
- в блоці *catch,* що розташований безпосередньо після зв'язного з ним блока try, знаходиться програмний код, який опрацьовує інформацію про виявлену помилку.
- оператор *throw* перериває виконання програми і передає керування зв'язаному з ним блоку *catch* або стандартному обробнику *terminate()*

```
try { …
if (…) throw (…); . . .…};
catch (…){…};
```

В операторі *if* відбувається перевірка умови виклику виключення. Якщо вона виконується, оператор *throw* викликає відповідне виключення і передає керування блоку *catch*, в якому воно і опрацьовується. Якщо умова не виконується, програма дійде кінця блоку *try* і розташований за ним *catch* не буде виконаний.

У разі генерування виключення, управління передається тому блоку *catch,* сигнатура якого відповідає сигнатурі *throw,* який згенерував дане виключення.

Приклад програми з виключеннями

```
#include "stdafx.h"
#include<iostream.h> . . .
Int_array:: Int_array(){ size=10;    p=new int[size+1];
for(i=0;i<size;i++)x[i]=0;};
Int_array:: Int_array(int nn): size(nn) { if(size<1) throw(size); p=new int[size+1];
        if(x==0) throw("size of memory"); for(int i=0;i<size;i++) p[i]=i;};
void g(int m){  Int_array qq(m); . . . };
int main(int argc, char* argv[])
```

```
{     . . .try{          g(nn);     }
      catch(int nn){     cerr<< " size error " << nn << endl; g(10);    }
      catch(const char *error){    cerr << error << endl;    }     return 0;}
```

Порядок виконання роботи

Ознайомитись з основними можливостями, що надає бібліотека STL.

Відповідно варіанту визначити необхідні стандартні контейнери *STL,* за потреби побудувати свій з поведінкою, ідентичною стандартному.

Розв’язати задачу обов’язково з використанням стандартних алгоритмів (не менше двох) і операцій контейнерних класів *STL.* За умови доцільності, побудувати свій об’єкт-функцію.

Ознайомитись з механізмом опрацювання виключних ситуацій.

У розробленій програмі передбачити обробку виключних ситуацій, означених в завданні або запропонувати свої, якщо такі у даному варіанті не конкретизовані.

Самостійно продумати і реалізувати спосіб демонстрації отриманих результатів.

Варіант 1

Текст подано у вигляді вектора, елементи якого – окремі рядки тексту. Відомо, що текст складається з п’яти частин, кожна з яких починається словом «Розділ», розташованим в окремому рядку.

Перетворити вказаний текст на стільки окремих списків, скільки розділів в ньому. Побудувати функції:

- порівняння двох списків;
- вклеювання частини одного списку (починаючи з номера n до кінця) – в другий – перед i-м елементом (з вилученням із першого). Передбачити виключні ситуації: i, n, $n+k$ – за межами списку.;
- визначення відсортованості елементів даних списків і злиття їх в один за умови відсортованості.

Варіант 2

Текст подано у вигляді вектора. Відсортувати елементи цього вектора і перетворити на два списки так, щоб у першому були тільки ті елементи, значення яких не повторюється, а у другому – тільки ті, що мають дублікати, наприклад, вектор з елементами *a, b, c, b, d, e, d, d* перетвориться на списки *a, c, e* і *b, d* відповідно.

Побудувати функції:

- визначення довжини списку;
- реверсування списку;
- виведення на екран *i*-го елемента, виняткова ситуація – вихід за границі списку.

Варіант 3

Колода із 36 карт (масть, ранг) організована як вектор. Зарезервувати в ньому місце ще для 16 карт додатково. «Перетасувати» елементи вектора і перетворити його у стек. Побудувати функції, необхідні для імітації гри:

- вибрати 6 карт зверху, утворивши список з відібраних карт;
- відсортувати елементи списку, передбачити власний оператор порівняння (з урахуванням масті) ;
- вставити елемент у відсортований список, не порушуючи відсортованості. Використати власну функцію порівняння; виключна ситуація – список невідсортований.

Варіант 4

Задано вектор комплексних чисел із *n* елементів. Утворити два списки: із перших *m* елементів і із останніх *n-m-1* (останній елемент залишити).

Відсортувати списки за збільшенням модуля.

Злити два списки в один, передбачити як виключну ситуацію відсутність відсортованості одного із списків.

Вставити до новоутвореного списку останній елемент із вектора так, щоб не порушити відсортованості його елементів.

Варіант 5

Задано два списки: перший включає співробітників усієї організації, згрупованих по відділам; другий – із співробітників тільки деякого відділу *A*.

Знайти перше входження списку співробітників відділу *A* в загальний список і циклічно пересунути елементи першого списку так, щоб він починався із списку співробітників відділу *A*.

Передбачити як виключну ситуацію відсутність входження другого списку у перший.

Відсортувати ту частину загального списку, яка включає тільки співробітників відділу *A*.

Варіант 6

Масив складається з двадцяти восьми об'єктів «доміно». Масив упорядкований – критерій визначити самостійно. Перетворити цей масив у список так, щоб елементи були «перетасовані». Створити функції, які були б необхідні для моделювання гри на два гравці в доміно. Всі ігрові набори кісток (окремих гравців, «базар», «черга») реалізувати як списки.

Функції:

- вибір необхідного елемента із списку гравця для доповнення черги з одної сторони;
- передбачити виключні ситуації:

відсутність потрібного елемента, обробник має доповнити список даного гравця елементами із списку «базар»;

«риба» – коли черга з обох сторін більше не може бути доповнена тому, що всі кістки з такими значеннями, як на її закінченнях, уже знаходяться в черзі;

- виведення на екран поточної ігрової ситуації.

Варіант 7

Імітація черги на отримання житла. Клас «учасник черги» – прізвище, телефон, дата постановки в чергу. Неупорядкований масив «учасників черги» перетворити в стандартний контейнер *priority queue*. Побудувати функції:

- визначення довжини черги;
- додавання нового елемента в чергу (постановлення на облік нового учасника);
- вилучення учасників, що отримали житло (по черзі).

Передбачити виключні ситуації: дата постановки на облік більша за поточну; дата постановлення на облік не дорівнює поточній для нових учасників.

Варіант 8

Побудувати свій контейнер *uses_priority_queue*, який мав би поведінку, еквівалентну стандартному *priority_queue* із *STL* щодо компонування елементів. Передбачити функцій, аналогічні функціям варіанта 7.

Варіант 9

Задано список учасників черги на друк видань: автор, назва, об’єм. Перетворити на стандартну приоритетну чергу відповідно до положення у списку. Побудувати шаблон функції для виведення на екран повної інформації про всіх учасників черги з використанням об’єкту-функції для виведення одного елемента.

Варіант 10

Задано масив, елементи якого містять інформацію про підручники в бібліотеці: код УДК, автор, назва, ціна, кількість примірників. Перетворити цей масив на дек. Передбачити функції:

- відповідно до курсу перевести ціну всіх книжок з гривень у євро (виключна ситуація: ціна = 0);

- підрахувати кількість книжок одного автора у каталозі; автор передається як параметр (виключна ситуація – нульове поле «кількість примірників»);
- видалити всю інформацію про книжки певного автора.

Варіант 11

Реалізувати шаблон контейнерного класу *deque*, який би характеризувався поведінкою, еквівалентному стандартному контейнеру *STL*. Перевірити для об'єктів типу «раціональний дріб».

Варіант 12

Реалізувати шаблон контейнерного класу *stack*, який би характеризувався поведінкою, еквівалентному стандартному контейнеру *STL*. Перевірити для об'єктів типу «гральна карта».

Варіант 13

Реалізувати шаблон контейнерного класу *queue*, який би характеризувався поведінкою, еквівалентному стандартному контейнеру *STL*. Перевірити для об'єктів типу «студент».

Варіант 14

Реалізувати шаблон контейнерного класу «кільце» на базі одного із стандартних послідовних контейнерів *STL*, який характеризувався би такою поведінкою: можливість циклічного переміщення «голови», проходження у двох напрямах, пошуку елементів за значенням, вклеювання і видалення груп елементів, порівняння двох кілець. Виключна ситуація – відсутність шуканого елемента або фрагмента кільця.

Варіант 15

Бібліотечний каталог задано у вигляді асоціативного масиву *map* (ключ – автор, значення – назва книги). Визначити функції:

- пошуку книг даного автора в каталозі (без додавання нового елемента);
- визначення автора потрібної книги, якщо книги немає, то розглядати це як виключну ситуацію;

- додавання нового елемента в масив;
- виключення елемента за заданим ключем з масиву.

Варіант 16

Реалізувати шаблон класу *set*, який імітував би роботу з множинами, аналогічно визначеному у Паскалі. Передбачити функцію, яка відповідала би логічній операції *in*, перевантажити операції +, - та * відповідно до семантики цих операцій у Паскалі. Виключна ситуація – намагання продублювати наявний у множині елемент.

Варіант 17

Задано вектор об'єктів «раціональний дріб». Передбачити функції:

- збільшення розмірів вектору на N елементів і заповнення значень нових елементів за певним законом (з використанням потрібної функції), виключна ситуація – відсутність вільної пам'яті потрібного розміру;
- створення нового вектора як копії старого з впорядкуванням перших n елементів.

Варіант 18

Перший вектор містить коефіцієнти поліному степеня *n* - $P_{n.,}$ другий – значення деякої величини *x* з певним кроком. Побудувати функції:

- перетворення значень елементів першого вектора на значення відповідного члена полінома для заданого значення x;
- створення третього вектора, і-й елемент якого дорівнює Pn(xi), де xi – поточне значення відповідного елемента із другого вектора.

Скористатись можливостями функцій із <*numeric*>.

Варіант 19

Задано два списки. Один складається із елементів «комплексне число», другий – із дійсних чисел. Побудувати функції:

- перетворення списку так, щоб кожний елемент містив значення квадрату відповідного числа;

- впорядкування списку за збільшенням.

Обидві функції перевірити для обох списків. Порівняння для комплексних чисел – за модулем.

Варіант 20

Задано два списки. Один складається із елементів «комплексне число», другий – із дійсних чисел. Побудувати функції:

- впорядкування тільки тих елементів списку, що задовольняють певну умову (більше двох за модулем для першого і більше двох для другого);
- розташувати елементи списку в довільному порядку.

Обидві функції перевірити для обох списків. Порівняння для комплексних чисел – за модулем.

Варіант 21

Задано масив коефіцієнтів поліному так, що *i*-й елемент масиву – коефіцієнт при x^i. Створити на основі цього масиву стек, який складатиметься з ненульових коефіцієнтів разом із значенням відповідного степеня полінома. Реалізувати функції:

- створення нового поліному як суми двох існуючих;
- порівняння двох поліномів.

Якщо модуль певного коефіцієнту поліному менший за деяке наперед задане значення ε, таку ситуацію розглядати як виключну, а такий елемент виключити із стеку.

Варіант 22

Побудувати шаблон класу *BTree* – «бінарне дерево» і передбачити:

- можливість додавання і видалення певного елемента (за значенням ключа), виключна ситуація – відсутність потрібного елемента;
- пошук (за ключем).

Допустимо скористатись стандартним контейнерним класом, доречність вибору обгрунтувати.

Варіант 23

Реалізувати шаблонний клас «динамічний вектор» на основі стандартного контейнера ***deque,*** в якому передбачити:

- можливість звернутись за індексом [] і at() – як в стандартному контейнері vector - з використанням виключної ситуації для перевірки діапазону;
- функцію для збільшення розміру «динамічного вектора» – аналогічну ***resize,*** але з можливістю збільшення вектора з обох сторін, з використанням переваг дека. Виключна ситуація – нестача пам'яті.

Контрольні запитання

1. Пояснити призначення, синтаксис і особливості використання шаблонів класів та функцій.
2. Пояснити механізм генерування та опрацювання виключень.
3. Пояснити загальний підхід до побудови і використання стандартного контейнера STL.
4. Пояснити принципи побудови і використання основних контейнерів, ітераторів і алгоритмів STL.

Література: [2, С. 81–103, 337–441];[5, С. 149–192]; [7, С. 227–239, 259–276, 311–326, 400–404];[8, С. 596–672]; [15, С. 70–114]

КОМП'ЮТЕРНИЙ ПРАКТИКУМ № 5. ОПИС АБСТРАКЦІЙ З ВИКОРИСТАННЯМ УСПАДКУВАННЯ ТА ПОЛІМОРФІЗМУ НА C#

(до теми: « Огляд інших об'єктно-орієнтованих мов програмування та відповідних інструментальних програмних засобів»)

Мета роботи: Навчитись коректно застосовувати основні властивості об'єктно-орієнтованого підходу з використанням особливостей *C#* для побудови працюючих програм.

Основні теоретичні відомості

Швидке розповсюдження суто об'єктно-орієнтованої мови *C#* та синтаксична схожість з *C++* обумовила вибір *C#* як другої мови для порівняння задіяних у них механізмів підтримки об'єктно-орієнтованого стилю.

Типи *C#* за способом збереження інформації розподілені на типи-значення і типи-посилання. До типів-значень належать перерахування та структурні типи, а саме: логічний, цілочисельні, дійсні, фінансовий, символьний, власне структурний. До типів-посилань – *object*, масиви, рядки, класи, інтерфейси, делегати.

Для створення нового об'єкта класу потрібно використати оператор *new*, наприклад: *object z= new object();*

Основні відмінності в описі класу, які потрібно врахувати у даній лабораторній роботі:

- ініціалізація полів відбувається одночасно з оголошенням;
- функції-члени повністю описані всередині опису класу;
- рівень доступу необхідно вказувати перед кожним членом класу;
- функцію-модифікатор і функцію-селектор для доступу до одного поля можна замінити властивістю.

Знищення об'єкта класу відбувається автоматично за допомогою механізму «прибирання сміття».

Структура консольного додатку. Оскільки мова суто об'єктно-орієнтована і не містить вільних функцій, у кожному додатку обов'язково присутній клас для визначення методу *Main()* , наприклад

```
using System;
namespace test
{
   public class B   { . . .  }
   class Program   {      static void Main(string[] args) {  . . .  }   }
}
```

За уведення-виведення відповідає клас *System.Console,* у якому визначені відповідні методи, наприклад:

```
Console.WriteLine("enter your name");
string s = Console.ReadLine();
```

Для перетворення типів використовують класс *Convert* або метод *Parse*, наприклад:

```
string s = Console.ReadLine();
int i=Convert.ToInt32(s);
string sl = Console.ReadLine();
double y=double.Parse(s1);
char c=(char)Console.Read();
```

З клавіатури дійсне число уводиться з десятковою комою. Для побудови математичних виразів використовують клас *Math.*

Успадкування. Підтримується лише від одного класу і відповідає відкритому успадкуванню в *C++*. Проте клас може реалізувати один або декілька інтерфейсів.

Похідний успадковує усі члени базового, крім конструкторів і деструктора.

Для явного виклику конструктора базового класу незалежно від назви цього класу використовується ключове слово *base* разом із потрібним списком параметрів, наприклад:

public Starosta(string a, string b, int l) : base(a,b) {level=l;}

Якщо у базовому класі метод, властивість, індексатор або подія оголошені віртуальними (*virtual)*, похідний може їх перевизначити з використанням ключового слова *override*.

Якщо невіртуальний метод похідного класу приховує метод базового, метод похідного класу краще оголосити з модифікатором *new*, інакше компілятор видасть попередження.

Інтерфейси. Визначені за допомогою ключового слова *interface* й описують набір функціональних можливостей, які можуть належати любому класу або структурі.

Використання інтерфейсів – альтернативний спосіб реалізації поліморфної поведінки. Типи, не зв'язані класичним успадкуванням, можуть демонструвати ідентичну поведінку.

Перевантаження операторів. Операторна функція обов'язково повинна бути статичним методом класу або структури. Хоча б один з операндів такого методу повинен бути об'єктом даного типу.

Оператор «=» не може бути перевантажений.

Допустиме перевантаження таких унарних операторів

!, ~, ++, - -, true, false

Допустиме перевантаження таких бінарних операторів

+, -, *, /, %, &, |, ^, <<, >>.

Для операторів з присвоєнням собі перевантаження не допустиме, але здійснюється автоматично, якщо перевантажений відповідний бінарний оператор. Наприклад, «+=» буде автоматично перевантажено, якщо для даного типу перевантажено «+».

Оператори порівняння допускають лише попарне перевантаження. Тобто «<» та «>», «<=» та «>=», «= =» та «*!=*».

Оператор *[]* не перевантажується, але існує конструкція індексатора, яка забезпечує аналогічні можливості.

Оператор *()* не перевантажується, але існує можливість перетворення користувацьких типів.

Оператор «=», який надається за умовчанням, реалізує переприсвоювання посиланя на той самий об'єкт.

Для реалізації глибокого копіювання використовується інтерфейс *ICloneable,* у якому визначений єдиний метод *object Clone().*

Для перевантаження логічних операторів «==» та «*!=*». спочатку перевизначають для даного типу метод *bool Equals(object)* із *System.Object* (виконує перевірку збігу посилань для двох об'єктів). Йому надається семантика перевірки за значеннями полів.

Перевизначення функції *Equals* вимагає перевизначити також функцію *int GetHashCode()* із *System.Object,* яка повертає значення, яке має однозначно ідентифікувати об'єкт в пам'яті.

Для перевантаження логічних операторів **«<»** та **«>»** потрібно спочатку визначити поле або комбінацію полів, за яким буде відбуватись порівняння. Тобто зробити так, щоб об'єкти даного типу можна було порівнювати.

Для цього потрібно реалізувати інтерфейс *IComparable* з визначенням функції *int CompareTo(object o),* яка повертає значення -1, 0, 1 в залежності від результату порівняння.

Порядок виконання роботи

Скористатись описами абстракцій, визначеними у лабораторній роботі №3. Клас *Date* не створювати.

Описи типів користувача переписати відповідно синтаксису мови C#. Врахувати наступне:

- у тих класах, де використовувалось поле типу *Date*, скористатись типом *System.DateTime;*
- для реалізації глибокого копіювання скористатись функцією *Clone()* (реалізація інтерфейсу *ICloneable*);
- функції-селектори та функції-модифікатори для доступу до закритих полів реалізувати як властивості;
- там, де це доцільно, скористатися специфікаторами *readonly, static, protected;*
- визначити перевантажені версії віртуального методу *ToString()* для кожного класу (за винятком класів-колекцій) з формуванням рядка зі значеннями усіх полів;
- для класу, у якому був визначений оператор індексації, визначити індексатор з одним параметром;
- там, де це доцільно, скористатись можливістю генерування виключної ситуації;
- арифметичні оператори перевизначити як статичні методи відповідно правилам C#;
- для реалізації логічних операторів «==»та «!=» потрібно:

  = перевизначити метод *Equals()* так, щоб відбувалась перевірка об’єктів, а не посилань на них;

  = перевизначити віртуальний метод *int GetHashCode(),* для чого скористатись любим рядком, що може ідентифікувати об’єкт у разі його використання в колекціях;

  = скористатись методом *Equals()* в операторних функціях для «==» та «!=»;

- для реалізації логічних операторів «<» та «>» або «<=» та «>=» потрібно скористатись функцією *CompareTo()* (реалізація інтерфейсу *IComparable*);
- оператори «<<» та «>>» не перевизначати, виведення повної інформації на консоль реалізувати як віртуальну функцію з використанням методу *ToString();*
- виведення скороченої інформації (наприклад, значення одного з полів) реалізувати як невіртуальні функції з однаковими іменами у базовому та похідному класах.

Реалізувати тестовий приклад, у якому продемонструвати:

- створення об’єкта та його копії;
- модифікацію одного з об’єктів;
- порівняння об’єктів;
- створення третього об’єкта з використанням конструктора умовчання;
- виконання арифметичної операції з присвоюванням третьому об’єкту та з присвоюванням собі для одного з об’єктів (наприклад, відповідно «+» і «+=»);
- використання індексатора;
- роботу віртуальної та невіртуальної функцій.

Контрольні запитання:

5. Надати класифікацію типів C#. Які конструкції мови надають можливість створення типів користувача?
6. Як відбувається конструювання та знищення об’єктів? Створення копій? Що таке властивість?
7. Які різновиди успадкування ви знаєте? Як відбувається констрування об’єктів похідних класів? Як реалізований механізм перевизначення функцій?

8. Дати визначення і пояснити призначення інтерфейсів.

9. Пояснити синтаксис і правила використання кожного різновиду операторних функцій.

Література: [3, С. 120–146, 151–183, 191, 203–225, 231–240,];[4, С. 73, 82, 106, 109, 127–143, 152];[11, С. 152–161, 172–198]

Список використаної літератури

1. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ [Текст] / Гради Буч. 2-е изд. — М. : БИНОМ, 1999. — 560 с.
2. Страуструп Б. Язык программирования C++ [Текст] / Бьерн Страуструп. 3-е изд. — М. : БИНОМ, 2004. — 1104 с.
3. Троелсен Э. Язык программирования и платформа.Net 2.0 [Текст] / Эндрю Троелсен. — М. : «И.Д.Вильямс», 2008. — 1168 с.
4. Нейгел К. C# 2005 для профессионалов / Кристиан Нейгел, Билл Ивьен, Джей Глин [и др.]. ; Пер. с англ. под ред. Ю. Н. Артеменко. – М. : «И.Д.Вильямс», 2007. — 1376 с.
5. Либерти Дж. C++. Энциклопедия пользователя [Текст] / Либерти Дж. – К.: «ДиаСофт», 2000. – 584с.
6. Вайнер Р. C++ изнутри [Текст] / Вайнер Р., Пинсон Л. – К. : «ДиаСофт», 1993. — 304с.
7. Пол А. Объектно-ориентированное программирование на C++ [Текст] / Айра Пол. 2-е изд. — М. : БИНОМ, 2001. — 464 с.
8. Либерти Дж. Освой самостоятельно C++ за 21 день [Текст] : учебн. пос. / Либерти Дж. – М.: «И.Д.Вильямс», 2001. — 816с.
9. Дейтел Х. Как программировать на C++ [Текст] / Дейтел Х., Дейтел П. – М.: БИНОМ, 2001. – 1152с.
10. Шлеер С. Объектно-ориентированный анализ: моделирование мира в состояниях [Текст] / Шлеер С., Меллор С. — К. : Диалектика, 1993. — 240 с.
11. Павловская Т.А. C#. Программирование на языке высокого уровня [Текст] : Учебник / Татьяна Александровна Павловская. – СПб. : Питер, 2009. – 432с.

12. Павловская Т.А. С++. Объектно-ориентированное программирование [Текст] : Практикум / Павловская Т.А., Щуляк Ю.А. – СПб. : Питер, 2006. – 265с.
13. Хабибуллин И.Ш. Самоучитель Java [Текст] / Ильдар Шауканович Хабибуллин – СПб. : БХВ, 2001. – 464с.
14. Об’єктно - орієнтоване програмування [Текст] : Метод. вказівки до лаб. робіт / Уклад. Назарчук І.В., Тимощук О.Л., Швачко Г.Г. – К. : ІВЦ «Політехніка», 2004. – 56с.
15. Назарчук І.В. Об’єктно - орієнтоване програмування [Текст] : Навч. посібн. / Ірина Василівна Назарчук – К. : ІВЦ «Політехніка», 2005. – 114с.

Додаток А. Вимоги до оформлення звіту

Звіт з компютерного практикуму починається з титульного аркуша, який має наступний вигляд:

Навчально-науковий комплекс «Інститут прикладного системного аналізу»
при Національному технічному університеті України «КПІ»
Кафедра математичних методів системного аналізу

Роботи компютерного практикуму
з курсу «Об'єктно-орієнтоване програмування»

Виконав: студент 3-го курсу
гр. КА-ХХ
Прізвище І.П.

Прийняв: Викладач І.П.

Київ 2011

Звіт з кожної роботи повинен містити:

1. Номер завдання та його умову
2. Діаграму класів (для першої та другої лабораторних робіт)
3. Текст програми
4. Результат роботи програми на контрольних прикладах

ЗМІСТ