

МЕТОДИЧНА РОЗРОБКА
для проведення лекційних занять
З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

«Об’єктно-орієнтоване програмування»

для студентів денної форми навчання за спеціальністю

121 «Інженерія програмного забезпечення»

(код і назва спеціальності)

Розроблено викладачем

_____ / Гапоненко Н.В. /

Розглянуто та схвалено

на засіданні ЦК ПІ

Протокол № _____ від _____ . 20 ____ р.

Голова ЦК ПІ

_____ /Ланська С.С. /

„_____” _____ 20 ____ р.

Лекція № 1

з навчальної дисципліни "Об'єктно-орієнтоване програмування"

Тема Методологія об'єктно-орієнтованого програмування. Робота з проектом

ВСТУП

Integrated Development Environment (інтегроване середовище розробки), або, скорочено, IDE — це програмний продукт, об'єднуючий текстовий редактор, компілятор, відладчик і довідкову систему. Microsoft Visual Studio, до складу якого входить IDE Microsoft Visual C++, це програмний продукт, який об'єднує текстовий редактор, компілятор, відладчик і довідкову систему.

Об'єктно-орієнтоване програмування — це методологія побудови програм, яка спирається на принципи інкапсуляції, успадкування і поліморфізму.

1 Середовище візуального програмування MVC++

Будь-яка програма, що створюється в середовищі Visual C++, навіть така проста, як «Hello, World!», завжди оформляється як окремий проект (project). Проект це набір взаємопов'язаних вихідних файлів і, можливо, що включаються (заголовних) файлів, компіляція і компоновання яких дозволяє створити виконувану програму. Проте, з'явилося поняття робочої області проекту (project workspace). Робоча область може містити будь-яку кількість різних проектів, згрупованих разом для узгодженої розробки: від окремого додатка до бібліотеки функцій або цілого програмного пакета.

Виклик Visual C++ здійснюється або через меню Пуск>Програми> Microsoft Visual C++, або клацанням мишею по піктограмі з відповідним ім'ям, розміщеної на робочому столі комп'ютера. Після запуску Visual C++ з'являється головне вікно програми. Робочий стіл Visual C++ включає в себе три вікна:

Вікно Project Workspace (вікно робочої області) призначено для надання допомоги при написанні і супроводі великих багатофайлова програм. Спочатку воно закрите, але після створення нового проекту (або завантажити їх раніше проекту) одна з вкладок цього вікна буде містити список файлів проекту.

Вікно Editor (вікно редактора) використовується для введення та перевірки вихідного коду.

Вікно Output (вікно виводу) служить для виведення повідомлень про хід компіляції, складання та виконання програми. Зокрема, повідомлення про виникаючі помилки з'являються саме в цьому вікні.

Під заголовком головного вікна, як і у всіх Windows-додатках, знаходиться рядок меню. Призначення команд меню і кнопок панелей інструментів ми будемо розглядати в міру необхідності, розбираючи основні прийоми роботи в IDE.

Середовище розробки дозволяє будувати проекти різних типів, орієнтовані на різні сфери застосування. Так як середовище спроектована на Windows-платформі, то майже всі типи проектів є віконними Windows-додатками з відповідним графічним інтерфейсом. У той же час розробники середовища передбачили роботу і з так званими консольними додатками. При запуску консольного застосування операційна система створює так зване консольне вікно, через яке йде весь ввід-вивід програми. Цей тип додатків найбільше підходить для цілей вивчення мови C / C ++, так як компільовані програми не «покриваються» товстим шаром проміжного Windows-коду.

2 Проект у MVC++

Для створення нового проекту типу «консольний додаток» виконайте наступні дії:

- 1 Виберіть у рядку меню головного вікна команду File> New ...
- 2 У діалоговому вікні New виберіть вкладку Projects.
- 3 Виберіть тип Win32 Console Application.
- 4 Введіть ім'я проекту в текстовому полі Project Name, наприклад First.
- 5 Введіть ім'я каталогу розміщення файлів проекту в текстовому полі Location1 (якщо зазначений вами каталог відсутній, то його буде створено автоматично).
- 6 Клацніть лівою кнопкою миші на кнопці OK.

Запускається так званий майстер додатків Application Wizard, який відкриває діалогове вікно Win32 Console Application з пропозицією визначитися, який підтип консольного застосування вам потрібно створити. Виберіть тип An empty project; клацніть на кнопці Finish.

З'явиться вікно New Project Information зі специфікаціями проекту та інформацією про каталог, в якому буде розміщено створюваний проект.

Перш ніж продовжувати роботу, звернемо тимчасово головне вікно Visual C++ на панель задач і заглянемо в папку First, створену майстром програм для нашого проекту, а точніше - для нашої робочої області. Там ми знайдемо файли First.dsw, First.dsp, First.opt, First.ncb, а також папку Debug (або Release - залежно від конфігурації проекту). Дамо короткий опис кожного з файлів:

First.dsw - файл робочої області проекту, який використовується всередині інтегрованого середовища розробки. Він об'єднує всю інформацію про проекти, які входять у цю робочу область.

First.dsp - проектний файл, який використовується для побудови (building) окремого проекту.

First.opt - файл, що містить опції робочої області проекту. Завдяки цьому файлу при кожному відкритті робочої області проекту всі параметри, вибрані під час останнього сеансу роботи з даної робочої областю, будуть відновлені.

First.ncb - службовий файл. Він створюється компілятором і містить інформацію, яка використовується в інструменті інтегрованого середовища під названням ClassView. Панель ClassView знаходиться у вікні Project Workspace і показує всі класи C++, для яких доступні визначення в рамках даного проекту, а також всі елементи цих класів.

Debug - папка, в яку будуть міститися файли, що формуються компілятором і збирачем. З цих файлів нас буде цікавити тільки один - виконуваний файл, що має розширення exe.

Розгорнемо головне вікно Visual C++ з відкритою робочою областю First, щоб продовжити роботу зі створення першої програми. Вікно Project Workspace має дві сторінки: Class View і File View. Клацанням миші перемкнемося на вкладку File View. Вона призначена для перегляду списку файлів проекту. Відкриємо список First files, клацнувши мишею на значку «+». З'явиться дерево списку файлів, що містить піктограми трьох папок: Source Files, Header Files, Resource Files. В консольних додатках файли ресурсів не використовуються. Перші дві папки порожні. Ми вибрали як підтип консольного застосування порожній проект - опцію An empty project.

Залишити (не закриваючи) середовище MVC++, відкрити папку Lab1_1, створену майстром додатків для нашого проекту, а точніше — для нашої робочої області, переглянути її вміст у вигляді таблиці. Там повинні розміщуватись файли:

Файл	Пояснення
Lab1_1.dsw	Project Workspace
Lab1_1.dsp	Project File
Lab1_1.cpp	C++ Source File
StdAfx.cpp	C++ Source File
StdAfx.h	C Header File
Lab1_1.opt	опції робочої області
Lab1_1.ncb	службовий файл

А також міститься папка Debug.

Дамо короткий опис файлів:

Lab1_1.dsw — файл робочої області проекту, використовуваний усередині інтегрованого середовища розробки. Він поєднує всю інформацію про проекти, що входять у дану робочу область.

Lab1_1.dsp — проектний файл, використовуваний для побудови (building) окремого проекту або підпроекту.

Lab1_1.opt — файл, що містить опції робочої області проекту. Завдяки цьому файлу при кожному відкритті робочої області проекту всі параметри середовища, обрані під час останнього сеансу роботи з даною робочою областю, будуть відновлені.

Lab1_1.ncb — службовий файл. Він створюється компілятором і містить інформацію, що використовується в інструменті інтегрованого середовища за назвою ClassView. Панель ClassView перебуває у вікні Project Workspace і показує всі класи C++, для яких доступні визначення в рамках даного проекту, а також всі елементи цих класів.

Debug — папка, у яку будуть міститися файли, формовані компілятором і збирачем. Із цих файлів нас буде цікавити, взагалі ж, тільки один - виконує файл, що, що має розширення .exe.

Повернутися у середовище MVC++. Відкрити у редакторі коду почергово файли StdAfx.cpp, Lab1_1.cpp, StdAfx.h, користуючись для цього сторінкою FileView браузера.

Відкрити файл Lab1_1.cpp і він змінити текст програми, додавши ще один рядок коду, наприклад, `printf("Hello !\n");`

Після цього зберегти проект за допомогою команди File⇒ Save All.

Виконати програму за допомогою команди Build⇒ Execute.

Закрити проект за допомогою команди File⇒ Close Workspace.

Закрити MVC++ за допомогою команди File⇒ Exit.

Висновки

Будь-яка програма , що створюється в середовищі Visual C + + завжди оформляється як окремий проект.

Середовище розробки дозволяє будувати проекти різних типів, орієнтовані на різні сфери застосування. При запуску консольного застосування операційна система створює так зване консольне вікно, через яке йде весь ввід-вивід програми. Цей тип додатків найбільше підходить для цілей вивчення мови C / C + +, так як компільовані програми не «покриваються» товстим шаром проміжного Windows-коду.

Питання та завдання до контролю знань студентів

1 Для актуалізації опорних знань та перевірки попереднього матеріалу

1 Що таке алгоритм?

2 В чому полягає процедурне програмування?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

1. Що входить до інструментарію пакету IDE Microsoft Visual C++?

2. Який склад консольного проекту у Microsoft Visual C++?

3. Які команди призначені для роботи з проектом.

Лекція № 2

з навчальної дисципліни "Об'єктно-орієнтоване програмування"

Тема Підвищення рівня абстракції програми

ВСТУП

Чим вище рівень абстракції програми, тим легше її супроводження та модифікація. Способи підвищення рівня абстракції програми:

- оформлення функцій та модулів;
- використання користувацьких типів даних (структур, класів).

Розглянемо дві ситуації для проєкту: додавання існуючого файлу, створення нового файлу.

Додавання нового файлу.

У цьому випадку необхідно виконати наступні дії:

1. Виберіть у рядку меню головного вікна команду File > New В результаті відкриється діалогове вікно New.

2. На вкладці Files: виберіть тип файлу (у даному випадку: C + + Source File) ; в текстовому полі File Name введіть ім'я файлу (у даному випадку: first.cpp); прапорець Add to project повинен бути включений.

Після зроблених кроків можна спостерігати наступні результати:

- у вікні Project Workspace папка Source Files списку файлів проєкту розкриється , і в неї буде поміщений файл first.cpp ;
- у вікні Editor з'явиться текстовий курсор.

У цьому випадку файл з вихідним кодом (нехай це буде файл first.cpp) ви вже підготували раніше в якомусь текстовому редакторі, викачали по мережі або принесли на дискеті.

Можна створити цей файл у редакторі середовища за допомогою команд File > New увійти на вкладку Files , вибрати C + + Source File і ввести в поле File Name ім'я файлу - first , натиснути ОК і в редакторі, що відкрився, ввести текст файлу, який наведено нижче.

Зберегти (або скопіювати) вихідний файл (first.cpp) в папку робочої області проєкту (в даному випадку - First) .

Поверніться до списку First files у вікні Project Workspace вашого проекту і клацніть правою кнопкою мишки на папці Source Files .

У контекстному меню клацанням миші виберіть команду додавання файлів Add Files to Folder

У діалоговому вікні Insert Files ... виберіть потрібний файл (first.cpp) і клацніть на кнопці ОК .

Папки Source Files і Header Files , піктограми яких ви бачите у вікні Project Workspace, насправді фізично не існують , то є всі файли поміщаються в основну папку робочої області проекту, ім'я якої було задано при створенні проекту у вікні Project Name. Але, погодьтеся, таке впорядкування дерева список файлів у вікні Project Workspace дуже зручно.

Компіляція, компоновка і виконання проекту.

Ці операції можуть бути виконані або через меню Build головного вікна, або за допомогою кнопок панелі інструментів. Опишемо коротко основні команди меню Build:

1 Compile - компіляція обраного файлу. Результати компіляції виводяться у вікно Output.

2 Build - компоновка проекту. Компілюються всі файли, в яких відбулися зміни з моменту останньої компонування. Після компіляції відбувається збірка (link) всіх об'єктних модулів, включаючи бібліотечні, в результуючий виконуваний файл. Повідомлення про помилки компоновки виводяться у вікно Output. Якщо обидві фази компонування завершилися без помилок, то створений виконуваний файл з розширенням .Exe може бути запущений на виконання.

3 Rebuild All - те ж, що і Build, але компілюються всі файли проекту незалежно від того, чи були в них зроблені зміни. Якщо при компонуванні багатофайлового проекту за допомогою команди Build ви отримуєте повідомлення про помилки компіляції або збірки, які ви не можете пояснити, то ми настійно рекомендуємо вам скористатися командою Rebuild All.

4 Execute - виконання виконуваного файлу, створеного в результаті компоновки проекту.

Операції Compile, Build і Execute зручніше виконувати через відповідні кнопки панелі інструментів.

Visual C++ дозволяє будувати проект або в відлагодчній конфігурації (Win32 Debug), або у випускній конфігурації (Win32 Release).

Як закінчити роботу над проектом. Можна вибрати меню File, пункт Close Workspace. А можна просто закрити програму Visual C++.

Як відкрити проект, над яким ви працювали раніше

1. Спосіб перший:

- Запустіть на виконання Visual C++.
- Виберіть меню File, пункт Open Workspace ...
- У діалоговому вікні знайдіть папку з вашим проектом, а в ній - файл ProjectName.dsw.

- Відкрийте цей файл, клацнувши по ньому мишею.

2. Спосіб другий:

- Запустіть на виконання Visual C++.
- Виберіть меню File, наведіть курсор миші на пункт Recent Workspaces.
- Якщо в меню зі списком останніх файлів, з якими йшла робота, ви знайдете який вас файл ProjectName.dsw, то клацніть по ньому мишею.

3. Спосіб третій:

- Не викликаючи Visual C++, знайдіть папку з вашим проектом, а в ній - файл ProjectName.dsw.
- Клацніть мишею на файлі ProjectName.dsw.

Середовище програмування (program environment) утворюється необхідними для створення програм засобами, до яких відносяться:

- редактор;
- препроцесор;
- компілятор;
- компоновальник;
- відладчик.

Редактор (editor) - програма, яка використовується для написання та зміни вихідних програм або даних.

Основні функції редактора середовища програмування MVC ++:

- введення і збереження на диску файлів з текстами вихідних програм;
- виділення фрагментів тексту, їх копіювання і перенесення;
- синтаксична розмальовка тексту програми;
- форматування тексту програми.

Препроцесор (preprocessor) - програма, яка виконує попередню обробку вихідної програми для компілятора.

Компілятор (compiler) - засіб для перекладу (трансляції) вихідної програми, написаної мовою програмування, в сукупність зрозумілих комп'ютеру машинних команд. Компілятор може виявити помилки трансляції (compile errors), а значить, він не зможе створити об'єктну програму. Як правило, це синтаксичні помилки (syntax errors). Компілятор також може повідомляти попередження (warnings), які необхідно пропрацювати як помилки.

Компонувальник (linker) - засіб зв'язування об'єктних програм з кодами функцій із стандартних бібліотек, що забезпечує збірку (компонування) виконуваної програми. Під час зв'язування об'єктних файлів можуть виникати помилки компонування (linker errors). Ці помилки показують, що вказані в програмі зовнішні посилання не можуть бути знайдені компоновщиком.

Тестові приклади (test cases) - контрольні набори тестових даних для перевірки правильності функціонування виконуваної програми. Тестові приклади готуються заздалегідь. Для кожного прикладу записуються контрольні значення необхідних вихідних даних, а також записуються очікувані вихідні дані (результати роботи виконаної програми). Тестові приклади повинні враховувати наступні можливі варіанти для вхідних даних:

- значення даних належать допустимому діапазону зміни;
- значення даних знаходяться на межі діапазону зміни;
- значення даних заборонені, тобто знаходяться поза дозволеного діапазону.

Під час перевірки виконуваної програми, можуть виникати помилки виконання (run-time errors). Однак помилка не завжди призводить до зупинки програми, і лише частина результатів може виявитися невірною. Така форма помилки виконання є логічною помилкою (logical error).

Відладчик (debugger) - засіб, призначений для аналізу поведінки виконуваної програми, що забезпечує її покрокове виконання.

Висновки

Ніяких особливих зусиль при створенні багатофайлових проектів докладати не доведеться. Треба буде просто кілька разів повторити процедуру створення / додавання вихідних файлів, описану вище. У багатофайлових проектах зазвичай присутні і заголовні файли - вони створюються / додаються після клацання правою кнопкою миші на піктограмі папки Header Files у вікні Project Workspace, при цьому на вкладці Files діалогового вікна New вибирається тип файлу C / C + + Header File, а при введенні імені файлу використовується розширення. h.

Питання та завдання до контролю знань студентів

1 Для актуалізації опорних знань та перевірки попереднього матеріалу

1 Що таке об'єктно-орієнтоване програмування?

2 Як створити консольний проєкт?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

1. Як підвищити рівень абстракції програми?

2. Для чого підвищують рівень абстракції програми?

3. Охарактеризувати кроки компіляції, компоновки.

Лекція

з навчальної дисципліни "Об'єктно-орієнтоване програмування"

Тема Поняття класу. Структура та опис класу

Мета заняття Ознайомитись з поняттям класу, вивчити структуру та опис класу

ВСТУП

Прообразом класу в C++ є структура в C. У той же час в C++ структура знайшла нові властивості і тепер є окремим видом класу, всі елементи якого за умовчанням є відкритими. Зі структурою `struct` в C++ можна робити все, що можна робити з класом. Проте в C++ структури зазвичай використовують лише для зручності роботи з невеликими наборами даних без будь-якої власної поведінки.

Сукупність принципів проектування, реалізації та розробки програм, яка базується на абстракції даних, передбачає створення нових типів даних, які з найбільшою повнотою відображають особливості задачі, яка розв'язується. В мові C++ програміст має можливість ввести власні типи даних та визначити операції над ними за допомогою класів.

1 Поняття класу

Клас – це абстрактний тип даних, що описує модель об'єкту реального світу. Механізм класів дозволяє створювати типи у повній відповідності до принципів абстракції даних, тобто клас задає деяку структуровану сукупність типізованих даних та дозволяє визначити набір операцій над цими даними.

Клас можна визначити за допомогою конструкції:

ключ_класу ім'я_класу {список членів класу};

- де ключ_класу – службове слово `class`;
- ім'я_класу – правильний ідентифікатор;
- список членів класу – визначення та описання типізованих даних та функцій, які належать класу.

Приклад описання структури, яку перетворимо в клас, наведений в лістингу 1.

Лістинг 1 – Структура, а далі клас «комплексне число»

```
struct complex //«комплексне число»
{
double real; //реальна частина
```

```
double imag; //уявна частина
//визначення значення комплексного числа
void define (double re = 0.0, double im = 0.0)
{
    real = re;
    imag = im;
}
//друк значення комплексного числа на екран
void display (void)
{
    cout << "real = " << real << " ,imag = " << imag;
}
};
```

Отже, клас – це тип, введений програмістом. Кожен тип служить для визначення об'єктів. Для описання об'єкта класу використовується конструкція:

ім'я_класу ім'я об'єкту;

В об'єкти (класу) входять дані (елементи), відповідні членам даним класу. Члени функції класу дозволяють обробляти дані конкретних об'єктів класу.

Визначення об'єктів класу передбачає виділення ділянки пам'яті та ділення цієї ділянки на фрагменти, відповідні окремим елементам об'єкта, кожен із котрих відображає окремий компонент даних класу.

Як тільки об'єкт є визначеним, з'являється можливість звертатися до його компонент, по-перше, за допомогою «кваліфікованих» імен, кожне з яких має формат: ім'я_об'єкта.ім'я_класу::ім'я компоненту

Ім'я класу з операцією вказівки області дії «::» звичайно може не використовуватися та дуже часто для доступу к даним конкретного об'єкту заданого класу (як і у випадку структур) використовується уточнене ім'я:

ім'я_об'єкту.ім'я_елементу;

Інший спосіб доступу до елементів об'єкту деякого класу передбачає використання вказівника на об'єкт класу та операції доступу →:

вказівник_на_об'єкт_класу → ім'я_елементу;

Вказівник на об'єкт класу дозволяє викликати функції, які належать до класу, для обробки даних того об'єкту, який адресується вказівником.

Для ініціалізації об'єктів класу в його визначення можна явно включати спеціальну компонентну функцію, яка називається конструктор. Формат визначення конструктора в тілі класу може бути таким:

ім'я_класу (список_формальних_параметрів) {оператори_тіла_конструктора};

Приклад конструктора для класу `complex` наведений в лістингу 3.2.

Лістинг 3.2 – Приклад класу `complex`

```
complex (double re = 0.0, double im = 0.0) {  
    real = re;  
    imag = im;  
}
```

Щоб від структури перейти до класу, треба `struct` замінити на `class` і додати специфікатор доступу `public`.

Ім'я цієї компонентної функції за правилами мови C++ повинно збігатися з іменем класу. Така функція автоматично викликається при визначенні або розташуванні в пам'яті за допомогою оператора `new` кожного об'єкту класу.

Конструктор не можна викликати як звичайну компонентну функцію. Для явного виклику конструктора можна використати дві різні синтаксичні форми:

- ім'я_класу ім'я_об'єкту (фактичні_параметри_конструктора);
- ім'я_класу (фактичні_параметри_конструктора);

Перша форма допускається тільки при непорожньому списку фактичних параметрів. Вона передбачає виклик конструктора при визначенні нового об'єкта даного класу:

```
complex ss(10.3, 0.22); //ss.real = 10.3; //ss.imag = 0.22;  
complex ee(2.345); //ee.real = 2.345; //ee.imag = 0.0 (за замовченням)  
complex dd();
```

Динамічне виділення пам'яті для об'єктів якого-небудь класу створює необхідність у звільненні цієї пам'яті при знищенні об'єкту. Таку можливість забезпечую спеціальний компонент класу – деструктор (знищувач об'єктів) класу. Для нього передбачений стандартний формат:

~ім'я_класу () {оператори тіла деструктора};

Назва деструктору в C++ завжди починається з символу тильда «~», за яким без пропусків та інших роздільних знаків поміщується ім'я класу. У деструктора не може бути параметрів (навіть типу `void`). Деструктор не має повертати жодного значення (навіть

типу void). Виклик деструктора виконується неявно, автоматично, як тільки об'єкт класу знищується.

C++ належить до об'єктно-орієнтованих мов програмування, в яких головними елементами програм є об'єкти і класи.

Загальний опис для всіх об'єктів із загальними властивостями і поведінкою називається класом об'єктів. Клас показує, як побудувати існуючу в часі і просторі змінну цього класу - об'єкт.

2 Структура та опис класу

Клас (class) - визначає тип даних, який описує безліч об'єктів із загальними властивостями, поведінкою і семантикою. Кожен клас має унікальне ім'я, що визначає назву нового типу даних. Наприклад, для безлічі книг можна визначити клас CBook.

Об'єкт (object) - змінна типу клас чи екземпляр класу. За правилами оголошення змінних для класу CBook можна, наприклад, оголосити об'єкт цього класу як CBook book.

У зв'язку з тим, що тип класу об'єднує дані і операції обробки цих даних в єдине ціле, для класів вводиться поняття членів класу (class members). До членів класу відносяться дані (data members) і функції (members function).

Член класу (дане) - іменована характеристика або властивість об'єктів класу, яке має власне значення для кожного об'єкта. Наприклад, у класі CBook можна оголосити дані author, title, year, які будуть визначати автора, назву та рік видання для об'єктів-книг класу. Для кожної книги ці дані будуть мати конкретні значення.

Член класу (функція) - це функція, що належить класу, за допомогою якої об'єкт класу взаємодіє з об'єктами інших класів або того ж самого класу. Функції-члени прийнято називати методами. Метод являє собою дію, яку повинен виконати об'єкт для реалізації свого поведінки, або сервіс, який може бути затребуваний одним об'єктом в іншого.

Оголошення класу (class declaration) являє собою опис членів класу : даних і методів. Часто оголошення класу називають специфікацією. Члени -дані оголошуються згідно з правилами оголошення змінних. Функції- члени в специфікації представлені прототипами.

Визначення оголошених у специфікації класу методів розташовується в окремому файлі з розширенням. Сpp , званому файлом реалізації класу .

Коли файли специфікації і реалізації класу знаходяться в різних файлах , заголовок функції- члена повинен включати область видимості згідно формату :

тіп_функції - члена ім'я_класу :: ім'я_функції - члена (список параметрів)

Оператор дозволу області видимості :: використовується в заголовку з тією метою, щоб всі оголошені усередині класу ідентифікатори оброблялися всередині їх власного простору імен, а також щоб уникнути неоднозначності використання імен. Запис тіла методу нічим не відрізняється від запису в звичайної функції. У файл реалізації класу слід включати файл зі специфікацією класу, а також необхідні при визначенні функцій заголовки стандартних бібліотек.

Висновки

Клас (class) - визначає тип даних, який описує безліч об'єктів із загальними властивостями, поведінкою і семантикою. Кожен клас має унікальне ім'я, що визначає назву нового типу даних.

Об'єкт (object) - змінна типу клас чи екземпляр класу. За правилами оголошення змінних для класу CBook можна, наприклад, оголосити об'єкт цього класу як CBook book.

Член класу (дане) - іменована характеристика або властивість об'єктів класу, яке має власне значення для кожного об'єкта.

Член класу (функція) - це функція, що належить класу, за допомогою якої об'єкт класу взаємодіє з об'єктами інших класів або того ж самого класу. Функції-члени прийнято називати методами.

Питання та завдання до контролю знань студентів

1 Для актуалізації опорних знань та перевірки попереднього матеріалу

1. Що таке структура?
2. Що таке масив?
3. Що таке покажчик?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

1. Що таке клас?
2. Що таке об'єкт класу?
3. Як описується клас?
4. 3 Що є членами класу?

Лекція

з навчальної дисципліни "Об'єктно-орієнтоване програмування"

Тема Поняття класу. Специфікатори доступу

Особливість об'єктно-орієнтованого програмування полягає в приховуванні даних, що захищає дані класу від несанкціонованого доступу функцій, розташованих за межами класу.

1 Поняття класу

Клас (class) - визначає тип даних, який описує безліч об'єктів із загальними властивостями, поведінкою і семантикою. Кожен клас має унікальне ім'я, що визначає назву нового типу даних. Наприклад, для безлічі книг можна визначити клас CBook.

Об'єкт (object) - змінна типу клас чи екземпляр класу. За правилами оголошення змінних для класу CBook можна, наприклад, оголосити об'єкт цього класу як CBook book.

У зв'язку з тим, що тип класу об'єднує дані і операції обробки цих даних в єдине ціле, для класів вводиться поняття членів класу (class members). До членів класу відносяться дані (data members) і функції (members function).

2 Специфікатори доступу

Специфікатори доступу визначають доступність даних і методів у програмі. Існують наступні специфікатори доступу:

public - відкритий режим доступу. Члени класу доступні за межами класу будь-яких функцій програми;

protected - захищений режим доступу. Члени класу доступні для методів цього класу, похідних класів і дружніх класів.

private - закритий режим доступу. Члени класу доступні тільки для методів цього класу та класів-друзів.

Особливість об'єктно-орієнтованого програмування полягає в приховуванні даних, що захищає дані класу від несанкціонованого доступу функцій, розташованих за межами класу. У зв'язку з цим члени-дані, як правило, мають

специфікатор доступу `private`. В окремих випадках виникає необхідність дані класу робити відкритими.

Методи, які надають сервісні послуги класу, завжди оголошуються з специфікатором `public`. Методи, використовувані класом для власних потреб, мають специфікатор доступу `private`. Ці методи забезпечують реалізацію відкритих функцій-членів і приховують складності їх реалізації.

Специфікація класу, як правило, розміщується в окремому заголовному файлі з розширенням. Н і має наступний формат, наведений в лістингу 4.1.

Лістинг 4.1 – Специфікація класу

```
class ім'я_класа {  
private:  
    // закриті члени класа  
protected:  
    // захищені члени класа  
public:  
    // відкриті члени класа } ;
```

Оголошення класу починається з ключового слова `class`, за яким слідує ім'я класу, і закінчується крапкою з комою. У середині фігурних дужок може бути будь-яке кількість секцій зі специфікаторами доступу, причому специфікатори можуть повторюватися.

Якщо після відкриваючої фігурної дужки не стоїть ніякої специфікатор доступу, члени класу за замовчуванням стають `private` (закритими).

У лістингу 4.2 наведений приклад класу `CBook`/

Лістинг 4.2 – Приклад класу `CBook`

```
// Book.h – специфікація класу CBook  
#pragma once  
class CBook {  
private:  
    char m_author [ 50 ] ;    // автор  
    char *m_pTitle ;          // вказівник на назву  
    int m_year ;              // рік видання  
public:  
    // методи встановлення значень  
    void setAuthor ( const char* ) ;  
    void setTitle ( const char* ) ;  
    void setYear ( const int ) ;  
    // методи повернення значень  
    char* getAuthor ( void ) ;  
    char* getTitle ( void ) ;
```

```
int getYear ( void ) ;  
} ;
```

Файл специфікації Book.h містить оголошення типу класу CBook . Щоб компілятор включав оголошення типу тільки один раз при трансляції програми , використовується директива препроцесора # pragma once .

У класі оголошені закриті дані з типом масиву m_author , покажчика на символьний тип m_pTitle і цілого m_year , які описують автора, назву і рік видання , відповідно. Тип CBook повідомляє компілятору, що для створення об'єкта типу CBook потрібно 50 байтів для зберігання автора, 4 байти для адреси, за якою буде записано назву книги , і 4 байта для зберігання року видання.

Щоб можна було змінювати значення даних у об'єктів класу і отримувати їх значення у програмі, в класі оголошено відкриті методи. Функції- члени з типом значення, що повертається void мають параметр , тип якого відповідає типу даного. Методи для повернення значень не потребують параметрах, оскільки змінні, значення яких вони повертають, відомі класу. Тип кожного методу обумовлений типом повертаємої змінної.

У лістингу 4.3 наведена реалізація класу CBook.

Лістинг 4.3 – Реалізація класу CBook

```
// Book.cpp - реалізація класа CBook  
#include "Book.h"  
#include <string>  
// встановити автора  
void CBook :: setAuthor ( const char* author ) {  
    strncpy_s ( m_author, 50, author, 49 ) ;  
    if ( strlen ( author ) > 49 ) m_author [ 49 ] = '\\0' ;  
}  
// встановити назву  
void CBook :: setTitle ( const char* title ) {  
    delete [ ] m_pTitle ;  
    m_pTitle = new char [strlen ( title )+ 1] ;  
    strcpy_s ( m_pTitle, strlen ( title ) + 1, title ) ;  
}  
// встановити рік видання  
void CBook :: setYear ( const int year )  
{    m_year = year ;    }  
// повернути автора  
char* CBook :: getAuthor ( void )  
{    return m_author ;    }  
// повернути назву  
char* CBook :: getTitle ( void )
```

```
{    return m_pTitle ;    }  
// повернути рік видання  
int CBook :: getYear ( void )  
{    return m_year ;    }
```

Висновки

`private` — закритий специфікатор доступу.

Методи, які надають сервісні послуги класу, завжди оголошуються з специфікатором `public`. Методи, використовувані класом для власних потреб, мають специфікатор доступу `private`. Ці методи забезпечують реалізацію відкритих функцій-членів і приховують складності їх реалізації.

Методи для повернення значень не потребують параметрах, оскільки змінні, значення яких вони повертають, відомі класу. Тип кожного методу обумовлений типом повертаємої змінної.

Питання та завдання до контролю знань студентів

1 Для актуалізації опорних знань та перевірки попереднього матеріалу

1. Що таке клас?
2. Що є членами класу?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 За допомогою чого регулюється доступ до елементів класу?
- 2 Які є специфікатори доступу, який з них за замовчуванням в класі?
- 3 Описати, який доступ надає кожний специфікатор доступу.

Лекція

з навчальної дисципліни "Об'єктно-орієнтоване програмування"

Тема Поняття об'єкту. Конструктори та деструктор

Оголошення об'єкта класу (object definition) не варто плутати з оголошенням класу. Важливо розуміти, що клас не містить ніяких значень даних, а лише описує загальну структуру і поведінку об'єктів, будучи як би трафаретом для їх створення.

Опис об'єкта задає його тип (ім'я класу) і, можливо, необхідні для ініціалізації членів-даних значення. При оголошенні об'єкта компілятор отримує вказівку на створення змінної класу на підставі заданого типу.

1 Поняття об'єкту

Клас (class) - тип даних, що визначається програмістом, який описує безліч об'єктів із загальними властивостями, поведінкою і семантикою. Кожен клас має унікальне ім'я, що визначає назву нового типу даних. Наприклад, для безлічі книг можна визначити клас CBook.

Об'єкт (object) - змінна типу клас чи екземпляр класу. За правилами оголошення змінних для класу CBook можна, наприклад, оголосити об'єкт цього класу як CBook book.

Оголошення або специфікація класу міститься у файлі специфікації (Book.h).

Реалізація класу або визначення оголошених у специфікації класу методів розташовується в окремому файлі з розширенням. cpp, званому файлом реалізації класу.

Оголошення об'єкта класу.

Оголошення об'єкта класу (object definition) не варто плутати з оголошенням класу. Важливо розуміти, що клас не містить ніяких значень даних, а лише описує загальну структуру і поведінку об'єктів, будучи як би трафаретом для їх створення.

Опис об'єкта задає його тип (ім'я класу) і, можливо, необхідні для ініціалізації членів-даних значення. При оголошенні об'єкта компілятор отримує вказівку на створення змінної класу на підставі заданого типу.

Коли об'єкт оголошується, то згідно з описом класу для об'єкта відбувається виділення оперативної пам'яті, а також при вказівці значень даних здійснюється

ініціалізація членів-даних зазначеними значеннями. Всю цю роботу робить спеціальний метод класу, званий конструктором.

Оголошення об'єкта може розташовуватися в головній функції, як наведено в лістингу 5.1.

Лістинг 5.1 – Оголошення об'єкта

```
// главная функция
#include <iostream>
using namespace std ;
#include "Book.h"
void view ( char*, CBook& ) ;
int main ( )
{   }
```

Основними форматами оголошення об'єкту класу є наступні :

```
ім'я_класа ім'я_об'єкта ;
ім'я_класа ім'я_об'єкта ( список параметрів ) ;
ім'я_класа ім'я_об'єкта ( ім'я_об'єкта копіювання ) ;
```

Приклади оголошення об'єктів класу наведені в лістингу 5.2.

Лістинг 5.2 – Приклади оголошення об'єктів класу

```
CBook book, aBook [ 100 ] ;
CBook obj ( "Carrol L.", "Alice in Wonderland", 2002 ) ;
CBook copy ( obj ) ;
```

У прикладі використані всі три формати для оголошення об'єктів типів класу CBook. Згідно з першим , оголошується об'єкт book і масив з 100 Об'єктів Book . За другим форматом оголошений об'єкт obj , і за третім - об'єкт copy.

Слід сказати , що створення об'єктів в пам'яті відрізняється від створення звичайних змінних (простих або структурованих). Про виділення пам'яті для об'єктів піклується спеціальний метод класу - конструктор. Ім'я цього методу збігається з ім'ям класу.

Опис методу- конструктора (як і інших методів класу) , розташовується у двох файлах : специфікації класу і реалізації класу .

У файлі специфікації класу *. h розміщується рядок заголовка методу , у файлі реалізації класу *.cpp розміщується визначення методу .

2 Конструктори та деструктори

При створенні об'єкту автоматично викликається спеціальний метод, який називається конструктором (constructor). У конструктора ім'я класу і обов'язково відкритий специфікатор доступу. Конструктор управляє побудовою об'єкта в оперативній пам'яті.

Процес побудови включає в себе виділення пам'яті для членів-даних типу показчиків, а також ініціалізацію даних.

Конструктор ніколи не повертає ніякого значення і не має типу, навіть void. Конструктор перевантажується, тому що об'єкти можна ініціалізувати по-різному. Кількість конструкторів практично не обмежується і залежить від доцільності їх наявності в класі. Існують конструктор за умовчанням, конструктор з параметрами і конструктор копіювання.

Важливо пам'ятати, ніякий конструктор не повинен робити нічого, що стосується обробки членів-даних. За обробку членів-даних несуть відповідальність не конструктори, а інші методи в класі.

Якщо у класі є конструктор, то він викликається завжди, коли створюється об'єкт класу. Якщо у класі є деструктор, то він викликається завжди, коли об'єкт класу знищується.

Об'єкти можуть створюватися як:

1 Автоматичний об'єкт: створюється щоразу, коли його опис зустрічається при виконанні програми, і знищується щоразу при виході з блоку, в якому воно з'явилося.

2 Статичний об'єкт: створюється один раз, при запуску програми, і знищується один раз, при її завершенні.

3 Об'єкт у вільній пам'яті: створюється за допомогою операції new і знищується за допомогою операції delete.

4 Об'єкт член - як об'єкт іншого класу.

Об'єкт також може бути сконструйований за допомогою явного застосування конструктора у виразі, в цьому випадку він є автоматичним об'єктом.

Передбачені різні способи ініціалізації об'єкта з використанням конструктора. Один з них - це явний виклик конструктора.

```
Person pers_yr = Person (1999);
```

Інший - це неявний виклик конструктора.

```
Person pers_yr2 ( 2010 );
```

Конструктор використовується також при виконанні операції динамічного створення об'єкту

```
Stock * pstock = new Stock ( " Games" , 18, 19.0 ) ;
```

Конструктор не може бути викликаний об'єктом , конструктор використовується для створення об'єкту.

Ініціалізатор в конструкторах.

Головним завданням конструктора класу є ініціалізація даних створюваних ним об'єктів.

Ініціалізація здійснюється за допомогою списку ініціалізації, розташованого між двокрапкою, що слідує за круглими дужками в заголовку конструктора, і тілом конструктора. Значення, що присвоюються, записуються в круглих дужках після ідентифікаторів членів-даних і відокремлюються один від одного комами.

Порядок проходження ідентифікаторів у списку ініціалізації не має значення.

Конструктор за замовчанням.

Конструктор за замовчанням - конструктор, який не потребує параметрів. Цей конструктор завжди повинен бути визначений для будь-якого класу. Конструктор за замовчуванням може не виконувати жодних дій, але найчастіше він ініціалізує дані класу нульовими значеннями.

Оголошення конструктора за замовчуванням має наступний формат:

public: ім'я класу ();

Наприклад, для класу CBook прототип конструктора за замовчуванням записується зі специфікатором відкритого доступу

public: CBook ();

Приклад реалізації конструктора за замовчуванням

CBook:: CBook (): m_year (0), m_pTitle (nullptr)

{m_author [0] = ' ' ;}

Для ініціалізації прізвища автора не може бути використаний ініціалізатор, так як цей член-дане оголошений в класі як символьний масив

Деструктор

Деструктор (destructor) - спеціальний метод класу, який використовується для руйнування об'єктів класу. Ім'я деструктора збігається з ім'ям конструктора (ім'ям класу), якому передує символ тильда ~. Він завжди має відкритий специфікатор доступу і не має ні типу, ні параметрів, деструктор управляє знищенням об'єкта з оперативної пам'яті.

Викликається деструктор автоматично при руйнуванні об'єкта. Якщо об'єкт створювався динамічно через покажчик за допомогою оператора new, то для знищення такого об'єкта слід використовувати оператор delete для покажчика.

Типовим використанням деструктора є звільнення динамічної пам'яті, яка раніше була виділена конструктором.

Оголошення деструктора має наступний формат:

```
public: ~ім'я_класу ();
```

Наприклад, для класу CBook прототип деструктора в специфікації класу має вигляд:

```
~ CBook ();
```

Приклад реалізації деструктора

```
CBook::~ ~ CBook () {delete [] m_pTitle;}
```

Результатом дії деструктора ~ CBook () є звільнення блоку пам'яті з початковою адресою з m_pTitle, виділеною раніше конструктором класу.

Конструктор і деструктор викликаються в програмі автоматично при оголошенні об'єктів класу. Немає значення, оголошується об'єкт явно або створюється динамічно за допомогою new.

Виклик того чи іншого конструктора залежить від форми опису об'єкту:

Якщо після імені об'єкта або типу класу в операторі new нічого не вказано або стоять порожні круглі дужки, викличеться конструктор за замовчанням.

Якщо в дужках записані параметри, викликається конструктор з параметрами.

Якщо в дужках знаходиться ім'я об'єкта, то відбувається виклик конструктора копіювання.

Крім того, конструктор викликається всередині функцій для створення копії фактичного параметра, якщо функція отримує цей параметр за значенням. Відомо, що при виході з функції копія знищується, отже автоматично буде викликаний деструктор для руйнування копії.

Конструктори викликаються в програмі в порядку оголошення об'єктів. Коли оголошується масив об'єктів, конструктор за замовчуванням буде викликаний стільки разів, скільки елементів зазначено в оголошенні масиву.

При оголошенні масиву вказівників на тип класу або одиничного вказівника ніякі конструктори не викликаються. Вони будуть викликані пізніше при ініціалізації вказівників оператором `new`.

Деструктор викликається автоматично при руйнуванні оголошеного на ім'я об'єкта, тому немає необхідності в явному виклику деструктора для руйнування локального об'єкта або для руйнування об'єктів перед виходом з програми. Деструктор також викликається автоматично, коли за допомогою оператора `delete` звільняється покажчик на об'єкт класу. Без оператора `delete` об'єкт зруйнований не буде. Дозволяється і явний виклик деструктора для об'єкта, який виконується, як і виклик будь-якого іншого методу класу.

Деструктори викликаються в порядку, зворотному порядку оголошення іменованих об'єктів у програмі. Виклик деструкторів об'єктів, оголошених через покажчики, визначається проходженням операторів `delete` для цих покажчиків.

Висновки

Конструктор і деструктор викликаються в програмі автоматично при оголошенні об'єктів класу. Не має значення, оголошується об'єкт явно або створюється динамічно за допомогою `new`.

Деструктор викликається автоматично при руйнуванні оголошеного на ім'я об'єкта, тому немає необхідності в явному виклику деструктора для руйнування локального об'єкта або для руйнування об'єктів перед виходом з програми. Деструктор також викликається автоматично, коли за допомогою оператора `delete` звільняється покажчик на об'єкт класу.

Питання та завдання до контролю знань студентів

- 1 Для актуалізації опорних знань та перевірки попереднього матеріалу
 - 1 За допомогою чого регулюється доступ до елементів класу?
 - 2 Які є специфікатори доступу, який з них за замовчуванням в класі?
 - 3 Описати, який доступ надає кожний специфікатор доступу.
- 2 Для узагальнення та перевірки засвоєного матеріалу на лекції
 1. Що таке клас?
 2. Які конструктори має клас?

3. Як використовується деструктор?

Лекція № 6

з навчальної дисципліни "Об'єктно-орієнтоване програмування"

Тема Поняття об'єкту. Вказівник this

Мета заняття Вивчити поняття об'єкту та ознайомитись з вказівником this

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН

Ілюстрації у вигляді рисунків

Час – 2 години

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5	Включає в себе: <i>привітання</i> , яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; <i>визначення присутності студентів на занятті</i>
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10	Включає в себе: <i>вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми</i>
3 Основна частина (викладення навчальних питань лекції)	55	Головне місце приділяється <i>викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень</i> . Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	15	Включає в себе: <i>узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу</i>

Література

1 Довбуш, Г. Ф. Visual C++ на прикладах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

2 Павловская, Т. А. С/C++. Программирование на языке высокого уровня [Текст] / Т. А. Павловская. – М. - СПб. : ПИТЕР, 2003. - 461 с.

Навчальні матеріали лекції

ВСТУП

Оперативна пам'ять для об'єктів виділяється при їх створенні. Для кожного об'єкту виділяється пам'ять, розмір якої визначається сумою розмірів членів-даних класу. Якщо в класі член-дане оголошується за допомогою вказівника або посилання, то блок оперативної пам'яті для розміщення цього даного буде виділятися динамічно.

Функції-члени розміщуються в оперативну пам'ять в одному екземплярі і сумісно використовуються всіма існуючими об'єктами цього класу. Функції-члени не займають область пам'яті, виділену для об'єкта.

1 Поняття об'єкту

Для об'єктів, які ми описуємо в програмі, виділяється пам'ять. Якщо створюється кілька об'єктів, то пам'ять виділяється окремо для полів кожного об'єкта, а методи займають пам'ять в єдиному екземплярі.

Оперативна пам'ять для об'єктів виділяється при їх створенні. Для кожного об'єкту виділяється пам'ять, розмір якої визначається сумою розмірів членів-даних класу. Якщо в класі член-дане оголошується за допомогою вказівника або посилання, то блок оперативної пам'яті для розміщення цього даного буде виділятися динамічно.

Функції-члени розміщуються в оперативну пам'ять в одному екземплярі і сумісно використовуються всіма існуючими об'єктами цього класу.

Функції-члени не займають область пам'яті, виділену для об'єкта.

Доступ до членів об'єктів

Будь-який об'єкт, як змінна свого класу, включає члени-дані і знає про методи обробки цих даних. Доступ до членів об'єкта забезпечується операторами точка та стрілка, як і доступ до елементів структур.

Доступ до даних і методів визначається значенням специфікатора доступу `private`, `protected` або `public`. Тільки специфікатор `public` дозволяє звертатися до елементів з будь-якого місця в програмі. Два інших специфікатора дозволяють доступ лише об'єктів того ж самого класу або його друзям.

Для отримання доступу на ім'я об'єкта необхідно використовувати оператор крапка.

Наприклад, `book.m_author`, `obj.m_pTitle`, `copy.m_author`, `obj.getTitle()`.

Для доступу через покажчик використовується оператор стрілка. Наприклад, якщо відомий покажчик `p` на тип `CBook`, то записується `p -> m_author`, `p -> getTitle ()`.

2 Вказівник `this`

Схема розміщення у пам'яті об'єктів і методів класу наведена на рисунку 6.1.

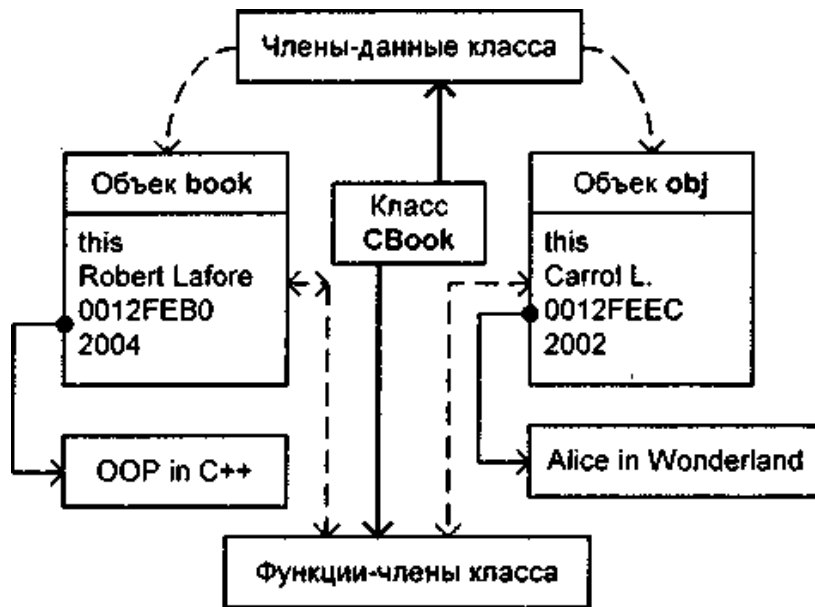


Рис. 6.1 – Схема розміщення у пам'яті об'єктів і методів класу

Кожен об'єкт має свій власний вказівник `this`.

При виклику методів, які належать класу, їм неявно передається той самий покажчик `this`. Це відбувається автоматично і непомітно для нас, так як він є прихованим. Покажчик `this` вказує на створений об'єкт класу і містить його адресу.

Є можливість використовувати покажчик `this` явно. Так, наприклад, виглядало б визначення методу `getData ()`, якби ми використовували `this` явно: `this->getData ()`.

Висновки

Будь-який об'єкт, як змінна свого класу, включає члени-дані і знає про методи обробки цих даних, але не включає їх. Через те, що об'єкт класу не включає методи, а звертається до методів, які знаходяться в одному екземплярі, методу необхідно знати, який об'єкт до нього звернувся. Для цього введено поняття `this` — константного прихованого вказівника на об'єкт класу.

Під час створення об'єкта цей вказівник отримує адресу об'єкта, яка буде автоматично використовуватися в методах, але може бути використана і явно.

Показчик `this` не є частиною самого об'єкта і не відбивається в результаті оператора `sizeof` об'єкта. Замість цього при виклику нестатичних функції-членів для об'єкта компілятор передає адресу об'єкта в функцію в якості прихованого аргументу.

Вираз `* this` зазвичай використовується для повернення поточного об'єкта з функції-члена.

Питання та завдання до контролю знань студентів

1 Для актуалізації опорних знань та перевірки попереднього матеріалу

1. Що таке клас?
2. Які конструктори має клас?
3. Як використовується деструктор?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Що таке вказівник `this`?
- 2 Для чого введено поняття вказівника `this`?

Лекція

з навчальної дисципліни "Об'єктно-орієнтоване програмування"

Тема **Функції-члени класу**

Мета заняття Закріпити поняття функцій-членів класу

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН

Лістинги з фрагментами програмного коду

Література

Довбуш, Г. Ф. Visual C++ на прикладах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

ВСТУП

Коли об'єкт оголошується, то згідно з описом класу для об'єкта відбувається виділення оперативної пам'яті, а також при вказанні значень даних здійснюється ініціалізація членів-даних зазначеними значеннями. Всю цю роботу робить спеціальний метод класу, званий конструктором.

Оперативна пам'ять для об'єктів віділяється при їх створенні.

Функції-члени розміщуються в оперативну пам'ять в одному екземплярі і сумісно використовуються всіма існуючими об'єктами цього класу. Функції-члени не займають область пам'яті, віділену для об'єкта.

1 Функції-члени класу

Серед інших функцій-членів конструктор виділяється тим, що його ім'я збігається з ім'ям класу. Для оголошення конструктора за замовчуванням ми пишемо код, приклад якого наведений в лістингу 7.1.

Лістинг 7.1 – Фрагмент програмного коду з оголошенням конструктора

```
class Account {
public:
    // конструктор за замовчуванням...
    Account ();
    // ...
private:
    char *_name;
    unsigned int _acct_nmbr;
```

```
double _balance;  
};
```

Синтаксичне обмеження, що накладається на конструктор, полягає в тому, що він не повинен мати тип значення, що повертає, навіть void.

Кількість конструкторів в одному класі може бути будь-яким, аби тільки всі вони мали різні списки формальних параметрів.

Звідки ми знаємо, скільки і які конструктори визначити? Як мінімум, необхідно присвоїти початкове значення кожному члену, який це потребує. Наприклад, номер рахунку або задається явно, або генерується автоматично таким чином, щоб гарантувати його унікальність. Припустимо, що він буде створюватися автоматично. Тоді ми повинні дозволити ініціалізувати два члени, що залишилися `_name` і `_balance`:

```
Account (const char *name, double open_balance);
```

Об'єкт класу `Account`, ініціалізуємий конструктором, можна об'явити в такий спосіб:

```
Account newAcct ("Mikey Matz", 0);
```

Якщо ж є багато рахунків, для яких початковий баланс дорівнює 0, то корисно мати конструктор, що задає тільки ім'я власника й автоматично ініцілізує `_balance` нулем.

Один зі способів зробити це - надати конструктор виду:

```
Account (const char *name);
```

Інший спосіб - включити в конструктор із двома параметрами значення за замовчуванням, рівне нулю:

```
Account (const char *name, double open_balance = 0.0);
```

Обидва конструктори володіють необхідною користувачеві функціональністю, тому обидва рішення прийнятні. Можна використати аргумент за замовчуванням, оскільки в такій ситуації загальне число конструкторів класу скорочується.

Потрібно підтримувати також завдання одного лише початкового балансу без вказівки імені клієнта? У цьому випадку специфікація класу явно забороняє це. Наш конструктор із двома параметрами, з яких другий має значення за замовчуванням, надає повний інтерфейс для задання початкових значень тих членів класу `Account`, які можуть бути ініціалізовані користувачем. У лістингах 7.2, 7.3 наведені приклади.

Лістинг 7.2 – Фрагмент програмного коду з оголошенням конструкторів

```
class Account {  
public:  
    // конструктор за замовчуванням...
```

```

Account ();
// імена параметрів в оголошенні вказувати необов'язково
Account (const char*, double=0.0);
const char* name () { return name; }
// ...
private:
// ...
};

```

Лістинг 7.3 – Приклади правильного визначення об'єкта класу Account, де конструкторів передається один або два аргументи:

```

int main () {
// правильно: в обох випадках викликається конструктор з двома
параметрами
Account acct ("Ethan Stern");
Account *pact = new Account ("Michael Lieberman", 5000);
if (strcmp (acct. name (), pact->name ()))
// ...
}

```

C++ вимагає, щоб конструктор застосовувався до певного об'єкта до його першого використання. Це означає, що як для acct, так і для об'єкта, на який указує pact, конструктор буде викликаний перед перевіркою в інструкції if.

Компілятор перебудовує нашу програму, вставляючи виклики конструкторів.

Звичайно, якщо конструктор визначений як вбудований, то він підставляється в точці виклику.

Обробка оператора new трохи складніше. Конструктор викликається тільки тоді, коли він успішно виділив пам'ять.

Існує три в загальному випадку еквівалентні форми завдання аргументів конструктора:

```
Account acct1 ("Anna Press"); // загалом ці конструктори еквівалентні
```

```
Account acct2 = Account ("Anna Press");
```

```
Account acct3 = "Anna Press";
```

Форма acct3 може використовуватися тільки при завданні єдиного аргументу. Якщо аргументів два або більше, рекомендовано користуватися формою acct1, хоча припустимо й acct2.

```
Account acct1 ("Anna Press"); // рекомендуєма форма
```

Визначати об'єкт класу, не вказуючи списку фактичних аргументів, можна в тому випадку, якщо в ньому або об'явлений конструктор за замовчуванням, або взагалі немає об'яв конструкторів. Якщо в класі об'явлений хоча б один конструктор, то не дозволяється визначати об'єкт класу, не викликаючи жодного з них. Зокрема, якщо в

класі визначений конструктор, що приймає один або більше параметрів, але не визначений конструктор за замовчуванням, то в кожному визначенні об'єкта такого класу повинні бути присутнім необхідні аргументи. Можна заперечити, що не має змісту визначати конструктор за замовчуванням для класу Account, оскільки не буває рахунків без імені власника.

На практиці часто потрібно задавати конструктор за замовчуванням, якщо є які-небудь інші конструктори.

А якщо для класу немає розумних значень за замовчуванням? Наприклад, клас Account вимагає задавати для будь-якого об'єкта прізвище власника рахунку.

У такому випадку найкраще встановити стан об'єкта так, щоб було видно, що він ще не ініціалізований коректними значеннями:

```
// конструктор за замовчуванням для класу Account
inline Account::Account () {
    _name = 0;
    _balance = 0.0;
    _acct_nmbr = 0;
}
```

Однак у функції-члени класу Account прийдеться включити перевірку цілісності об'єкта перед його використанням.

Існує й альтернативний синтаксис: список ініціалізації членів, у якому через кому вказуються імена й початкові значення. Наприклад, конструктор за замовчуванням можна переписати в такий спосіб:

```
// конструктор за замовчуванням класу Account з використанням
// списку ініціалізації членів
inline Account::Account () : _name (0), _balance (0.0), _acct_nmbr (0) { }
```

Такий список допустимий тільки у визначенні, але не в оголошенні конструктора. Він міститься між списком параметрів і тілом конструктора й відділяється двокрапкою. От як виглядає наш конструктор із двома параметрами при частковому використанні списку ініціалізації членів:

```
inline Account::Account (const char* name, double opening_bal) : _balance (opening_bal) {
    _name = new char [strlen (name) + 1];
```

```
strcpy (_name, name);  
_acct_nmbr = get_unique_acct_nmbr ();  
}
```

Деструктор класу.

Одна із цілей, що ставляться перед конструктором, - забезпечити автоматичне виділення ресурсу. В прикладі із класом Account є конструктор, де за допомогою оператора new виділяється пам'ять для масиву символів і привласнюється унікальний номер рахунку. Можна також уявити ситуацію, коли потрібно одержати монопольний доступ до поділюваної пам'яті або до критичної секції потоку. Для цього необхідна симетрична операція, що забезпечує автоматичне звільнення пам'яті або повернення ресурсу після завершення часу життя об'єкта, - деструктор. Деструктор - це спеціальна обумовлена користувачем функція-член, що автоматично викликається, коли об'єкт виходить із області видимості або коли до покажчика на об'єкт застосовується операція delete. Ім'я цієї функції створено з імені класу з попереднім символом "тильда" (~). Деструктор не повертає значення й не приймає ніяких параметрів, а отже, не може бути перевантажений.

Хоча дозволяється визначати кілька конструкторів, лише один з них буде застосовуватися до всіх об'єктів класу. От, наприклад, деструктор для нашого класу Account, що наведено в лістингу 7.4. Зверніть увагу, що в нашому деструкторі не скидаються значення членів.

Лістинг 7.4 – Клас з деструктором

```
class Account {  
public:  
    Account ();  
    explicit Account (const char*, double=0.0);  
    Account (const Account&);  
    ~Account ();  
    // ...  
private:  
    char *_name;  
    unsigned int _acct_nmbr;  
    double _balance;  
};  
inline Account::~~Account () {  
    // необхідно  
    delete [] _name;  
    return_acct_number (_acct_nnmbbr);  
}
```

```
// необов'язково
_name = 0;
_balance = 0.0;
_acct_nmbr = 0;
}
```

Робити це необов'язково, оскільки відведена під члени об'єкта пам'ять однаково буде звільнена. Розглянемо наступний клас:

```
class Point3d { private: float x, y, z; };
```

Конструктор тут необхідний для ініціалізації членів, що представляють координати точки. Чи потрібний деструктор? Ні. Для об'єкта класу Point3d не потрібно звільняти ресурси: пам'ять виділяється й звільняється компілятором автоматично на початку й наприкінці його життя.

В загальному випадку, якщо члени класу мають прості значення, скажімо, координати точки, то деструктор не потрібний. Не для кожного класу необхідний деструктор, навіть якщо в нього є один або більше конструкторів. Основною метою деструктора є звільнення ресурсів, виділених або в конструкторі, або під час життя об'єкта, наприклад звільнення пам'яті, виділеної оператором new.

Але функції деструктора не обмежені тільки звільненням ресурсів. Він може реалізовувати будь-яку операцію, що за задумом проектувальника класу повинна бути виконана відразу по закінченні використання об'єкта.

Висновки

Конструктор тут необхідний для ініціалізації членів-даних.

Основною метою деструктора є звільнення ресурсів, виділених або в конструкторі, або під час життя об'єкта, наприклад звільнення пам'яті, виділеної оператором new.

Конструктори і деструктор теж є функціями-членами класу, тобто, методами.

Питання та завдання до контролю знань студентів

1 Для актуалізації опорних знань та перевірки попереднього матеріалу

1 Що таке вказівник this?

2 Для чого введено поняття вказівника this?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

1. Що таке функції-члени класу?
2. Яке призначення функцій-членів класу?
3. Як описуються функції-члени класу?

Лекція № 8

з навчальної дисципліни "Об'єктно-орієнтоване програмування"

Тема **Інкапсуляція та приховування інформації**

Мета заняття Ознайомитись з поняттям інкапсуляції та приховування інформації

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН

Лістинги з фрагментами програмного коду

Література

Павловская, Т. А. С/С++. Программирование на языке высокого уровня [Текст] /

Т. А. Павловская. – М. - СПб. : ПИТЕР, 2003. - 461 с.

ВСТУП

Особливість об'єктно-орієнтованого програмування полягає в приховуванні даних, що захищає дані класу від несанкціонованого доступу функцій, розташованих за межами класу. У зв'язку з цим члени-дані, як правило, мають специфікатор доступу `private`. В окремих випадках виникає необхідність дані класу робити відкритими.

Поняття інкапсуляції є одним з базових понять об'єктно-орієнтованого програмування.

Зміна значень полів відбувається за допомогою функцій.

1 Інкапсуляція та приховування інформації

Термін інкапсуляція тлумачиться в ООП в двох значеннях.

З одного боку, це розміщення даних і функцій, які обробляють ці дані, в межах одного класу.

З іншого боку, під інкапсуляцією розуміють приховування даних об'єкту з метою забезпечення доступу до них тільки за допомогою методів класу.

Тобто, доступ до даних об'єкту забезпечується за допомогою двох методів: методу установки значення даного, який ще називається методом запису даного (сеттер), та методу отримання значення даного, який ще називається методом читання даного (геттер).

Специфікатори доступу визначають доступність даних і методів у програмі. Існують наступні специфікатори доступа:

public - відкритий режим доступу. Члени класу доступні за межами класу будь-яких функцій програми;

protected - захищений режим доступу. Члени класу доступні для методів цього класу, похідних класів і друзів;

private - закритий режим доступу. Члени класу доступні тільки для методів цього класу та друзів.

Методи, які надають сервісні послуги класу, завжди оголошуються з специфікатором public.

Методи, використовувані класом для власних потреб, мають специфікатор доступу private. Ці методи забезпечують реалізацію відкритих функцій-членів і приховують складності їх реалізації.

Компонент (об'єкт) можна розглядати як якийсь «чорний ящик», коли відомі тільки виконувані їм функції і «входи» і «виходи». У програмуванні сукупність «входів» і «виходів» визначає інтерфейс компонента. Інтерфейс реалізується як набір деяких функцій (або запитів до компоненту), викликаючи які клієнт або отримує якусь інформацію, або змінює стан компонента. Слово «клієнт» означає просто компонент, якому знадобилися послуги іншого компонента, виконуючого в цьому випадку роль сервера.

Методи, розташовані у відкритій частині, формують інтерфейс класу і можуть вільно викликатися клієнтом через відповідний об'єкт класу. Доступ до закритої частини класу можливий тільки з його власних методів, а до захищеної - з його власних методів а також з методів класів-нащадків.

Інкапсуляція підвищує надійність програм, запобігаючи ненавмисний помилковий доступ до полів об'єкту. Крім цього, програму легше модифікувати, оскільки при збереженні інтерфейсу класу можна міняти його реалізацію, і це не торкнеться зовнішній програмний код (код клієнта).

Є ще одне досить точне визначення поняття інкапсуляції.

Інкапсуляція - це здатність керувати доступністю (іноді вживають термін видимістю) даними об'єкта для кого-небудь, відмінного від самого об'єкта.

У найпростішому випадку втілення ідей інкапсуляції полягає в поділі інтерфейсу і реалізації.

Управляти доступністю об'єкта можна досить гнучко, і в C++ це досягається за допомогою специфікаторів `public` (загальні дані), `protected` (захищені дані) і `private` (приватні дані). Крім власне самого класу, загальні (або публічні) дані доступні всім, захищені - тільки нащадкам класу, і цей тип доступності буде розглянуто більш детально пізніше, приватні дані - нікому. (Зауважимо, що специфікатор може бути відсутнім, тоді члени класу вважаються приватними, але читаність коду від цього програє.)

Однак необхідність отримувати доступ до даних одного класу тільки з якого-небудь, цілком певного, іншого класу, породила таке явище, як дружні класи та функції. Таким чином, наявність специфікатором і дружніх класів і функцій дозволяє досить гнучко управляти доступністю об'єктів.

Належним чином спроектувавши взаємодіючі класи, можна домогтися того, що до якихось даними екземплярів якогось конкретного класу зможуть отримувати доступ або всі, або тільки об'єкти якихось цілком певних класів.

Інкапсуляцію можна вважати основоположною властивістю об'єктно-орієнтованого програмування. Успадкування та поліморфізм є як би способом втілення концепції інкапсуляції.

Інкапсуляція - це завдання не стільки етапу програмування, скільки проектування. Оскільки це поняття досить загальне і його втілення в конкретному кодї може бути вельми різноманітним, затурбуватися питаннями незалежності даних, доступу до них і їх приховування слід ще на етапі проектування. При програмуванні про це думати буде вже пізно, незважаючи на весь ітеративний процес розробки програм.

Причому при проектуванні абсолютно не порушується питання реалізації принципів інкапсуляції в конкретних мовах, що підтримують об'єктно-орієнтований підхід, будь то C++, настільки популярна нині Java або мало кому відомий Smalltalk. Але ж і там концепція інкапсуляції реалізована, хоча і дещо інакше, ніж в C++. В ідеалі, добре проектування робить неістотним питання, якою об'єктно-орієнтованою мовою вести розробку.

Навіть можна поглянути на це ширше: принципи інкапсуляції реалізуються практично на всіх досить високорівневих мовах програмування. Просто в деяких мовах підтримка інкапсуляції вбудована, що значно полегшує життя.

Твердження про те, що та чи інша система задовольняє вимогам інкапсуляції - досить умовні. Так, наприклад, деякі вважають не найкращим способом реалізації ідей інкапсуляції досить поширене рішення, коли якась змінна класу оголошується як приватна, а для операцій з нею надаються дві функції: одна для отримання значення цієї змінної, а інша - для встановлення її значення, що наведено в лістингу 8.1.

Лістинг 8.1 – Функції для отримання та встановлення значення змінної

```
class A
{
private:
int var;
public:
int get_var () {
return var;
}
void set_var (int v) {
var = v;
}
};
```

Основним способом організації інформації в C++ є класи. У C++ тип структура аналогічний типу клас, але є відмінність в тому, що за умовчанням поля і функції-члени у структури публічні, а у класу — приватні.

З публічними полями можна робити зовні класу все, що завгодно. До захищених і приватних полів не можна звертатися ззовні класу, щоб не порушити цілісність даних класу. Спроба такого звернення викличе помилку компіляції. До таких полів можуть звертатися тільки функції-члени класу (а також так звані функції-друзі і функції-члени класів-друзів)

Поза тілом функцій-членів (а також друзів) захищені і власні поля недоступні навіть для читання. Такий захист полів називається інкапсуляцією.

Використовуючи інкапсуляцію можна захистити свої дані від некоректного використання. Зміна способу зберігання даних, якщо вони оголошені як захищені або приватні не вимагає відповідних змін в класах, які використовують змінений клас.

Функції-члени, як і поля, можуть бути публічними, захищеними і приватними.

Публічні функції може викликати будь-хто, а захищені і власні — тільки функції-члени і друзі.

Інкапсуляція — один з трьох принципів об'єктно-орієнтованого програмування і дизайну. Також відомий як приховування даних.

Зміст інкапсуляції полягає у приховуванні від зовнішнього користувача деталей реалізації об'єкту, замість цього надаючи інтерфейс взаємодії. Слугує передусім для того, щоб не давати можливості клієнту змінювати внутрішній стан об'єкта без відома останнього, залишаючи лише можливість впливу через «офіційні» методи роботи з полями (як правило, вони звуться `set` і `get`) та власне методи.

Інкапсуляція надає засоби забезпечення цілісності даних стану. Замість визначення відкритих полів (які можуть сприяти порушенню цілісності даних) слід взяти за звичку визначати закриті поля даних, якими викликаючи сторона може маніпулювати шляхом визначення: пари традиційних методів доступу і зміни (`get`, `set`); іменованих властивостей.

Добре інкапсульований клас повинен приховувати і самі необроблені дані, і те, як він ними оперує. Цей підхід часто називають програмуванням методом чорного ящика. Автор класу може вільно змінювати внутрішню реалізацію методів, не порушуючи існуючий код їх використання (за умови, що сигнатури методів залишаються такими самими). Тепер доступитися до них ззовні не можна. Нагадаємо, що модифікатор доступу `private` дозволяє доступитися до членів лише всередині класу.

Реалізація інкапсуляції традиційними методами доступу і зміни даних.

Якщо ви хочете, аби зовнішній світ міг взаємодіяти із закритим полем даних `salary`, потрібно визначити метод доступу (`get`) і метод зміни (`set`).

Висновки

Компонент (об'єкт) можна розглядати як якийсь «чорний ящик», коли відомі тільки виконувані їм функції і «входи» і «виходи». У програмуванні сукупність «входів» і «виходів» визначає інтерфейс компонента. Інтерфейс реалізується як набір деяких функцій (або запитів до компоненту), викликаючи які клієнт або отримує якусь інформацію, або змінює стан компонента. Слово «клієнт» означає просто компонент,

якому знадобилися послуги іншого компонента, виконуючого в цьому випадку роль сервера.

Завдяки використанню інкапсуляції, досягаються щонайменше дві переваги:

Можливість зміни внутрішнього представлення даних без необхідності зміни програмного коду клієнта.

Зменшення ймовірності помилок через приведення екземпляра до непередбаченого стану.

Питання та завдання до контролю знань студентів

1 Для актуалізації опорних знань та перевірки попереднього матеріалу

1. Що таке функції-члени класу?
2. Яке призначення функцій-членів класу?
3. Як описуються функції-члени класу?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Що таке інкапсуляція?
- 2 Що являє собою метод запису?
- 3 Що являє собою метод читання?

Перевантаження функцій

Перевантаження функцій (function overloading) є використанням одного й того ж імені для декількох функцій. При виклику перевантаженої функції компілятор визначає відповідну функцію шляхом аналізу кількості, типів та порядку слідування параметрів у виклику. Перевизначення функцій має відрізнятися або типами параметрів, або їх кількістю. Якщо за однакового списку параметрів функції будуть відрізнятися тільки типом значення, що повертається, компілятор повідомить про помилку неоднозначності. Параметри за замовчуванням так само можуть стати джерелом помилок з тієї причини, що виклик однієї функції може виявитися аналогічним виклику іншої перевантаженої функції.

Лістинг – Приклад перевантаження функцій

```
#include <iostream>

using namespace std ;

void swapR ( int&, int& ) ;
void swapR ( char&, char& ) ;
void reverseArray ( int*, int = 10 ) ;
int* reverseArray ( const int*, int, int ) ;
char* reverseArray ( const char* ) ;
void view ( const int*, int = 5 ) ;

int main ( )
{
    int ar [ 10 ] = { 1, 2, 3, 4, 5, 6 } ;
    cout << "Initial array\n" ; view ( ar, 10 ) ;
    // реверс всього масиву
    reverseArray ( ar ) ;
    cout << "\nReverse array\n" ; view ( ar, 10 ) ;
    // реверс частини масиву
    int* p = reverseArray ( ar, 5, 9 ) ;
    cout << "\nReverse part array\n" ; view ( p ) ;
    // реверс строки
    char* s = "NOMEL ON NOLEM ON" ;
    char* pr = reverseArray ( s ) ;
```

```

cout << "\nInitial string\n" ; cout << s << endl ;
cout << "\nReverse string\n" ; cout << pr << endl ;
return 0 ;
}

// перестановка з використанням посилань
void swapR ( int& rx, int& ry )
{ int t = rx ; rx = ry ; ry = t ; }

void swapR ( char& rx, char& ry )
{ char t = rx ; rx = ry ; ry = t ; }

// реверс масиву в собі
void reverseArray ( int* p, int size )
{
for ( int i = 0, j = size - 1; i < j; i++, j-- )
swapR ( p [ i ], p [ j ] ) ;
}

// формування зворотного масиву з частини іншого масиву
int* reverseArray ( const int* pIn, int top, int end )
{
if ( end <= top ) return 0 ;
int size = end - top + 1 ; int* pRes = new int [ size ] ;
int i, j, ir, jr ;
for ( i = top, j = end, ir = 0, jr = size - 1; i < j; )
{
pRes [ ir++ ] = pIn [ j-- ] ;
pRes [ jr-- ] = pIn [ i++ ] ;
}
if ( i == j ) pRes [ ir ] = pIn [ i ] ;
return pRes ;
}

// реверс рядка
char* reverseArray ( const char* p )

```

```

{
char* pOut = new char [ strlen ( p ) + 1 ] ;
strcpy_s ( pOut, strlen ( p ) + 1, p ) ;
char *pTop = pOut, *pEnd = pOut + strlen ( pOut ) - 1 ;
while ( pTop < pEnd )
swapR ( *pTop++, *pEnd-- ) ;
return pOut ;
}

```

У лістингу є дві перевантажені функції: `swapR ( )` та `reverseArray ( )`. Функція `swapR ( )` перестановки значень перевантажена для параметрів посилань на цілий та символьний тип. Таким чином, функції відрізняються типами параметрів зі списку параметрів. Версія з посиланнями на цілий тип викликається всередині функції реверсу масиву цілих чисел `swapR (p[i], p [j])`, а з посиланнями на символьний тип - у функції реверсу рядка `swapR ( * pTop++, * pEnd--)`. Другий виклик вимагає розіменування, оскільки `pTop` та `pEnd` є покажчиками.

Перевантажена функція `reverseArray ( )` має три варіанти реалізації, які різняться типом значення, що повертається, типом параметрів, а також їх кількістю.

Прототип `void reverseArray (int *, int = 10);` оголошує функцію, яка використовується для реверсу масиву цілих чисел у собі. У головній функції `main ( )` вона викликається як `reverseArray ( ar )`, використовуючи значення за умовчанням 10 для кількості елементів.

Прототип `int* reverseArray (const int*, int, int);` відрізняється від попереднього типом значення, що повертається, і не використовує значень за замовчуванням. Функція призначена для створення нового зворотнього масиву з частини вихідного. Адреса вихідного масиву задається першим параметром. Два останні параметри визначають початковий та кінцевий індекс елементів у вихідному масиві, які мають бути переписані у новий масив у зворотньому порядку. Функція повертає покажчик на новий масив. Перший параметр є вказівником на константу, що забороняє зміну елементів у вихідному масиві. У попередньому варіанті таке рішення неможливе тому, що реверс усередині масиву вимагає перестановки існуючих значень. Два останні

параметри не можна оголосити параметрами за умовчанням, оскільки виникне невизначеність, і компілятор не зможе розрізнити виклики цих функцій.

Обробка масиву починається з перевірки заданих у виклику індексів. Повертається 0, якщо кінцевий індекс менше або дорівнює початковому. Інакше обчислюється розмір необхідної пам'яті, динамічно виділяється блок пам'яті відповідно до обчисленого значення і запускається цикл `for`, в якому формується новий масив. Велика кількість змінних у циклі обумовлена тим, що необхідні два індекси `i`, `j` для переміщення по вихідному масиву і ще два індекси `ir`, `jr` для переміщення по створюваному масиву. Індекс `i` змінюється від значення, заданого параметром `top`, в сторону збільшення на 1 до тих пір, поки він буде меншим за індекс `j`. Індекс `j` змінюється від значення, заданого параметром `end`, у бік зменшення на 1. Індекс `ir` змінюється від 0, оскільки індексація у масиві починається саме з цього значення і збільшується на 1 для нової ітерації циклу. Індекс `jr` зменшується на 1 наприкінці кожної ітерації, починаючи від значення індексу останнього елемента нового масиву `size - 1`. Коли кількість елементів, що підлягають реверсу, непарна, на виході з циклу індекси `i` і `j` будуть рівні. При істинному відношенні `i == j` в новий масив переписується центральний елемент заданого інтервалу. Функція повертає покажчик на сформований масив.

У головній функції оператор `int * p = reverseArray (ar, 5, 9);` призначений для виклику реверсу частини масиву `ar`, обмеженого індексами 5 і 9. В результаті покажчик `p` отримає повернене функцією значення, а `main ( )` - доступ через цей покажчик до елементів зворотної частини масива `ar`. Сам масив `ar` не змінюється. Слід пам'ятати, що операцію `delete` не можна використовувати для вказівника `p` в головній функції, тому що динамічного виділення пам'яті вона не виконувала.

Прототип `char* reverseArray (const char*)`; оголошує перевантажену функцію для реверсу рядка. Функція не змінює вихідний рядок, покажчик на яку є її параметром. Довжину рядка передавати немає необхідності, оскільки вона може бути обчислена. Функція повертає покажчик на динамічно виділений блок пам'яті, в якому розташований результат. На початку функції виділяється необхідний простір оперативної пам'яті, куди копіюється вихідний рядок. Реверс проводиться з рядком, розташованим за виділеною адресою `pOut`, для чого в циклі `while` багаторазово

викликається перевантажена функція перестановлення елементів `swapR ( )`. У головній функції оголошується покажчик `pr`, якому присвоюється значення, повернене викликом функції `reverseArray (s)`. Стан вихідного рядка `s` і результуючого `pr` виводиться після виклику функції, причому рядок `s` не змінює свого значення після виклику. Результати виконання програми можна побачити на рис.

```
Initial array
1 2 3 4 5
6 0 0 0 0

Reverse array
0 0 0 0 6
5 4 3 2 1

Reverse part array
1 2 3 4 5

Initial string
NOMEL ON NOLEM ON

Reverse string
NO MELON NO LEMON
```

Рис. – Результати виконання програми з використанням перевантажених функцій

Запитання до матеріалу:

- Що таке перевантаження функцій?
- Як компілятор розрізняє, яку функцію викликають?
- Наведіть відомий приклад перевантаження методів класу.

Лекція

з навчальної дисципліни "Об'єктно-орієнтоване програмування"

Тема Дружні функції. Дружні класи. Використання дружніх функцій та класів
для доступу до членів класу

Мета заняття Ознайомитися з поняттям дружніх функцій та дружніх класів.
Навчитися використовувати дружні функції та дружні класи

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН
Лістинги з фрагментами програмного коду

Час – 2 години

Література

1 Довбуш, Г. Ф. Visual C++ на прикладах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

2 Глушаков, С.В. Язык программирования C++. [Текст] / С.В. Глушаков, А.В.Коваль, С.В.Смирнов. - Х.:Фолио, 2001. - 500 с.

Навчальні матеріали лекції

ВСТУП

Як відомо, програми можуть звертатися до приватних (private) елементів класу тільки за допомогою функцій-елементів цього ж класу. Використовуючи приватні елементи класу замість загальних у всіх ситуаціях, де це тільки можливо, зменшується можливість програми зіпсувати значення елементів класу, так як програма може звертатися до таких елементів тільки через інтерфейсні функції (які управляють доступом до приватних елементів).

Проте залежно від використання об'єктів вашої програми, іноді можна істотно збільшити продуктивність, дозволяючи сторонній функції або методу іншого класу або іншому класу безпосередньо звертатися до приватних елементів даного класу. У подібних ситуаціях C++ дозволяє визначити сторонню функцію або метод іншого класу або інший клас як друга (friend) і дозволяє другу доступ до приватних елементів нашого класу.

Приватні елементи та друзі.

Використання у програмах на C++ друзів дозволяє одному класу звертатися до приватних елементів іншого класу прямо, використовуючи оператор доступу.

Щоб обмежити кількість дружніх методів, які можуть звертатися до приватних даних класу, C++ дозволяє вказати дружні функції.

Розглянемо, як програми можуть зазначити, що стороння функція або метод іншого класу або інший клас є друзями.

- Використовуючи ключове слово `friend`, клас може повідомити C++, хто є його другом, тобто іншими словами, що інші функції/методи/класи можуть звертатися безпосередньо до його приватних елементів.

- Приватні елементи класу захищають дані класу, отже, слід обмежити коло друзів тільки тими, яким дійсно необхідний прямий доступ до приватних елементів нашого класу. Тобто, немає потреби оголошувати другом клас, якщо можна оголосити другом тільки функцію.

- C++ дозволяє обмежити дружній доступ певним набором функцій.

Приватні (`private`) елементи дозволяють вам захищати класи і зменшити ймовірність помилок. Таким чином, ви повинні обмежити використання класів-друзів настільки, наскільки це можливо. Іноді програма безпосередньо може змінити значення елементів класу, це збільшує ймовірність появи помилок.

Визначення друзів класу

Наприклад, наведений нижче клас `book` оголошує клас `librarian` своїм другом. Тому об'єкти класу `librarian` можуть безпосередньо звертатися до приватних елементів класу `book`, використовуючи оператор точку:

```
class librarian
{
public:
    void change_catalog(book *, char *);
};
void librarian::change_catalog(book *this_book, char *new_catalog)
{
    strcpy(this_book->catalog, new_catalog);
}
```

```

class book
{
public:
    book (char *, char *, char *);
    void show_book (void);
    friend librarian;
private:
    char title [64];
    char author [64];
    char catalog [64];
};

```

Щоб вказати друга, необхідний тільки один оператор `friend librarian;` в оголошенні класу. Отже, функції класу `librarian` можуть безпосередньо звертатися до приватних елементів класу `book`. У даному випадку програма використовує функцію `change_catalog` класу `librarian` для зміни номера картки каталогу певної книги.

Програма передає об'єкт `book` у функцію `change_catalog` класу `librarian` за адресою. Оскільки ця функція змінює елемент класу `book`, програма повинна передати параметр за адресою, а потім використовувати покажчик для звернення до елемента цього класу.

Якщо ви оголошите інший клас другом нашому класу, ви забезпечите класу-другу доступ до приватних, захищених та відкритих (тобто, всіх) елементів даних нашого класу. Ви також знаєте і те, що чим більше доступу до приватних даними класу, тим більше шансів на внесення помилок у програму. Отже, якщо доступ до приватних даними іншого класу необхідний тільки кільком функціям класу, C++ дозволяє вказати, що тільки певні функції дружнього класу будуть мати доступ до приватних елементів. Припустимо, наприклад, що клас `librarian`, представлений в попередній програмі, містить багато різних функцій. Однак припустимо, що тільки функції `change_catalog` необхідний доступ до приватних та захищених елементів класу `book`. Всередині визначення класу `book` ми можемо обмежити доступ до приватних елементів лише цією функцією, як показано нижче:

```

class book
{
public:
    book (char *, char *, char *);
    void show_book (void);
    friend void librarian::change_catalog (book *, char *);
private:

```

```
    char title [64];  
    char author [64];  
    char catalog [64];  
};
```

Як бачите, оператори friend містять повні прототипи всіх дружніх функцій, які можуть безпосередньо звертатися до приватних елементів.

І приклад зі сторонньою функцією-другом:

```
void fun_change_catalog(book *this_book, char *new_catalog)  
{  
    strcpy(this_book->catalog, new_catalog);  
}
```

```
class book  
{  
public:  
    book (char *, char *, char *);  
    void show_book (void);  
    friend void fun_change_catalog (book *, char *);  
private:  
    char title [64];  
    char author [64];  
    char catalog [64];  
};
```

Висновки

1. Використання в ваших програмах на C++ друзів дозволяє одному класу звертатися до ВСІХ елементів іншого класу (private, protected, public) безпосередньо, використовуючи оператор доступу до елементу класу.
2. Для оголошення одного класу другом (friend) іншого класу ви повинні у середині визначення цього іншого класу вказати ключове слово friend, за яким слідує ім'я першого класу. Аналогічно з функціями.
3. Після оголошення класу одним по відношенню до іншого класу, всі функції-елементи класу-одного можуть звертатися до всіх елементів цього іншого класу.
4. Щоб обмежити кількість дружніх методів, які можуть звертатися до приватних даними класу, C++ дозволяє вказати дружні функції. Для оголошення функції-друга ви повинні вказати ключове слово friend, за яким слід прототип функції, якій, власне, і необхідно звертатися до приватних елементів класу.

5. При оголошенні дружніх функцій ви можете отримати синтаксичні помилки, якщо невірний порядок визначень класів. Якщо необхідно повідомити компілятору, що ідентифікатор представляє ім'я класу, який програма визначить пізніше, ви можете використовувати оператор такого виду `class class_name;`.

Висновки

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на лекції

1. Що таке дружні функції класу?
2. Для чого використовуються дружні функції класу?
3. Для чого використовуються дружні класи?
4. Як описуються дружні класи?
5. Яка різниця між дружнім класом та дружньою функцією?

Перевантаження операторів

Актуальність і основна ціль теми:

- оволодіти поняттями перевантаження.

Студент повинен знати:

- для чого призначено перевантаження операторів.

Студент повинен уміти:

- перевантажувати оператори для нових типів даних – класів .

Основні питання:

1. Поняття перевантаження.
2. Поняття перевантаження унарних та бінарних операцій.
3. Перевантаження операцій методами класу.

Методичні рекомендації

При вивченні запропонованої теми студенту варто звернути увагу на такі основні положення:

по-перше: перевантаження операторів та функцій – це вид поліморфізму;

по-друге: перевантаження унарних та бінарних операцій здійснюється по-різному.

Рекомендована література:

1. Довбуш, Г. Ф. Visual C++ на прикладах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

2. Павловская, Т. А. С/C++. Программирование на языке высокого уровня [Текст] / Т. А. Павловская. – М. - СПб. : ПИТЕР, 2003. - 461 с.

Перевантаження операторів

Перевантаження операторів надає можливість перевантажити більшість стандартних операторів C++ щодо типу даних `class`.

Коли оператор перевантажується, його стандартне призначення для типів даних, відмінних від типу `class`, зберігається. У той же час оператор набуває додаткового змісту, пов'язаного з тим класом, для якого він визначений. Отже, перевантаження оператора розширює набір типів, для яких може застосовуватися.

Перевантаження операторів здійснюється за допомогою операторних функцій (`operator function`), які можуть бути функціями-членами (належати класу) чи функціями-друзями (не належати класу). У кожному випадку операторна функція оголошується по-своєму.

Для здійснення перевантаження оператора використовується функція типу `operator імя_оператора`. Тут параметр `імя_оператора` визначає назву перевантажуваного оператора.

В процесі перевантаження не допускається зміна асоціативності і пріоритету операторів.

У C++ дозволено перевантажувати вбудовані операції - це один із проявів поліморфізму. Наявність перевантаження операцій в C++ забезпечує додатковий рівень зручності при використанні нових типів даних.

При програмуванні нового типу даних бажано оголошувати об'єкти нового типу аналогічно вбудованим - форми ініціалізації не повинні відрізнятися від форм ініціалізації об'єктів вбудованого типу. Ці можливості забезпечують конструктори.

При перевантаженні операції потрібно враховувати такі обмеження.

Операції можна перевантажувати тільки для нового типу даних - не можна перенавантажувати операцію для вбудованого типу. У C++ новий тип даних можна оголосити за допомогою, наприклад, класу.

Не можна змінити пріоритет перевантажуваної операції і кількість операндів. Унарна операція зобов'язана мати один операнд, бінарна - два операнди, не дозволяється використовувати параметри за замовчуванням.

Операції можна перевантажувати як зовнішні функції або як методи класу.

Деякі операції допускається перевантажувати тільки як методи класу.

Якщо операція перевантажується як метод класу, то лівим (або єдиним) аргументом обов'язково буде об'єкт класу, для якого перевантажується операція.

Прототип функції-операції виглядає наступним чином:

тип оператор @ (список параметрів);

де @ - символ операції. Слово operator є зарезервованим словом.

Перевантаження операцій інкремента

Операція інкремента має дві форми: префіксну і постфіксну. Для першої форми спочатку змінюється стан об'єкта відповідно до даної операції, а потім він (об'єкт) використовується в тому чи іншому виразі. Для другої форми об'єкт використовується в тому стані, який у нього було до початку операції, а потім вже його стан змінюється.

Щоб компілятор зміг розрізнити ці дві форми операції інкремента, для них використовуються різні сигнатури, наприклад:

```
Point& operator ++():    // префіксний інкремент
```

```
Point operator ++(int);    // постфіксний інкремент
```

Покажемо реалізацію даних операцій на прикладі класу Point:

```
Point& Point::operator ++() {
```

```
    x++;    y++;
```

```
    return *this;
```

```
}
```

```
Point Point::operator ++(int) {
```

```
    Point old = *this;
```

```
    x++;    y++;
```

```
    return old;
```

```
}
```

Зверніть увагу, що в префіксній операції здійснюється повернення результату за посиланням. Це запобігає виклику конструктора копіювання для створення значення, що повертається і наступного виклику деструктора. У постфіксної операції інкремента повернення за посиланням не підходить, оскільки необхідно повернути первісний стан

об'єкта, збережене в локальній змінній `old`. Таким чином, префіксний інкремент є більш ефективною операцією, ніж постфіксний інкремент.

Перевантаження операції присвоювання

Про перевантаження цієї операції слід поговорити окремо з кількох причин.

По-перше, якщо ви не визначите цю операцію в деякому класі, то компілятор створить операцію присвоювання за замовчуванням, яка виконує поелементне копіювання об'єкта. У цьому випадку можлива поява тих же проблем, які виникають при використанні конструктора копіювання за замовчуванням. Тому якщо в класі треба визначити конструктор копіювання, то повинна бути перевантажена операція присвоювання, і навпаки. По-друге, операція присвоювання може бути визначена тільки в формі методу класу.

Зверніть увагу на кілька простих, але важливих моментів при реалізації операції присвоювання:

1. переконайтеся, що не виконується присвоювання виду $x = x$. Якщо ліва і права частини посилаються на один і той самий об'єкт, то робити нічого не треба;
2. видаліть попередні значення полів в динамічно виділеній пам'яті;
3. виділіть пам'ять під нові значення полів;
4. копіюйте в неї нові значення всіх полів;
5. поверніть значення об'єкта, на яке вказує `this` (тобто `* this`).

Питання для самоперевірки і контролю:

4. Що таке перевантаження операцій?
5. Які операції можуть перевантажуватися?
6. Як використається перевантаження операцій?

Тема Операторні функції-члени класу. Операторні функції-друзі класу

Актуальність і основна ціль теми:

- оволодіти поняттями перевантаження.

Студент повинен знати:

- для чого призначено перевантаження операторів.

Студент повинен уміти:

- перевантажувати оператори для нових типів даних – класів .

Основні питання:

1. Поняття перевантаження.
2. Поняття перевантаження унарних та бінарних операцій.
3. Перевантаження операцій методами класу.

Методичні рекомендації

При вивченні запропонованої теми студенту варто звернути увагу на такі основні положення:

по-перше: перевантаження операторів та функцій – це вид поліморфізму;

по-друге: перевантаження унарних та бінарних операцій здійснюється по-різному.

Рекомендована література:

1. Довбуш, Г. Ф. Visual C++ на прикладах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.
2. Павловская, Т. А. С/С++. Программирование на языке высокого уровня [Текст] / Т. А. Павловская. – М. - СПб. : ПИТЕР, 2003. - 461 с.

Перевантаження операторів

Операторна функція або функція-операція перевантажується в тих випадках, коли хоча б одним із операндів є об'єкт класу.

Перевантажити операторну функцію можна двома способами:

- методом класу;
- сторонньою функцією, при необхідності дружньою.

Оголошення дружньої функції в класі має наступний формат:

```
friend прототип_функції-друга;
```

Якщо функція-друг є методом іншого класу, то для неї слід вказати область видимості (ім'я класу, в якому вона визначена) за форматом:

```
friend ім'я_класу:: прототип_функції-друга;
```

Під операторною функцією-другом розуміють операторну функцію, яка перевантажує оператор для класу за допомогою дружньої функції.

Стороння (глобальна) операторна функція-друг для перевантаження унарних операторів має наступний формат:

```
тип_функції operator знак_оператора (параметр1) { /*операції*/ }
```

Стороння (глобальна) операторна функція-друг для перевантаження бінарних операторів має наступний формат:

```
тип_функції operator знак_оператора (параметр1, параметр2)  
{ /*операції*/ }
```

Тобто, стороння операторна функція в загальному випадку має стільки параметрів, скільки операндів має операція.

Список параметрів функцій-друзів не може бути порожнім через те, що `this` цим функціям не передається, як це є в операторних функціях-членах класу.

Операторна функція-метод класу для перевантаження унарних операторів має наступний формат:

```
тип_функції operator знак_оператора () { /*операції*/ }
```

Операторна функція-метод класу для перевантаження бінарних операторів має наступний формат:

```
тип_функції operator знак_оператора (параметр1)
{ /*операції*/ }
```

Тобто, операторна функція-метод класу в загальному випадку має на один параметр менше, ніж кількість операндів, які має операція. На один менше тому, що в якості лівого операнда операції виступає `this`.

Тобто, методом класу перевантажують операції, да лівий операнд — об'єкт класу.

Наприклад:

```
class D
{   int* m_p ; int m_n ;
public:
    D ( int = 10 ) ;
    D ( D& ) ;
    ~ D ( ) ;
    int getN ( void ) const ;
    int& operator [ ] ( const unsigned int ) const ; //перевантаження
методом класу операції індексації
    D& operator = ( D& ) ; //перевантаження методом класу операції
присвоювання
    D operator + ( const D& ) const ; //перевантаження методом класу
складання об'єкта з об'єктом
    D operator + ( const int& ) const ; //перевантаження методом класу
складання об'єкта з цілим числом
    D operator - ( const D& ) const ; //перевантаження методом класу
віднімання об'єкта від об'єкта
    friend void view ( const D& ) ;
} ;
```

Приклад виклику цих операцій з головної функції:

```
int main(int argc, char* argv[])
{
    D o1 ( 5 ), o2 ( 5 ) ;
    for ( int i = 0; i < o1.getN ( ); i++ )
    {   o1 [ i ] = ( i + 1 ) * 2 ; o2 [ i ] = i ; } //виклик операторної
функції індексації
    D res ;
    res = o1 + o2 ; //виклик операторної функції складання об'єкта з об'єктом і
присвоювання
    int x ( 5 ) ; D o3 ( 5 ) ;
    res = o3 + x ; //виклик операторної функції складання об'єкта з цілим
числом
    res = o3 - o2 ; //виклик операторної функції віднімання об'єктів
    return 0 ;
}
```

Перевантаження операції присвоювання.

Про перевантаження цієї операції слід поговорити окремо з кількох причин.

По-перше, якщо ви не визначите цю операцію в деякому класі, то компілятор створить операцію присвоювання за замовчуванням, яка виконує поелементно копіювання об'єкта. У цьому випадку можлива поява тих же проблем, які виникають при використанні конструктора копіювання за замовчуванням (див. вище). Тому якщо в класі потрібно визначити конструктор копіювання, то повинна бути перевантажена операція присвоювання, і навпаки.

По-друге, операція присвоювання може бути визначена тільки в формі методу класу.

По-третє, операція присвоювання не успадковується (на відміну від всіх інших операцій).

Зверніть увагу на кілька простих, але важливих моментів при реалізації операції привласнення:

1. переконайтеся, що не виконується присвоювання виду $x = x$. Якщо ліва і права частини посилаються на один і той самий об'єкт, то робити нічого не треба. Якщо не перехопити цей особливий випадок, то наступний крок знищить значення, на яке вказує `pName`, ще до того, як воно буде скопійовано;

2. видаліть попередні значення полів в динамічно виділеної пам'яті;

3. виділіть пам'ять під нові значення полів;

4. копіюйте в неї нові значення всіх полів;

5. верніть значення об'єкта, на яке вказує `this` (тобто `* this`).

Приклад для оголошеного вище класу:

```
D& D :: operator = ( D& right )
{
    if ( m_n != right.m_n )
    { delete [ ] m_p ; m_n = right.m_n ; m_p = new int [ m_n ] ; }
    for ( int i = 0; i < m_n; i++ ) m_p [ i ] = right.m_p [ i ] ;
    return *this ;
}
```

Питання для самоперевірки і контролю:

7. Що таке перевантаження операцій?

8. Як перевантажити операцію методом класу, які передбачити параметри і чи для всіх випадків можливо таке перевантаження?
9. Як перевантажити операцію сторонньою функцією, в яких випадках її треба оголосити дружньою і які параметри передавати?

Література

Довбуш, Г. Ф. Visual C++ на прикладах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

ВСТУП

Статичні елементи – поля класу ініціалізуються за межами класу та головної функції проекту.

Статичні елементи – методи класу використовуються для роботи з статичними даними класу.

1 Статичні елементи класу**Статичні дані класу**

Іноді буває корисно виділити загальну пам'ять для всіх об'єктів одного класу. Щоб отримати таку пам'ять, слід оголосити член-дане зі специфікатором `static`. Ключове слово `static` поміщається перед типом того даного, яке оголошується статичним. Щоб статичні дані класу стали доступні за межами класу, необхідно призначити їм відкритий специфікатор доступу `public`. Якщо ж дані оголосити закритими, то будуть потрібні відкриті методи для отримання їх значень.

Перелічимо основні властивості статичних даних класу:

1. Статичні дані не є частиною об'єктів класу і зберігаються за його межами. Пам'ять для статичної змінної-члена при оголошенні об'єкту класу не виділяється.
2. Одна копія використовується всіма об'єктами класу, отже, всі об'єкти мають одне і те ж значення статичного даного.
3. Оголошення статичних даних в класі не є їх описом, тому присвоєння їм значень проводиться поза класом і поза функцій програми
4. Статичні дані існують навіть при відсутності об'єктів класу, для якого вони оголошені як статичні.
5. Доступ до статичної змінної-члена зі специфікатором `public` може бути реалізований без об'єкта.

Статичні методи класу

Статичними можуть бути не тільки дані класу, а й методи. Ключове слово `static` ставиться перед типом функції-члена. Статичні методи в основному використовуються для роботи зі статичними членами-даними класу.

Існують наступні обмеження на статичні методи:

1. Методи мають прямий доступ лише до статичних даних класу.
2. Показчик `this` не видно в статичних методах, тому в їх визначенні не можна звернутися до нестатичних даних класу.
3. Неможливо оголосити в класі статичну і нестатичні версії функції.
4. Статичні методи не можуть бути віртуальними і не можуть бути константними.

До цього моменту кожен створюваний вами об'єкт мав свій власний набір елементів даних. Залежно від призначення вашої програми можуть бути ситуації, коли об'єкти одного і того ж класу повинні спільно використовувати один або кілька елементів даних. Наприклад, припустимо, що ви пишете програму платежів, яка відстежує робочий час для 1000 службовців. Для визначення податкової ставки програма повинна знати умови, в яких працює кожен службовець. Нехай для цього використовується змінна класу `state_of_work`. Однак, якщо всі службовці працюють в однакових умовах, ваша програма могла б спільно використовувати цей елемент даних для всіх об'єктів типу `employee`. Таким чином, ваша програма зменшує необхідну кількість пам'яті, викидаючи 999 копій однакової інформації. Для спільного використання елемента класу ви повинні оголосити цей елемент як `static` (статичний). Розглянемо кроки, які повинні виконати для спільного використання елемента класу декількома об'єктами:

- С++ дозволяє мати об'єкти одного і того ж типу, які спільно використовують один або кілька елементів класу.
- Якщо ваша програма присвоює значення спільно використовуваного елемента, то всі об'єкти цього класу відразу ж отримують доступ до цього нового значення.
- Для створення спільно використовуваного елемента даних класу ви повинні випереджати ім'я елемента класу ключовим словом `static`.
- Після того як програма оголосила елемент класу як `static`, вона повинна оголосити глобальну змінну (поза визначення класу), яка відповідає цьому спільно використовуваного елемента класу.
- Ваші програми можуть використовувати ключове слово `static`, щоб зробити метод класу викликаються в той час, як програма, можливо, ще не оголосила жодних об'єктів даного класу.

СПІЛЬНЕ ВИКОРИСТАННЯ ЕЛЕМЕНТА ДАНИХ

Коли створюються об'єкти певного класу, кожен об'єкт отримує свій власний набір елементів даних. Однак можливі такі ситуації, при яких об'єктах одного і того ж класу необхідно спільно використовувати один або кілька елементів даних (статичні елементи даних). У таких випадках поля оголошуються статичними, як показано нижче:

`private:`

```
static int shared_value;
```

Після оголошення класу слід оголосити елемент як глобальну змінну поза класом (можна над головною функцією), як показано нижче:

```
int class_name:: shared_value;
```

Наступна програма SHARE_IT.CPP визначає клас book_series, спільно використовує елемент page_count, який є однаковим для всіх об'єктів (книг) класу (серії). Якщо програма змінює значення цього елементу, зміна відразу ж виявляється в усіх об'єктах класу:

```
#include <iostream.h>
#include <string.h>
class book_series
{
public:
    book_series(char *, char *, float);
    void show_book(void);
    void set_pages(int) ;
private:
    static int page_count;
    char title[64];
    char author[ 64 ];
    float price;
};
int book_series::page__count;
void book_series::set_pages(int pages)
{
    page_count = pages;
}
book_series::book_series(char *title, char *author, float price)
{
    strcpy(book_series::title, title);
    strcpy(book_series::author, author);
    book_series::price = price;
}
void book_series:: show_book (void)
{
    cout << "Заголовок: " << title << endl;
    cout << "Автор: " << author << endl;
    cout << "Цена: " << price << endl;
    cout << "Страницы: " << page_count << endl;
}
void main(void)
{
    book_series programming( "Учимся программировать на C++", "Jamsa",
22.95);
    book_series word( "Учимся работать с Word для Windows", "Wyatt",
19.95);
    word.set_pages(256);
    programming.show_book ();
    word.show_book();
    cout << endl << "Изменение page_count " << endl;
    programming.set_pages(512);
    programming.show_book();
```

```
    word.show_book();  
}
```

Як бачите, клас оголошує `page_count` як `static int`. Відразу ж за визначенням класу програма оголошує елемент `page_count` як глобальну змінну. Коли програма змінює елемент `page_count`, зміна відразу ж виявляється в усіх об'єктах класу `book_series`.

При оголошенні елемента класу як `static` цей елемент спільно використовується всіма об'єктами даного класу. Проте можливі ситуації, коли програма ще не створила об'єкт, але їй необхідно використовувати елемент. Наприклад, наступна програма `USE_MBR.CPP` використовує елемент `page_count` з класу `book_series`, навіть якщо об'єкти цього класу не існують:

```
#include <iostream.h>  
#include <string.h>  
class book_series  
{  
public:  
    static int page_count;  
private:  
    char title [64];  
    char author[64];  
    float price;  
};  
int book_series::page_count;  
void main(void)  
{  
    book_series::page_count = 256;  
    cout << "Поточне значення page_count дорівнює " <<  
book_series::page_count << endl;  
}
```

В даному випадку, оскільки клас визначає елемент класу `page_count` як `public`, програма може звернутися до цього елемента класу, навіть якщо об'єкти класу `book_series` не існують.

ВИКОРИСТАННЯ СТАТИЧНИХ ФУНКЦІЙ - МЕТОДІВ

Попередня програма ілюструвала використання статичних елементів даних. Подібним чином C++ дозволяє вам визначити статичні функції- елементи (методи). Якщо ви створюєте статичний метод, ваша програма може викликати такий метод, навіть якщо об'єкти не були створені. Наприклад, якщо клас містить метод, який може бути використаний для даних поза класом, ви могли б зробити цей метод статичним.

Таким чином:

1. Коли ви оголошуєте елемент класу як `static`, то такий елемент може спільно використовуватися всіма об'єктами даного класу.
2. Після того як ваша програма оголошує елемент класу як `static`, слід поза межами класу оголосити глобальну змінну, відповідну спільно використовуваного елементу класу.

3. Якщо ви оголошуєте елемент як `public static i`, ваша програма може використовувати такий елемент, навіть якщо об'єкти даного класу не існують. Для звернення до цього елемента ваша програма повинна використовувати оператор глобального дозволу, наприклад `class_name:: member_name`.
4. Якщо ви оголошуєте загальну статичну функцію-елемент, ваша програма може викликати цю функцію, навіть якщо об'єкти даного класу не існують. Для виклику даної функції програма повинна використовувати оператор глобального дозволу, наприклад `menu:: clear_screen ()`.

Висновки

Іноді буває корисно виділити загальну пам'ять для всіх об'єктів одного класу. Щоб отримати таку пам'ять, слід оголосити член-дане зі специфікатором `static`. Ключове слово `static` поміщається перед типом того даного, яке оголошується статичним. Щоб статичні дані класу стали доступні за межами класу, необхідно призначити їм відкритий специфікатор доступу `public`. Якщо ж дані оголосити закритими, то будуть потрібні відкриті методи для отримання їх значень.

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на лекції

1. Для чого використовуються статичні дані класу?
2. Для чого використовуються статичні методи класу?
3. Як описуються і ініціалізуються статичні дані класу?
4. Як описуються і викликаються статичні методи класу?

Література

Довбуш, Г. Ф. Visual C++ на прикладах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

ВСТУП

Шаблони функцій представляють собою функцію.

Шаблони дають можливість багаторазово використати один і той же програмний код.

1 Шаблони функцій

Деякі перевантажені функції мають один і той самий код, незалежно від типу параметрів, наприклад, перестановка елементів або копіювання вмісту одного масиву в інший. У мові C++ існує можливість замість кількох ідентичних за операціями функцій створювати тільки одну універсальну функцію, яка називається шаблоном або шаблонною функцією.

Шаблонна функція (template function) - це функція, яка повністю контролює відповідність типів даних, які задаються їй як параметри.

Оголошення шаблону функції повторює визначення звичайної функції, але першим рядком в оголошенні обов'язково має бути рядок наступного вигляду:

```
template <class ім'я_типу1, ..., class ім'я_типу N>
```

У кутових дужках <> після ключового слова template вказується список формальних параметрів шаблону. Ім'я_типу називають параметром типу, котрий є ідентифікатор типу і використовується для визначення типу параметра в заголовку шаблонної функції, типу повертаемого функцією значення та типів змінних, які декларуються в тілі функції. Кожний параметр типу повинен мати унікальний ідентифікатор, який може бути використаний в різних шаблонах. Кожному параметру типу має передувати ключове слово class або typename.

За рядком `template` слідує рядок зі стандартним оголошенням функції, до в списку параметрів використовуються як параметри типу, так і будь-які інші допустимі базові або похідні типи даних.

Шаблонна функція може бути перевантажена іншим шаблоном або нешаблонною функцією з таким же ім'ям, але з іншим набором параметрів (лістинг).

Компілятор вибирає варіант функції, відповідний виклику. Спочатку вибирається функція, яка повністю відповідає по імені і типам параметрів тій, що викликається. Якщо спроба закінчується невдачею, вибирається шаблон, за допомогою якого можна згенерувати функцію з відповідністю типів всіх параметрів та імені. Якщо ж і ця спроба невдала, виконується підбір перевантаженої функції.

```
#include <iostream>
using namespace std ;
// шаблонна функція
template < class T > void view ( const T *p, const size_t size )
{
    for ( int i = 0; i < size; i++ )
        cout << p [ i ] << '\t' ;
    cout << endl ;
}
// головна функція
int main ( )
{
    int ia [ ]= { 1, -3, 0, -9, 2, 7, 0, -19 } ;
    double da [ ] = { 0.0875, 1.25, -3, 0.0, -7.986 } ;
    long la [ ] = { 1L, -2L, 3L, -100L } ;
    float fa [ ] = { -2.12, 1.5, -3.25, -17.2 } ;
    char s [ ] = "Vivat!" ;
    cout << "\nint Array\n" ;
    view ( ia, sizeof ( ia ) / sizeof ( int ) ) ;
    cout << "\ndouble Array\n" ;
    view ( da, sizeof ( da ) / sizeof ( double ) ) ;
    cout << "\nlong Array\n" ;
    view ( la, sizeof ( la ) / sizeof ( long ) ) ;
    cout << "\nchar Array\n" ;
    view ( s, strlen ( s ) ) ;
    return 0 ;
}
```

Тут продемонстровані результати роботи шаблону для виведення масиву . Як параметр типу шаблонна функція використовує ідентифікатор `T`, який застосовується для покажчика `p` в списку параметрів шаблонної функції. Покажчик визначає адресу масиву, елементи якого необхідно вивести. Другий параметр має явний тип `size_t` . Функція не повертає значення і має тип `void` . Для кожного масиву генерується функція з відповідним типом. Замість параметра типу `t` підставляються значення `int` для масиву `ia` , `double` для

масиву da , long для масиву la і char для масиву s. У кожному виклику функції view () вказується відповідно ім'я масиву, а також розмір масиву, який обчислюється за допомогою оператора sizeof ().

```
#include <iostream>
using namespace std ;
// шаблонна функція
template < class T > T abs ( const T n ) { return n < 0 ? -n : n ; }
// головна функція
int main ( )
{
    int i ( -5 ) ; float f ( -1.125e-5 ) ; long l ( 123L ) ;
    cout << abs ( i ) << endl ;           // виводиться 5
    cout << i << endl ;                   // виводиться -5
    cout << abs ( f ) << endl ;           // виводиться 1.125e-005
    cout << abs ( l ) << endl ;           // виводиться 123
    cout << abs ( -123L ) << endl ; // виводиться 123
    return 0 ;
}
```

Шаблонна функція abs () в приведеному вище лістингу змінює знак тільки у негативних чисел будь-якого типу і повертає позитивне значення. Параметр типу T визначає як тип вхідного параметра n , так і тип значення, яке повертається функцією. Вхідний параметр вказаний зі специфікатором const , тому в тілі функції значення n не змінюється. Це означає, що фактичний параметр збереже своє значення після виконання функції. Наприклад, якщо буде потрібно змінити значення змінної i, в головній функції слід скористатися оператором присвоювання i = abs (i).

```
template < class T > void swap ( T *a, T *b )
{
    T tmp ( *a ) ; *a = *b ; *b = tmp ;
}
#include <iostream>
using namespace std ;
int main ( )
{
    int unsigned short x ( 150 ), y ( 15 ) ;
    double a ( 0 ), b ( -25.4 ) ;
    char t ( 'a' ), e ( 'z' ) ;
    cout << x << '\t' << y << endl ;      // виводиться 150      15
    swap ( x, y ) ;
    cout << x << '\t' << y << endl ;      // виводиться 15 150
    cout << a << '\t' << b << endl ;      // виводиться 0      -25.4
    swap ( a, b ) ;
    cout << a << '\t' << b << endl ;      // виводиться -25.4  0
    cout << t << '\t' << e << endl ;      // виводиться a      z
    swap ( t, e ) ;
    cout << t << '\t' << e << endl ;      // виводиться z      a
    return 0 ;
}
```

Шаблон `reorg ()` дозволяє змінювати порядок проходження елементів в масиві будь-якого числового типу так, що на початку розташовуються негативні елементи, після яких записуються позитивні. Важливо пам'ятати, що покажчик `p` на вихідний масив не можна оголошувати з специфікатором `const`, тому що зміни здійснюються безпосередньо у вихідному масиві. Функція використовує єдиний параметр типу `T`, застосовуваний в заголовку і в тілі шаблону. Головна функція перевіряє роботу шаблону масиву довгого цілого типу `long` і масиву чисел з плаваючою точкою `float`.

```
// в начало массива записываются отрицательные элементы
// после отрицательных элементов записываются положительные элементы
template < class T > void reorg ( T *p, const int size )
{
    T tmp ;
    for ( int i = 0, j = 0 ; i < size ; i++ )
    {
        if ( p [ i ] > 0 )    continue ;
        tmp = p [ i ] ;
        for ( int k = i ; k > j ; k-- ) p [ k ] = p [ k - 1 ] ;
        p [ j ] = tmp ;
        j++ ;
    }
}
```

Висновки

Шаблонна функція може бути перевантажена іншим шаблоном або не шаблонною функцією з таким же ім'ям, але з іншим набором параметрів.

Компілятор вибирає варіант функції, відповідний виклику. Спочатку вибирається функція, яка повністю відповідає по імені і типами параметрів викликається. Якщо спроба закінчується невдачею, вибирається шаблон, за допомогою якого можна згенерувати функцію з відповідністю типів всіх параметрів та імені. Якщо ж і ця спроба невдала, виконується підбір перевантажених функцій.

Питання та завдання до контролю знань студентів

1 Для актуалізації опорних знань та перевірки попереднього матеріалу

1. Що таке статичний елемент класу?
2. Для чого використовуються статичні дані класу?
3. Для чого використовуються статичні методи класу?
4. Як описуються статичні дані класу?

5. Як описуються статичні методи класу?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

1. Що таке шаблон функцій?

2. Як описується шаблон функцій?

Тема**Константні члени класу. Константні методи****Мета заняття**

Ознайомитись з поняттям константних членів класу

Література

1 Глушаков, С.В. Язык программирования C++. [Текст] / С.В. Глушаков, А.В.Коваль, С.В.Смирнов. - Х.:Фолио, 2001. - 500 с.

2 Павловская, Т. А. C/C++. Программирование на языке высокого уровня [Текст] / Т. А. Павловская. – М. - СПб. : ПИТЕР, 2003. - 461 с.

Правильне використання `const` дозволяє організувати багато перевірок ще на етапі компіляції та уникнути багатьох помилок, які буває важко знайти за допомогою відладки та аналізу кода.

1 Константні члени класу

Константними можуть бути як члени-дані класу, так і члени функції. Розглянемо їх більш детально, а також приклад використання константних об'єктів.

Поля класу:

- можуть мати будь-який тип, крім типу цього ж класу (але можуть бути покажчиками або посиланнями на цей клас);
- можуть бути описані з модифікатором `const`, при цьому вони ініціалізуються тільки один раз (за допомогою конструктора) і не можуть змінюватися.

2 Константні методи

Константний метод:

- оголошується з ключовим словом `const` після списку параметрів;
- не може змінювати значення полів класу;
- може викликати тільки константні методи;
- може викликатися для будь-яких об'єктів (не тільки константних об'єктів).

Рекомендується описувати як константні ті методи, які призначені для отримання значень полів.

Можна створити константний об'єкт, значення полів якого змінювати заборонено. До нього можуть застосовуватися тільки константні методи. Приклад наведений в лістингу нижче.

Лістинг – Приклад константного об'єкту.

```
class monstr {  
    ...  
    int get_health() const { return health; }  
};  
...  
const monstr Dead(0,0); //константний об'єкт  
cout<<Dead.get_health();
```

Отже, функція класу в C++ може бути оголошена з специфікатором `const`, який записується після списку параметрів перед тілом функції. Подібні функції-члени можуть тільки читати значення даних класу, але їм не дозволяється змінювати ці значення. Специфікатор `const` записується як в прототипі, так і при визначенні методу.

Використання константних методів забороняє виконання довільного збільшення даних класу. Перелічимо деякі властивості цих функцій:

- Метод з специфікатором `const` можна викликати для будь-яких об'єктів.
- Для об'єкта, що є константою, можна викликати константних метод, але не можна викликати метод, оголошений без специфікатора `const`.
- Константний метод не може змінювати значення членів-даних об'єкта.

Лістинг 10.2 – Специфікація класу CBook.

```
class CBook  
{  
private:  
    char m_author [ 50 ] ;    // автор  
    char *m_pTitle ;         // вказівник на назву  
    int m_year ;              // рік видання  
public:  
    CBook () ;  
    virtual ~ CBook () ;  
    // методи встановлення значень полів  
    void setAuthor ( const char* ) ;
```

```
void setTitle ( const char* ) ;  
void setYear ( const int ) ;  
// методи повернення значень полів  
char* getAuthor ( void ) const;  
char* getTitle ( void ) const;  
int getYear ( void ) const;  
}
```

Отже, константні методи доцільно використовувати, наприклад, для повернення значень.

Висновки

Константними можуть бути як члени-дані класу, так і члени функції.

Константні поля класу описуються з модифікатором `const`, при цьому вони ініціалізуються тільки один раз (за допомогою конструктора) і не можуть змінюватися.

Константні методи оголошуються з ключовим словом `const` після списку параметрів і не можуть змінювати значення полів класу.

Питання та завдання до контролю знань студентів

1 Для актуалізації опорних знань та перевірки попереднього матеріалу

- 1 Які є види конструкторів?
- 2 Як визначити, який конструктор працює під час створення вказаного об'єкта?
- 3 Для чого використовується специфікатор `explicit`?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Які члени класу можуть бути константними і як вони описуються?
- 2 Як ініціалізуються константні поля класу, чи можуть вони змінюватися?
- 3 Як оголошуються і для чого використовуються константні методи класу?
- 4 Чи можна створити константний об'єкт класу?

Перевантаження операцій введення-виведення

Тема Перевантаження операцій введення-виведення

Мета заняття *Навчитися перевантажувати операції введення-виведення*

Стандартний потік введення-виведення пов'язаний з класами `istream`, `ostream`.

Можна перевантажувати операції введення-виведення, якщо необхідно передбачити виконання цих операцій для об'єкта класу.

Операції введення-виведення вважаються бінарними, тобто мають два операнди, перший з яких характеризує стандартний потік введення або виведення відповідно. Ці потоки описуються класами `istream` та `ostream`.

Перевантаження виведення для класів

Потік виведення `ostream` є механізмом мови для перетворення значень різного типу в послідовність символів і є спеціалізацією універсального шаблону `basic::ostream`. Цей шаблон представлений в заголовку `<iostream>` і визначений в просторі імен `std`.

У `<iostream>` оголошений стандартний символний потік виведення `cout` і визначений оператор `<<` для управління виводу вбудованих типів даних.

Оператор `<<` для типів `class` можна перевизначити так, що цим оператором можна користуватися так само, як і для вбудованих типів, як наведено в лістингу 1.

Лістинг 1 – Приклад перевантаження оператора виведення

```
#include <iostream>
using namespace std;
class CPoint {
    int m_x, m_y ;
public:
    CPoint ( ) : m_x ( 0 ), m_y ( 0 ) { }
    CPoint ( int x, int y ) : m_x ( x ), m_y ( y ) { }
    ~CPoint ( ) { }
```

```

    int getX ( ) { return m_x ; }
    int getY ( ) { return m_y ; }
};

ostream& operator << ( ostream& str, CPoint p ) {
    return ( str << p.getX() << '\t' << p.getY() << endl );
}

int main ( ) {
    CPoint a ( 5, 7 );
    cout << a ;
    return 0 ;
} // буде виведено 5 7

```

Перевантаження введення для класів

Стандартний потік введення `cin` визначається класом `istream` універсального шаблону `basic::istream`, який представлений в заголовку `<iostream>` простору імен `std`. Клас `istream` забезпечує оператор введення `>>` для вбудованих типів даних.

Оператор `>>` для типів `class` можна перевизначити подібно до того, як це було наведено для оператора виведення. Але для введення суттєво, щоб параметр типу `class` був переданий за посиланням, як наведено в лістингу 2.

Лістинг 1 – Приклад перевантаження оператора введення

```

#include <iostream>
using namespace std;
class CPoint {
    int m_x, m_y ;
public:
    CPoint ( ) : m_x ( 0 ), m_y ( 0 ) { }
    CPoint ( int x, int y ) : m_x ( x ), m_y ( y ) { }

```

```

~CPoint ( ) { }

int getX ( ) { return m_x ; }

int getY ( ) { return m_y ; }

};

istream& operator >> ( istream& str, CPoint& p ) {

    int x(0), y(0);

    str >> x >> y;

    p = CPoint(x, y);

    return str;

}

int main ( ) {

    CPoint a;

    cout >> a ;

    cout << a.getX() << '\t' << a.getY() << endl ;

    return 0 ;

}

```

Реалізація перевантаження операцій в загальному випадку може бути виконана операторною функцією – методом класу або сторонньою функцією.

У випадку перевантаження операцій введення-виведення операторна функція має бути сторонньою, тому що для бінарної операції виведення лівий операнд має тип `ostream`, а для бінарної операції введення лівий операнд має тип `istream`.

В тому ж випадку, якщо оператор перевантажується методом класу, лівий операнд повинен мати тип клас. Виведення у власний клас замість потоку може бути розглянуто тільки якщо цей клас описує відповідну предметну область.

Таким чином, операторні й функції введення та виведення реалізуються сторонніми функціями. При необхідності вони оголошуються як друзі. Така необхідність може бути, якщо їм необхідно надати доступ до закритих або захищених полів класу.

Висновки

Операції введення-виведення вважаються бінарними, тобто мають два операнди, перший з яких характеризує стандартний потік введення або виведення відповідно.

Ці перевантажені операції реалізуються сторонніми функціями, при необхідності оголошеними як дружні.

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на лекції:

- 1 З якими класами пов'язаний стандартний потік введення-виведення?
- 2 Як перевантажити операції введення-виведення?
- 3 Скільки операндів має операція виведення?

Література

Довбуш, Г. Ф. Visual C++ на примерах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

Тема **Успадкування та повторне використання коду. Просте поодиноке**

успадкування

Мета заняття *Ознайомитись з поняттям успадкування та повторного
використання коду, простим поодиноким успадкуванням*

Література

Довбуш, Г. Ф. Visual C++ на прикладах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под
ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

ВСТУП

Поняття успадкування є одним з базових понять об'єктно-орієнтованого програмування.

Похідний клас при успадкуванні повинен мати свої власні конструктори.

Успадкування та повторне використання коду. Просте поодиноке успадкування

Мета об'єктно-орієнтованого програмування полягає в повторному використанні створених вами класів. Використання успадкування для побудови нових класів заощадить вам багато часу і сили на програмування. Об'єктно-орієнтоване програмування широко використовує спадкування, дозволяючи вашій програмі будувати складні об'єкти з невеликих легко керованих об'єктів.

Якщо ви вже створили певний клас , то можливі ситуації, що нового класу потрібно багато або навіть усі особливості вже існуючого класу, і необхідно додати один або кілька елементів даних або функцій. У таких випадках C++ дозволяє вам будувати новий об'єкт, використовуючи характеристики вже існуючого об'єкта. Іншими словами, новий об'єкт буде успадковувати елементи існуючого класу (званого базовим класом). Коли ви будувате новий клас з існуючого, цей новий клас часто називається похідним класом.

Спадкування у класах

Коли ви визначаєте один клас на базі іншого, то такий клас називається похідним. Похідний клас автоматично отримує дані-члени того класу, який використано як базовий при його визначенні, і, з деякими обмеженнями, також функції-члени цього базового класу. Кажуть, що клас наслідують дані-члени та функції-члени класу, на якому він базується.

Деструктор, конструктори і будь-які функції-члени, перенавантажують операцію присвоювання при успадкуванні не стають відповідно конструкторами і т.ін. похідного класу, тому що в ньому є свої. Всі інші функції-члени, разом з усіма даними-членами базового класу, успадковуються похідним класом як члени класу. Похідний клас завжди має свої власні конструктори і деструктор. Якщо у базового класу є операція присвоювання, похідний клас представляє її власну версію.

Коли говорять, що ці функції не успадковуються, то мають на увазі, що вони не існують як члени об'єкта похідного класу. Однак вони, як і раніше, існують як частина об'єкта, що відноситься до базового класу.

Спадкування класів від базового класу

Повернемося до початкового класу CBox з даними-членами public.

```
// Заголовний файл Box.h в проєкті
# pragma once
class CBox {
public:
    double m_Length;
    double m_Width;
    double m_Height;
    CBox (double lv = 1.0, double wv = 1.0, double hv = 1.0):
        m_Length (lv), m_Width (wv), m_Height (hv) {}
};
```

Директива `# pragma once` гарантує, що визначення класу `sbox` з'явиться в збірці тільки один раз. У класі є конструктор, так що можна ініціалізувати об'єкти при їх оголошенні. Припустимо, що знадобиться інший клас об'єктів - `CCandyBox`, який у всьому схожий на `CBox`, але також має ще один член даних - покажчик на текстовий рядок, що ідентифікує вміст коробки.

Ви можете визначити CCandyBox як похідний клас, що має CBox в якості базового, як показано нижче:

```
// Заголовний файл CandyBox.h в проекті
#pragma once
#include "Box.h"
class CCandyBox: CBox
{
public:
    char* m_Contents;
    CCandyBox(char* str = "Candy") // Конструктор
    {
        m_Contents = new char [ strlen(str) + 1 ]; strcpy_s(m_Contents,
strlen(str) + 1, str);
    }
    ~CCandyBox() // Деструктор
    { delete[] m_Contents; };
};
```

Додайте цей заголовний файл в проект. Вам знадобиться директива `#include` для заголовка `Box.h`, оскільки ви посилаєтеся в цьому коді на клас `CBox`. Якщо цю директиву опустити, компілятору не буде відомий `CBox`, так що код компілюватися не буде. Ім'я базового класу `sbox` з'являється після імені похідного класу `CCandyBox` і відокремлюється від нього двокрапкою. У всіх інших відношеннях він виглядає як нормальне визначення класу. Ви додали новий член `Contents`, і, оскільки це покажчик на рядок, потрібно, щоб конструктор ініціалізував його, а деструктор звільняв пам'ять, зайняту цим рядком. Ви також повинні передбачити в конструкторі значення за замовчуванням для рядка, що описує вміст об'єкта `CCandyBox`. Об'єкти типу класу `CCandyBox` містять всі члени базового класу `sbox` плюс додатковий член даних `m_Contents`.

Управління доступом при спадкуванні

Питання про доступ до успадкованих членам в похідному класі вимагає більш ретельного вивчення. Розглянемо стан `private`-членів базового класу в похідному класі.

Існує вагома причина для вибору в попередньому прикладі версії класу `sbox` з даними-членами `public`. Причина в тому, що хоча `private`-члени базового класу є також

членами похідного класу, вони залишаються private по відношенню до базового класу, тому функції-члени, додані до похідному класу, не можуть залучити до них доступ. Вони можуть бути доступні в похідному класі через функції-члени базового класу, які не знаходяться у розділі private базового класу.

Похідні класи

Похідні класи дають простий, гнучкий і ефективний апарат завдання для класу альтернативного інтерфейсу і визначення класу за допомогою додавання можливостей до вже наявного класу без перепрограмування або перекомпіляції.

За допомогою похідних класів можна також забезпечити загальний інтерфейс для декількох різних класів так, щоб інші частини програми могли працювати з об'єктами цих класів однаковою чиною. При цьому зазвичай в кожен об'єкт зберігається інформація про тип, щоб ці об'єкти могли оброблятися відповідною чиною у ситуаціях, коли їх тип не можна дізнатися під час компіляції.

Основна ідея полягає в тому, що класи, які вже одного разу написані, можуть використовуватися іншими програмістами.

Фізично класи складаються з одного або більше заголовків файлів, що визначають інтерфейс, і одного або більше файлів, визначають реалізацію. Заголовки будуть поміщені туди, звідки користувач може взяти їх копії за допомогою директиви #include. Файли, що визначають реалізацію, звичайно компілюють і поміщають в бібліотеку.

Просте спадкування

Простим називається успадкування, при якому похідний клас має одного батька.

Ключі доступу

При описі класу в його заголовку перераховуються всі класи, є для нього базовими. Можливість звернення до елементів цих класів регулюється за допомогою ключів доступу private, protected і public :

```
class ім'я: [ private | protected | public ] базовий_класс { тіло класу };
```

Різниця між специфікаторами виявляється при спадкуванні, що можна бачити з наведеної таблиці:

Ключ доступу успадкування	Специфікатор у базовому класі	Доступ у похідному класі
private	private protected public	немає private private
protected	private protected public	немає protected protected
public	private protected public	немає protected public

Як видно з таблиці, private елементи базового класу в похідному класі недоступні незалежно від ключа. Звернення до них може здійснюватися лише через методи базового класу.

Елементи protected при спадкуванні з ключем private стають у похідному класі private , в інших випадках права доступу до них не змінюються.

Висновки

Коли ви визначаєте один клас на базі іншого, то такий клас називається похідним. Похідний клас автоматично отримує дані-члени того класу, який використано як базовий при його визначенні, і, з деякими обмеженнями, також функції-члени цього базового класу.

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на лекції

1. Що таке просте успадкування?
2. Що означають ключі доступу.
3. Які елементи базового класу при побудові похідного класу недоступні?
4. Як відбувається повторне використання коду?

Тема**Побудова ієрархії класів****Мета заняття***Навчитись будувати ієрархії класів***Література**

Лафоре, Р. Объектно-ориентированное программирование в С++ [Текст] / Р. Лафоре. – М. - СПб. : ПИТЕР, 2004.- 923 с.

Поняття успадкування є одним з базових понять об'єктно-орієнтованого програмування.

Похідний клас, незважаючи на те, що успадковує елементи батьківського класу, має свої власні конструктори і деструктор.

Побудова ієрархії класів

Механізм успадкування класів дозволяє будувати ієрархії, в яких похідні класи отримують елементи батьківських, або базових, класів і можуть доповнювати їх або змінювати їх властивості. При великій кількості ніяк не зв'язаних класів управляти ними стає неможливим. Спадкування дозволяє впоратися з цією проблемою шляхом упорядкування та ранжування класів, тобто об'єднання спільних для декількох класів властивостей в одному класі і використання його в якості базового.

Класи, що знаходяться ближче до початку ієрархії, об'єднують в собі найбільш загальні риси для всіх нижчих класів. У міру просування вниз по ієрархії класи набувають все більше конкретних рис. Множинне наслідування дозволяє одному класу мати властивості двох і більше батьківських класів.

Похідний клас, в свою чергу, сам може служити базовим класом. Такий набір пов'язаних класів традиційно називається ієрархією класів. Ієрархія найчастіше є деревом, але може мати і більш загальну структуру графа.

Доступ до елементів класу може регулюватися за допомогою двох модифікаторів (специфікатором): `private` - закрита частина класу, `public` - відкрита частина класу. Для базових класів можливо використання ще одного модифікатора - `protected`, який визначає так звану захищену частину класу. Сенс «захисту» полягає в тому, що

елементи цієї частини класу є доступними для будь-якого похідного класу, але в той же час вони недоступні ззовні класів цієї ієрархії.

Крім цього, доступність і похідному класі регулюється ключем доступу до базового класу, що вказується в оголошенні похідного класу. Цей ключ визначає вид спадкування: відкрите (public), захищене (protected) або закрите (private).

Відкрите спадкування зберігає статус доступу всіх елементів базового класу, захищене - знижує статус доступу public елементів базового класу до protected, і нарешті, закрите - знижує статуси доступу public і protected елементів базового класу до private. Зауважимо, що в C++ відношення між класами "is a" має місце тільки при відкритому успадкуванні.

Конструктори і деструктори з базового класу не успадковуються як конструктори і деструктор похідного класу, тому при створенні похідного класу постає питання, чи потрібні ці методи і як вони повинні бути реалізовані. Але вони успадковуються в межах підоб'єкту базового класу.

Вирішуючи питання про конструктори похідного класу, потрібно керуватися наступними правилами:

- Якщо в базовому класі взагалі немає конструктора або є конструктор за умовчанням, то похідному класу конструктор потрібен тільки в тому випадку, коли потрібно ініціалізувати поля, введені в цьому класі.

- Якщо ви не визначили жодного конструктора, компілятор самостійно створить конструктор за умовчанням, з якого буде викликаний конструктор за замовчуванням базового класу.

- Якщо в базовому класі є конструктор з параметрами, то в похідному класі, як правило, потрібно задати конструктор зі списком параметрів, які мають значення для передачі конструктору базового класу; конструктор базового класу треба викликати в списку ініціалізації.

Необхідність в деструкції для похідного класу визначається тим, потрібно чи ні звільняти будь-які ресурси, виділені в конструкторі. Якщо такої необхідності немає, то можна довірити компілятору створити деструктор по замовчанню. У ньому забезпечується виклик деструктора базового класу.

На етапі виконання програми при створенні об'єкта похідного класу спочатку викликаються конструктори базових класів, починаючи з самого верхнього рівня, потім конструктори об'єктів-елементів класу, і в останню чергу — конструктор класу. При знищенні об'єкта (наприклад, коли покидається область його видимості) деструктори викликаються в порядку, зворотному викликам конструкторів.

Якщо базовий клас успадковується з ключем `private`, можна вибірково зробити деякі його елементи доступними в похідному класі, оголосивши їх в секції `public` похідного класу за допомогою операції доступу до області видимості:

```
class Base{
public:
void f() { std::cout<<"Hello"; }
};

class Derived : private Base{
public:
    using Base::f;
};
```

Висновки

Механізм успадкування класів дозволяє будувати ієрархії, в яких похідні класи отримують елементи батьківських, або базових, класів і можуть доповнювати їх або змінювати їх властивості. При великій кількості ніяк не зв'язаних класів управляти ними стає неможливим. Спадкування дозволяє впоратися з цією проблемою шляхом упорядкування та ранжування класів, тобто об'єднання спільних для декількох класів властивостей в одному класі і використання його в якості базового.

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на лекції

1. Що таке успадкування?
2. Як впливає на успадкування специфікатор доступу перед іменем базового класу?

Т Конструктори та деструктори при успадкуванні

ема

Мета

Ознайомитись з конструкторами та деструкторами при

Література

1 Довбуш, Г. Ф. Visual C++ на примерах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко
под

ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

2 Павловская, Т. А. C/C++. Программирование на языке высокого уровня [Текст]
Т.

А. Павловская. – М. - СПб. : ПИТЕР, 2003. - 461 с.

Поняття успадкування є одним з базових понять об'єктно-орієнтованого програмування.

Похідний клас при успадкуванні повинен мати свої власні конструктори.

Конструктори та деструктори при успадкуванні

Для різних методів класу існують різні правила наслідування. Розглянемо спадкування класів і проблеми, що виникають при цьому, на прикладі.

Конструктори і деструктори з базового класу не успадковуються як конструктори і деструктор похідного класу, тому при створенні похідного класу постає питання, чи потрібні ці методи і як вони повинні бути реалізовані.

Вирішуючи питання про конструкторів похідного класу, потрібно керуватися наступними правилами:

- Якщо в базовому класі взагалі немає конструктора або є конструктор за умовчанням, то похідному класу конструктор потрібен тільки в тому випадку, коли потрібно ініціалізувати поля, введені в цьому класі.

- Якщо ви не визначили жодного конструктора, компілятор самостійно створить конструктор за умовчанням, з якого буде викликаний конструктор за замовчуванням базового класу.

- Якщо в базовому класі є конструктор з аргументами, то в похідному класі, як правило, потрібно задати конструктор зі списком аргументів, які мають значення для

передачі конструктору базового класу; конструктор базового класу треба викликати в списку ініціалізації.

Необхідність в деструкції для похідного класу визначається тим, чи потрібно звільняти будь-які ресурси, виділені в конструкторі. Якщо такої необхідності немає, то можна довірити компілятору створити деструктор по замовчанню. У ньому забезпечується виклик деструктора базового класу.

На етапі виконання програми при створенні об'єкта похідного класу спочатку викликаються конструктори базових класів, починаючи з самого верхнього рівня, потім конструктори об'єктів-елементів класу, і в останню чергу - конструктор класу. При знищенні об'єкта (наприклад, коли покидається область його видимості) деструктори викликаються в порядку, зворотному викликом конструкторів.

Створимо похідний від класу `monstr` клас `daemon`. Базовий клас наведено з деякими методами, доданими у попередніх розділах:

```
enum color {red, green, blue};  
//      Клас monstr  
class monstr{  
    int health, ammo;  
    color skin;  
    char *name;  
public:  
    //      Конструктори:  
    monstr(int he = 100, int am = 10);  
    monstr(color sk);  
    monstr(char * nam);  
    monstr(monstr &M);  
    //      Деструктор:  
    ~monstr() (delete [] name;)  
    //      Операції:  
    monstr& operator ++();  
    monstr operator ++(int);  
    operator int();
```

```

    bool operator >(monstr &M);
    const monstr& operator = (monstr &M);
//    Методи доступу до полів:
    int get_health( ) const {return health;}
    int get_ammo( ) const {return ammo;}
//    Методи, що змінюють значення полів:
    void change_health(int he){ health = he;}
//    Інші методи:
    void draw(int x, int y, int scale, int position);
}:

//    Клас daemon
class daemon : public monstr{
    int brain;
public:
//    Конструктори:
    daemon(int br = 10){brain = br;};
    daemon(color sk) : monstr (sk) {brain = 10;}
    daemon(char * nam) :onstr (nam) {brain =10;}
    daemon(daemon &M) :onstr (M) {brain = M.brain;}
//    Операції:
    const daemon& operator = (daemon &M){
        if (&M == this) return *this;
        brain = M.brain;
       onstr::operator = (M);
        return *this;
    }
//    Методи, що змінюють значення полів:
    void think ( );
//    Інші методи:
    void draw(int x, int y, int scale, int position):

```

};

У класі `daemon` введено поле `brain` і метод `think`, визначені власні конструктори і операція присвоювання, а також перевизначений метод відтворення `draw`. Всі поля класу `monstr`, операції (крім привласнення) і методи `get_health`, `get_ammo` і `change_health` успадковуються в класі `daemon`, а деструктор формується за замовчуванням. Розглянемо правила успадкування різних методів.

Конструктори успадковуються, як конструктори підоб'єкту базового класу, але не успадковуються як конструктори об'єкту похідного класу, похідний клас повинен мати власні конструктори. Порядок виклику конструкторів визначається наведено нижче правилами.

1. Якщо в конструкторі похідного класу явний виклик конструктора базового класу відсутній, автоматично викликається конструктор базового класу за замовчуванням (тобто той, який можна викликати без параметрів). Це використано в першому з конструкторів класу `daemon`.

2. Для ієрархії, що складається з декількох рівнів, конструктори базових класів викликаються, починаючи з самого верхнього рівня. Після цього виконуються конструктори тих елементів класу, які є об'єктами, в порядку їх оголошення в класі, а потім виповнюється конструктор класу.

3. У випадку декількох базових класів їх конструктори викликаються в порядку оголошення.

Якщо конструктор базового класу вимагає вказівки параметрів, він повинен бути явним чином викликаний в конструкторі похідного класу в списку ініціалізації (це продемонстровано в трьох останніх конструкторах).

Операція присвоювання успадковується за тими є принципами, що і конструктор, тому її також потрібно явно визначити в класі `daemon`. Зверніть увагу на запис функції-операції: у її тілі застосований явний виклик функції-операції привласнення з базового класу. Щоб краще уявити собі синтаксис виклику, ключове слово `operator` разом зі знаком операції можна інтерпретувати як ім'я функції-операції.

Виклик функцій базового класу краще копіювання фрагментів коду з функцій базового класу в функції похідного. Окрім скорочення обсягу коду, цим досягається

спрощення модифікації програми: зміни потрібно вносити тільки в одну точку програми, що скорочує кількість можливих помилок.

Нижче перераховані правила успадкування деструкторів.

1. Деструктори успадковуються за тими ж принципами, що і конструктори, і якщо програміст не описав в похідному класі деструктор, він формується за замовчуванням і викликає деструктори всіх базових класів.

2. На відміну від конструкторів, при написанні деструктора похідного класу в ньому не потрібно явно викликати деструктори базових класів, оскільки це буде зроблено автоматично.

3. Для ієрархії класів, що складається з декількох рівнів, деструктори викликають в порядку, строго зворотному викликом конструкторів: спочатку викликається деструктор класу, потім - деструктори елементів класу, а потім деструктор базового класу.

Поля, успадковані з класу `monstr`, недоступні функцій похідного класу, оскільки вони визначені в базовому класі як `private`. Якщо функцій, визначених в `daemon`, потрібно працювати з цими полями, можна або описати їх в базовому класі як `protected`, або звертатися до них за допомогою функцій з `monstr`, або явно перевизначити їх в `daemon` так, як було показано в раніше.

Розглядаючи спадкування методів, зверніть увагу на те, що в класі `daemon` описаний метод `draw`, перевизначаються метод з тим же ім'ям в класі `monstr` (оскільки отрисовка різних персонажів, природно, виконується по-різному). Таким чином, похідний клас може не тільки доповнювати, але і коректувати поведінку базового класу.

(Перевизначати в похідному класі рекомендується тільки віртуальні методи (див. слід, розділ).)

Доступ до перевизначення методу базового класу для похідного класу виконується через ім'я, уточнене за допомогою операції доступу до області видимості (`::`).

Похідні класи успадковують всі перевантажені у базовому класі операторні функції-члени, крім функції, яка перевантажує оператор привласнення. Цей оператор повинен бути визначений у кожному похідному класі тільки за допомогою операторної функції-члена похідного класу.

Висновки

Вирішуючи питання про конструкторів похідного класу, потрібно керуватися наступними правилами:

- Якщо в базовому класі взагалі немає конструктора або є конструктор за умовчанням, то похідному класу конструктор потрібен тільки в тому випадку, коли потрібно ініціалізувати поля, введені в цьому класі.

- Якщо ви не визначили жодного конструктора, компілятор самостійно створить конструктор за умовчанням, з якого буде викликаний конструктор за замовчуванням базового класу.

- Якщо в базовому класі є конструктор з аргументами, то в похідному класі, як правило, потрібно задати конструктор зі списком аргументів, які мають значення для передачі конструктору базового класу; конструктор базового класу треба викликати в списку ініціалізації.

Необхідність в деструкції для похідного класу визначається тим, чи потрібно звільняти будь-які ресурси, виділені в конструкторі.

Питання та завдання до контролю знань студентів

1 Для актуалізації опорних знань та перевірки попереднього матеріалу

1 Як перевизначити метод базового класу?

2 Чим відрізняються поняття перевантаження та перевизначення?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

1. Які правила успадкування для конструкторів?

2. Яка послідовність створення та знищення об'єктів в ієрархії класів?

Тема**Перевизначення методів базового класу****Мета заняття**

Навчитись перевизначати методи базового класу

Література

Довбуш, Г. Ф. Visual C++ на прикладах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

Механізм успадкування класів дозволяє будувати ієрархії, в яких похідні класи отримують елементи батьківських, або базових, класів і можуть доповнювати їх або змінювати їх властивості. При великій кількості ніяк не зв'язаних класів управляти ними стає неможливим. Спадкування дозволяє впоратися з цією проблемою шляхом упорядкування та ранжування класів, тобто об'єднання спільних для декількох класів властивостей в одному класі і використання його в якості базового.

Перевизначення методів базового класу

Як правило, базові класи мають відкриті елементи для того, щоб ними користувались класи-нащадки, точніше, коли ми обираємо базовий клас, то потрібно знати, чим ми збираємось користуватися. Крім того, потрібно обирати потрібний ключ доступу.

Методи при наслідуванні можуть бути перевизначені у класах-нащадках. Тобто, наприклад методи

```
CBase :: method_1( ) { }  
CDerived1:: method_1( ) { }  
CDerived2 :: method_1( ) { }
```

можуть мати кожен свою реалізацію, якщо клас CBase є базовим для CDerived1, а CDerived1 є базовим для CDerived2.

Дані при наслідуванні також можуть бути перевизначені у класах-нащадках, тобто може бути змінений їх тип.

Якщо в похідному класі потрібно дещо інша реалізація методу, успадкованого з базового класу, можна це зробити. Перевизначення методу проводиться шляхом оголошення в похідному класі методу з таким же ім'ям і списком параметрів (якщо

список параметрів буде іншим, метод нащадка замінить або приховає відповідний метод базового класу) Якщо знадобиться все-таки викликати з нащадка метод предка, використовується операція доступу до області видимості::, наприклад:

```
#include <iostream>
using namespace std;
class Base {
public:
    void Display( ) { cout << "Hello, world!" << endl; }
};
class Derived : public Base {
public:
void Display() {
    Base::Display( );    // Вызов метода базового класса
    cout <<"How are you?" << endl; }
};
class SubDerived : public Derived {
public:
    void Display() {
        Derived::Display( );    // Вызов метода базового класса
        cout << "Bye!" << endl;
    };
};

int main( ) {
SubDerived sd;
sd.Display( );
return 0;
}
Програма виведе на екран:
Hello, world!
How are you?
Bye!
```

Доступ до об'єктів ієрархії

Найефективніше працювати з об'єктами однієї ієрархії через вказівник на базовий клас. Справа в тому, що при відкритому (public) спадкуванні можна присвоюються такому вказівником адреса об'єкту як базового класу, так і будь-якого похідного класу. Така можливість є цілком логічною: в якості представника більш загального класу (наприклад, класу Polygon) використовується об'єкт спеціалізованого класу (наприклад, класу Triangle або Tetragon). Однак, якщо не вжити спеціальних заходів (про які йдеться в наступному розділі), то при роботі з таким вказівником - незалежно від того, яку адресу йому присвоєно - виявляються доступними лише елементи базового класу. Причиною є те, що в загальному випадку під час компіляції

компілятор не може знати, з яким об'єктом насправді пов'язаний цей покажчик.

Наприклад, розглянемо наступний фрагмент:

```
class Base {
public:
    //...
    void Modify();
};
class Derived : public Base {
public:
    void Modify();
};

int main() {
    int mode;
    Base* pB;
    cin >> mode;
    if (mode == 1)    pB = new Base;
    else             pB = new Derived;
    pB->Modify();      // на що вказує
pB?
    return 0;
}
```

Так як компілятор не може передбачити, який вибір буде зроблений на етапі виконання, то він вибирає метод по типу покажчика PB, тобто Base :: Modify (). Така стратегія називається раннім, або статичним , зв'язуванням . Тому, якщо в похідному класі заміщений деякий метод базового класу (наприклад, Derived::Modify()), то він виявляється недоступним. Продемонструємо це на прикладі:

```
#include <iostream>
using namespace std;
class Base {
public:
    Base( int _x = 10) { x = _x };
    void ShowX() { cout << "x="<<x<<" "; }
    void ModifyX( ) { x *= 2; }
protected:
    int x;
};
class Derived : public Base {
public:
    void ModifyX() { x /= 2; }
};
int main( ) {
    Base b;           Derived d;
    b.ShowX();        d.ShowX();
    Base* pB;

    pB = &b;          pB->ModifyX();      pB->ShowX();
    pB = &d;          pB->ModifyX();      pB->ShowX();
    return 0;
}
```

```
}  
Програма виведе на екран:  
x = 10; x = 10; x = 20; x = 20;  
тобто другий виклик метода ModifyX() є також з базового класу.
```

Висновки

Іноді в похідному класі потрібно дещо інша реалізація методу, успадкованого з базового класу. Мова дозволяє це зробити. Заміщення (перевизначення) методу проводиться шляхом оголошення в похідному класі методу з таким же ім'ям. Якщо знадобиться все-таки викликати з нащадка метод предка, використовується операція доступу до області видимості::

Питання та завдання до контролю знань студентів

- 1 Для актуалізації опорних знань та перевірки попереднього матеріалу
 1. Що таке успадкування?
 2. Як впливає на успадкування специфікатор доступу перед іменем базового класу?
- 2 Для узагальнення та перевірки засвоєного матеріалу на лекції
 - 1 Як перевизначити метод базового класу?
 - 2 Чим відрізняються поняття перевантаження та перевизначення?

Література

Довбуш, Г. Ф. Visual C++ на прикладах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

ВСТУП

Шаблони класів представляють собою клас, в якому визначені дані та методи.

Шаблони дають можливість багаторазово використати один і той же програмний код.

Шаблони класів

Шаблон класів (`class template`) — параметризований клас, на основі якого будуються інші класи. Шаблон визначає правила конструювання кожного окремого класу з безлічі допустимих цим шаблоном варіантів. Класи, в свою чергу, визначають правила конструювання об'єктів. Графічно зв'язок між шаблоном, класами та об'єктами представлена на рисунку 1.

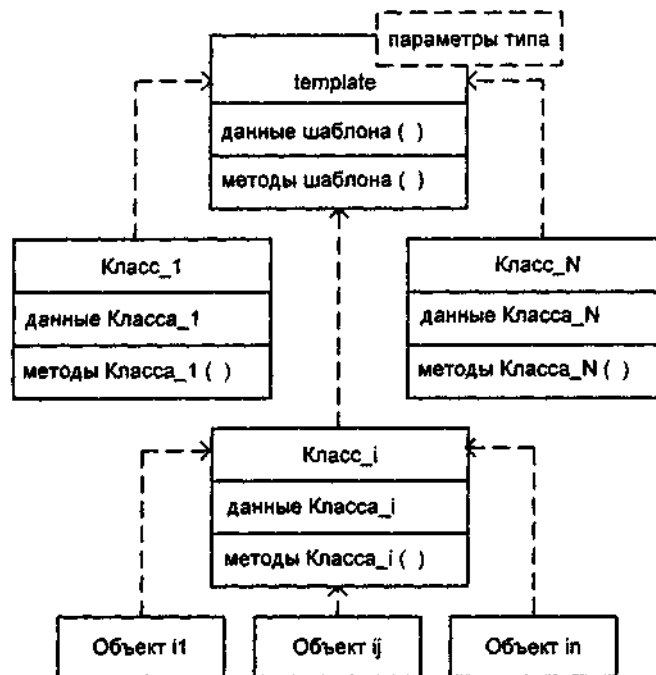


Рис. 1 – Графічний вигляд зв'язку між шаблоном, класами та об'єктами

Шаблон класів є клас, в якому визначені дані і методи, але фактичний тип даних задається як параметр при створенні об'єктів класу. Шабини дають можливість багаторазово використовувати один і той самий код, який дозволяє компілятору автоматизувати процес реалізації типу. Для шаблонних класів характерними є наступні властивості:

1. Шаблон дозволяє передати в клас один або кілька типів у вигляді параметрів. Переданим типом може бути будь-який базовий або похідний тип, включаючи тип class.

2. Параметрами шаблону можуть бути не тільки типи, а й вирази.

3. Оголошення шаблону повинно бути тільки глобальним.

4. Кожен реалізований шаблон (створений об'єкт шаблону класів) вимагає власного скомпільованого об'єктного коду. Щоб скоротити розмір об'єктного модуля програми, рекомендується успадковувати найбільш загальну частину шаблону від базового класу. Для базового та похідного класів в цьому випадку необхідний один і той же реалізований тип (базовий та похідний клас будемо розглядати в темі успадкування класів).

5. Базовий клас для шаблону може бути як шаблонним, так і звичайним класом. Звичайний клас може бути породжений від реалізованого шаблону класів.

6. Не можна використовувати покажчик на базовий шаблонний клас для отримання доступу до методів похідних класів. Справа в тому, що типи, отримані навіть з одного і того ж шаблону, завжди є різними.

7. В опис шаблону класів можна включати дружні функції. Якщо функція-друг не використовує специфікатор шаблону, то вона вважається універсальною для всіх примірників шаблону. Якщо ж у прототипі функції-друга міститься шаблон параметрів, то ця функція буде дружній тільки для того класу, примірник якого створюється.

8. Статичні члени-дані специфічні для кожного реалізованого шаблону.

Найчастіше шаблони класів використовуються для програмування контейнерних класів (контейнерів), які призначені для зберігання об'єктів $\neg$ тів інших класів.

Оголошення шаблону класів

Оголошення шаблону класів може бути тільки глобальним і має починається з рядка, що має наступний формат:

```
template <class параметр_тіпа1, ..., class параметр_тіпаN>
```

Параметр_тіпа заміщає довільний тип і використовується в реалізації класу як ім'я типу. Кутові дужки і ключове слово `class` перед кожним параметром_тіпа - обов'язкові атрибути оголошення шаблону.

Методи повинні бути оголошені як шаблони функцій, тому заголовок методу, визначення якого знаходиться за межами специфікації класу, має наступний формат:

```
template <class параметр_тіпа1, ..., class параметр_тіпаN>
```

```
тіп_функції ім'я_шаблону <параметр_тіпа1, ..., параметр_тіпаN>
```

```
:: Ім'я_функції (список параметрів функції)
```

Специфікація і реалізація шаблону класів при роздільної компіляції обов'язково повинні знаходитися в одному файлі. Щоб можна було компілювати шаблон, цей файл повинен мати розширення. `.cpp`. Файл з шаблоном включається в програму за допомогою директиви препроцесора `# include`.

Приклад оголошення шаблону класів

```
template <class Type>
class stack
```

```

{
private:
    int top;
    Type s[10];

public:
    stack (): top(-1)
    {}

    void push(Type var)
    {
        top++;
        s[top] = var;
    }

    Type pop();
};

template <class Type>
Type stack<Type>::pop()
{
    Type var = s[top];
    top--;
    return var;
}

```

При створенні об'єкту після імені класу треба поставити кутові дужки, в яких вказати необхідний тип. Після цього об'єкти використовуються як звичайно.

```

int main(...)
{
    stack<int> s1;
    stack<float> s2;

    s1.push(3);
    s1.push(2);

    s2.push(0.5);

    return 0;
}

```

Шаблон `stack` (лістинг) описує безліч класів для зберігання об'єктів різного типу.

Висновки

Шаблон класів є клас, в якому визначені дані і методи, але фактичний тип даних задається як параметр при створенні об'єктів класу. Шаблони дають можливість

багаторазово використовувати один і той самий код, який дозволяє компілятору автоматизувати процес реалізації типу.

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на лекції

1. Що таке шаблон класів?
2. Як описується шаблон класів?
3. Як передаються в клас типи за допомогою шаблону?
4. У яких файлах розміщують шаблони класів?

Література

Довбуш, Г. Ф. Visual C++ на прикладах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

Механізм успадкування дозволяє класу отримувати елементи іншого класу, а також при необхідності змінювати та доповнювати їх власними елементами.

Простим (в окремих джерелах поодиноким) успадкуванням називають таке, при якому клас-нащадок (який іще називають похідний клас) має один батьківський (базовий).

Похідний клас, в свою чергу, сам може служити базовим класом. Класи, пов'язані успадкуванням, складають ієрархію.

Множинне успадкування, на відміну від поодинокого, дозволяє вирішити питання додавання властивостей від декількох базових класів.

У разі множинного спадкування після двокрапки перераховуються через кому всі базові класи зі своїми ключами (модифікаторами) доступу.

У попередніх прикладах частіше доступ до елементів класу регулювався за допомогою двох модифікаторів (специфікатором): `private` - закрита частина класу, `public` - відкрита частина класу. Для базових класів можливо використання ще одного модифікатора - `protected`, який визначає так звану захищену частину класу. Сенса «захисту» полягає в тому, що елементи цієї частини класу є доступними для будь-якого похідного класу, але в той же час вони недоступні ззовні класів цієї ієрархії.

Крім цього, доступність в похідному класі регулюється ключем доступу до базового класу, що вказується в оголошенні похідного класу. Цей ключ визначає вид спадкування: відкрите (`public`), захищене (`protected`) або закрите (`private`).

Відкрите спадкування зберігає статус доступу всіх елементів базового класу, захищене - знижує статус доступу `public` елементів базового класу до `protected`, і нарешті, закрите - знижує статуси доступу `public` і `protected` елементів базового класу

до private. В C++ відношення між класами «is a» ("як є") має місце тільки при відкритому успадкуванні.

Множинне успадкування

Множинне успадкування означає, що клас має декілька базових класів.

Приклад [оголошення](#):

```
class A: public B, protected C { ...}
```

Базові класи оголошуються зі специфікаторами доступу, можливо різними перед кожним іменем класу, за замовчуванням, як і раніше, специфікатор private.

Якщо в базових класах є, наприклад, однойменні елементи (поля або методи), при цьому може відбутися конфлікт ідентифікаторів, який усувається за допомогою операції доступу до області видимості. Приклад конфлікту ідентифікаторів методів get_health() та його вирішення наведений в лістингу 1.

Лістинг 1 – Приклад конфлікту ідентифікаторів методів get_health()

```
class monstr{
public:
    int get_health( ) { cout<<"Hello, monstr"; }
};

class hero{
public:
    int get_health( ) { cout<<"Hello, hero"; }
};

class ostrich: public monstr, public hero{
... //Власні поля та методи класу.
};

int main( ){
    ostrich A;
    cout << A.get_health( ); //ПОМИЛКА!!!
    cout << A.monstr::get_health( );//Немає помилки.
    cout << A.hero::get_health( );//Немає помилки.
    return 0;
}
```

Незалежно від наявності власних полів та методів об'єкт класу ostrich отримає два підоб'єкти базових класів monstr та hero. Завдяки відкритому успадкуванню методи get_health() будуть доступні як відкриті. Але при спробі звернення з головної функції до get_health() з об'єкта нащадка виникне помилка, пов'язана з конфліктом

ідентифікаторів, тому що є два таких методи і компілятор не знає, до якого з них звернення.

Як видно з прикладу, для виклику методу `get_health` потрібно явно вказувати клас, в якому він описаний. Використання звичайної для виклику методу класу конструкції `A.get_health()` призведе до помилки, оскільки компілятор не в змозі розібратися, до методу якого з базових класів потрібно звернутися.

Іноді в похідному класі потрібно дещо інша реалізація методу, успадкованого з базового класу. Заміщення (перевизначення) методу проводиться шляхом оголошення в похідному класі методу з таким же ім'ям. Якщо знадобиться все-таки викликати з нащадка метод предка, використовується операція доступу до області видимості::, наприклад, як наведено в лістингу 2.

Лістинг 2 – Приклад використання операції доступу до області видимості.

```
class Base {
public:
void Display( ) { cout << "Hello, world!" << endl; }
};

class Derived : public Base {
public:
void Display( ) {
Base ::Display( ) ; // Виклик методу базового класу
cout << "How are you?" << endl;
}
};

class SubDerived : public Derived {
public:
void Display( ) {
Derived :: Display( ); // Виклик методу базового класу
cout << "Bye!" << endl;
}
};

int main( ) {
SubDerived sd;
sd.Display( );
return 0;
}
Ця програма, виведе на екран:
Hello, world!
How are you?
Bye!
```

В ієрархії, подібної до наведеної, клас Base називають косвеним для SubDerived. Нагадаємо, що на діаграмі класів відношення успадкування позначається стрілкою, які направлена до прямокутника, що схематично зображує базовий клас.

Висновки

Множинне успадкування означає, що клас має декілька базових класів. Якщо в базових класах є однойменні елементи, при цьому може відбутися конфлікт ідентифікаторів, який усувається за допомогою операції доступу до області видимості:

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу:

1. Що таке множинне успадкування?
2. Як успадковуються елементи базового класу?

Література

1 Довбуш, Г. Ф. Visual C++ на примерах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под
ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.
2 Лафоре, Р. Объектно-ориентированное программирование в C++ [Текст] /
Р. Лафоре. – М. - СПб. : ПИТЕР, 2004.- 923 с.

Література

Довбуш, Г. Ф. Visual C++ на примерах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

Множинне успадкування дозволяє вирішити питання додавання властивостей від декількох базових класів. При множинному успадкуванні можуть виникати конфлікти, пов'язані з однаковими ідентифікаторами елементів класів.

Протокол класу ілюструє більшість ключових можливостей опису класу. Опис протоколу класу представляється в файлі заголовка (інтерфейсу) класу: тут містяться прототипи всіх необхідних функцій-членів класу. Визначення функцій-членів розміщується в реалізації класу.

Таким чином, в протоколах класу містяться конструктор за замовчуванням, конструктори з параметрами, конструктор копіювання, деструктор. Сюди входять також різні методи класу.

Наприклад, щоб забезпечити можливість завжди знати, скільки об'єктів даного типу існує, методи, які реалізують виведення інформації про стан об'єкта на екран і в файл.

Всі функції, що не міняють стан об'єкта, повинні бути описані як константні. Всі службові методи (які не призначені для виклику користувачем) повинні бути не відкритими.

Якщо у базовому та похідному класах містяться однойменні члени-дані, які мають різний тип, то доступ виконується через ім'я, уточнене за допомогою операції доступу до області видимості (::).

Доступ до перевизначеного методу базового класу у похідному класі виконується так само.

Коли базові класи є похідними від одного класу, може виникати конфлікт імен.

При так званому ромбовидному множинному спадкуванні виникають проблеми з доступом до успадкованих елементів.

Припустимо, що від базового класу А, що має певний елемент х, наслідуються два класи В і С, а клас D є похідним від цих двох класів ([множинне успадкування](#)). Якщо спробувати звернутися до елементу х з методів класу D, то компілятор сприйме вираз D::x як неоднозначний (оскільки неясно, який елемент класу А мається на увазі: успадкований через клас В чи успадкований через клас С) і припинить роботу.

Для вирішення цієї проблеми передбачений механізм, завдяки якому в клас D буде включена лише одна копія класу А. Досягається це додаванням специфікатора virtual перед модифікаторами доступу до А в [оголошеннях](#) класів В і С. Приклад наведений в лістингу 1.

Лістинг 1 – Приклад неоднозначності при ромбовидній ієрархії

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

class A {
public:
    A(int _x = 0) { x = _x; }
protected:
    int x;
};

class B : virtual public A {
public:
    void AddB(int y) { x += y; }
};

class C : virtual public A {
public:
    void AddC(int y) { x += y; }
};

class D : public B, public C { //Без virtual в оголошення класів В,
    С буде двукратно успадковуватись підоб'єкт А, що призведе до конфлікту
public:
    void ShowX( ) { cout << "x = " <<x << endl; }
};

int main( ) {
    D d;
    d.ShowX( );
    d.AddB(10);
    d.ShowX( );
    d.AddB(5);
    d.ShowX( );
    return 0;
}
```

}

В ієрархії класів зі складним спадкуванням цілком може вийти так , що клас побічно успадкує кілька примірників деякого базового класу. Якщо В і С обидва є нащадками А, а D успадковує В і С, то D отримує подвійний набір елементів класу А. Це може приводити до неоднозначності при зверненні до них, що буде викликати помилки часу компіляції.

У лістингу похідні класи В і С віртуально успадковують базови клас А, щоб в похідному класі D не було копій підоб'єктів базового класу А.

Деструктор цього класу оголошений віртуальним, щоб при руйнуванні об'єктів похідних класів викликалися їх деструктори.

Висновки

Якщо у базовому та похідному класах містяться однойменні члени-дані, які мають різний тип, або якщо у двох базових класів є одноіменні ідентифікатори, то доступ до них виконується через ім'я, уточнене за допомогою операції доступу до області видимості (::).

У випадку ромбовидної ієрархії клас нащадок отримує дві копії підоб'єкта косвеного базового класу, що призведе до помилки при зверненні до його елементу. Для вирішення даного конфлікту використовують так зване віртуальне успадкування, коли перед іменем першого базового класу в ієрархії в оголошенні успадкування використовують слово `virtual`. Це забороняє створення другої копії підоб'єкту при успадкування, тобто буде успадковуватись тільки один підоб'єкт даного класу.

Питання та завдання до контролю знань студентів

1 Для актуалізації опорних знань та перевірки попереднього матеріалу

- Що таке [множинне успадкування](#)?

- Як успадковуються елементи базового класу?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

- Як вирішити конфлікт, пов'язаний з одноіменними ідентифікаторами при успадкуванні?
- Як вирішити конфлікт при ромбовидній ієрархії?

Література

Лафоре, Р. Объектно-ориентированное программирование в С++ [Текст] / Р. Лафоре. –
М. - СПб. : ПИТЕР, 2004.- 923 с.

Тема

Контейнерні класи

Мета заняття

Розглянути особливості використання контейнерних класів

Література

Довбуш, Г. Ф. Visual C++ на примерах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

Шаблонний клас, по аналогії з шаблоною функцією, може приймати параметр типу і містити методи, які є шаблонними функціями. Тобто, замість декількох класів, які відрізняються тільки типами даних, можна написати шаблонний клас.

Успадкування та шаблони класів

Універсальність шаблонів полягає в тому, що в шаблонних класах можна застосовувати ті ж прийоми програмування, що і в звичайних класах. Шаблонні класи можуть виступати як в якості базових класів, так і нащадків і компонентів інших класів.

Можна створити шаблон стека на основі шаблону масиву. Або можна взяти шаблон масиву і застосувати його для створення масиву, елементи якого є стеками, заснованими на шаблоні стека. Це означає, що можливий код, приклад якого наведений в лістингу 29.1.

Лістинг 29.1 – Приклад успадкування з використанням шаблонного класу.

```
template <typename T>                                // или <class T>
class Array
{
private:
    T entry;
};

template <typename Type>
```

```

class GrowArray : public Array<Type> {...}; // успадкування
template <typename Tp> class Stack
{
Array<Tp> ar; // використовує Array<> в якості компонента
...
}

Array < Stack<int> > asi;      // масив стеків значень int

```

Розглянемо використання шаблонів класів на прикладі програми побудови стека з 10 елементів. Ці елементи можуть бути різних типів. У прикладі функція push визначена всередині класу, а функція pop - поза. Для всіх функцій, визначених за межами класу, потрібно обов'язково вказувати ключове слово template. Вираз перед ім'ям функції збігається з тим, яке вказується перед ім'ям класу. Приклад наведений в лістингу 29.2.

Лістинг 29.2 – Приклад використання шаблонів класів.

```

template <class Type>
class stack
{
private:
    int top;
    Type s[10];

public:
    stack (): top(-1)
    {}

    void push(Type var)
    {
        top++;
        s[top] = var;
    }

    Type pop();

```

```

    void view();
};

template <class Type>
Type stack<Type>::pop()
{
    Type var = s[top];
    top--;
    return var;
}

template <class Type>
void stack<Type>::view()
{
    cout << endl;
    for (int i = 0; i<=top; i++) {
        cout << s[i] << '\t';
    }
    cout << endl;
}
}

```

Шаблонні класи мають таку особливість, що крім стандартних типів вони можуть працювати і з одними. Для демонстрації цього в прикладі створений простий клас **warrior**.

```

class warrior
{
public:
    int health;
    warrior () : health(0) {}
};

```

Розглянемо головну функцію. При створенні об'єкта після імені класу потрібно поставити кутові дужки, в яких вказати необхідний тип. Після цього об'єкти використовуються як зазвичай. Тепер в стеках також можна розміщувати змінні типу **warrior**.

```

int main(int argc, char* argv[])
{

```

```
    printf("Hello World!\n");
stack<int> s1;
stack<float> s2;

s1.push(3);
s1.push(2);
//s1.pop();
s2.push(0.5);
stack<warrior> s;

warrior w1;
warrior w2;
warrior w3;

s.push(w1);
s.push(w3);

s.pop();
s.push(w2);
s1.view();
s2.view();
    return 0;
}
```

Головна мета створення всіх розглянутих методів - надати програмісту можливість повторного використання перевіреного коду без його ручного копіювання. Це спрощує програмування і підвищує надійність програм.

Висновки

Шаблони функцій і шаблони класів дозволяють користувачам створювати родові функції і класи, що працюють з різними типами даних і класами. Після їх кодування і тестування користувачі можуть створювати різні екземпляри цих функцій і класів (спеціалізації класів) для конкретних типів даних і (або) класів. Таким чином, застосування шаблонів функцій і класів дозволяє домогтися істотної економії часу розробника.

Еще одна перевага шаблонів - скорочення (в розрахунку на одо програму) числа унікальних функцій і імен класів, які потребують визначення. Це прискорює процес компіляції і знижує ймовірність конфлік тієї імен в призначених для користувача програмах.

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Які правила успадкування для шаблонів класів?
- 2 Що таке спеціалізація класу?

Тема Узагальнення знань і умінь студентів

Мета заняття Узагальнення знань і умінь студентів

Література

Довбуш, Г. Ф. Visual C++ на примерах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

I. Теоретичні питання:

1. Дати визначення класу. Як оголошується клас? Які файли описують клас? Навести приклад класу.
2. Для чого призначений конструктор класу? Чим відрізняються конструктор за замовчанням та конструктор з параметрами?
3. Охарактеризувати команди роботи з проектом у середовищі MS Visual C++.
4. Скільки даних та скільки методів може мати клас? Привести приклади.
5. Як описуються методи класу? Як звертаються до методів класу? Привести приклади.
6. Скільки конструкторів та деструкторів може мати клас? Привести приклади.
7. Який інструментарій використовується для створення проектів у середовищі MS Visual C++? Які типи проектів можна при цьому створювати?
8. Для чого призначаються специфікатори доступу до членів класу?
9. Що описується в специфікації класу? Який файл містить специфікацію класу? Як створюється цей файл?
10. Для чого використовуються та як описуються операторні функції?
11. Охарактеризувати редактор коду у середовищі Microsoft Visual C++ та його можливості.
12. Як оголошується об'єкт класу? Привести основні формати оголошення об'єкту класу.
13. Що таке члени класу? Де вони описуються? Як отримати доступ до членів-даних?
14. У якій частині проекту оголошується об'єкт класу? Привести основні формати оголошення об'єктів класу.
15. Яким способом можна створити у проекті новий клас та які інструменти при цьому потрібно використати, які файли редагувати?
16. Охарактеризувати можливості регулювання доступу до членів класу за допомогою ключів доступу `private`, `protected` та `public`.

Між об'єктами програми існують зв'язки, які підтримують їх взаємодію. множину однотипних зв'язків між об'єктами двох різних класів називають відношенням.

В попередньому матеріалі було розглянуто відношення успадкування, при якому класи поділяються на батьківські та дочірні, і правила їх взаємодії (які елементи як успадковуються і який до них доступ).

Поширеним видом відносин є відносини типу частина - ціле (part of або has), коли один клас складає ціле, а об'єкти інших — його частини. Такий тип відносин називають агрегацією (aggregation).

Агрегація та композиція класів

Агрегація — можливість включення в один клас в якості членів-даних об'єкти іншого класу (відношення "містить" або "частина-ціле"). Ці поля, які є об'єктами інших класів, називають об'єктами-частинами, а об'єкт класу, який містить ці поля, називають об'єкт-ціле.

З погляду на програмний код, можна говорити про агрегацію, якщо поля одного класу є об'єктами іншого класу. Це зовсім інше відношення, ніж успадкування.

Композицію можна розглядати як більш сильний за відношенням вид агрегації.

Композиція — вид відношення "частина-ціле", при якому частина не може існувати без цілого. Ще говорять, що час життя об'єкту частини співпадає з часом життя об'єкта цілого.

Наприклад, клас візуальної форми містить два об'єкти візуальних кнопок "Зберегти", "Очистити". З погляду на предметну область ці класи було б недоцільно пов'язувати між собою успадкуванням. Успадкування використовують, якщо необхідно отримати елементи батьківського класу і, при необхідності, змінити або доповнити їх. Але з погляду на відношення "містить" можна сказати, що візуальна форма МІСТИТЬ кнопки. Відношення "містить" реалізується, як було сказано, як агрегація або композиція. В даному випадку більш доцільна композиція, тому що візуальна кнопка не існує без візуальної форми.

Приклад агрегації та композиції наведений в лістингу 1.

Лістинг 1 – Приклад агрегації та композиції

```
class A {
    int a_priv;
protected:
    int a_prot;
public:
    int a;
    /* ... */
};

class B {
public:
    int b;
    /* ... */
};

class C {
public:
    int c;
    /* ... */
};

class D {
public:
    int d;
    A o1;
    B* o2;
    C& o3;
    /* ... */
};

int main() {
    D od;
    cout<< od.o1.a_priv; // помилка
    cout<< od.o1.a_prot; // помилка
```

```
cout<< od.ol.a;  
  
}
```

Зверніть увагу на те, до яких елементів об'єктів частин є доступ у об'єкта цілого.

Тут не діють правила доступу успадкування, але діють правила специфікаторів доступу в самому класі.

Для класів А, В, С доступними ззовні будуть тільки відкриті елементи. В програмному коді звернення до закритих або захищених елементів класу А з об'єкта класу D викличе помилку.

Звернення до елементів класу через об'єкт, як і раніше, з використанням операції доступу до елементів класу (. або ->).

Під час вивчення теми успадкування було розглянуто, в якій послідовності створюються об'єкти класів в ієрархії класів, як відпрацьовують конструктори і деструктори у випадку поодинокого або множинного успадкування.

У випадку відношення частина-ціле об'єкти-частини, оголошені за значенням, створюються в тому порядку, в якому вони оголошені у класі цілого, який їх включає, і до того, як буде створений об'єкт цілого. Для ініціалізації об'єкта частини у списку ініціалізації конструкторів класу цілого можна явно викликати конструктори класів частин.

Відношення. оголошені за вказівником або посиланням, будуть розглянуті пізніше.

Особливості реалізації агрегації, пов'язані зі створенням і знищенням цілого і частин, розглянемо на прикладах. Агрегація надає більш слабкий зв'язок між класами, ніж композиція.

Фрагмент програмного коду прикладу "Бібліотека - книги" наведено в лістингу 2.

Лістинг 2 – Фрагмент програмного коду "Бібліотека - книги"

```
class CBook // Книга  
{  
private:  
    string m_author ;           // автор  
    string m_title ;           // назва  
    int m_year ;               // рік видання  
public:  
    // методи встановлення значень
```

```

void setAuthor ( const string & ) ;
void setTitle ( const string & ) ;
void setYear ( const int ) ;
// методи повернення значень
string getAuthor ( ) const { return m_author ; }
string getTitle ( ) const { return m_title ; }
int getYear ( ) const { return m_year ; }
} ;

class CCatalogue // Каталог книг
{
    int m_q ;                // фактична кількість книг
public:
    CBook* m_pBook [ 100 ] ; // масив вказівників на книги
public:
    bool ins ( const string&, const string&, int ) ;    // додавання
    bool del ( const string& ) ;    // видалення
    int find ( const string& ) const ;    // пошук
    int getNumberOfBook ( ) const ; // повернення кількості книг
} ;

class CMenu // Меню для роботи з каталогом книг
{
public:
    CCatalogue* m_p ;        // вказівник на каталог книг
    void showMenuAndSelect ( ) ;    // активізація меню
} ;

// головна функція
int main ( )
{
    CMenu menu ;
    menu.showMenuAndSelect ( ) ;
    return 0 ;
}

```

У лістингу реалізовано взаємодія об'єктів трьох класів: CMenu (описує

користувальницький інтерфейс), CCatalogue (описує масив з 100 покажчиків на об'єкти типу CBook) і CBook (описує книгу).

У головній функції оголошується об'єкт класу CMenu з ідентифікатором menu, який викликає відкритий метод класу ShowMenuAndSelect ()

Подальші дії програми залежать від реакції користувача на пропоновані цим методом послуги.

Об'єкти частки оголошені з використанням вказівників.

Висновки

Агрегація — можливість включення в один клас в якості членів-даних об'єкти іншого класу (відношення "містить" або "частина-ціле").

Композицію можна розглядати як більш сильний за відношенням вид агрегації.

Об'єкти-частини створюються в тому порядку, в якому вони оголошені у включає їхньому класі цілого і до того, як буде створений об'єкт цілого.

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на лекції

1. Що таке агрегація класів?
2. Що таке композиція класів?
- 3 Як створюються об'єкти при агрегації?

Література

Довбуш, Г. Ф. Visual C++ на прикладах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

Особливості конструювання класів при агрегації

Тема **Особливості конструювання класів при агрегації**

Мета заняття Розглянути особливості конструювання класів при агрегації

Агрегація та композиція — це поняття, які відтворюють взаємовідносини класів, пов'язаних відношенням частина-ціле. Такі відносини ще називають "містить".

В програмному кодї агрегація та композиція виглядають таким чином: клас містить поля, які являються об'єктами інших класів. Такі поля можуть бути оголошені за значенням, за вказівником або за посиланням.

Агрегація відрізняється від композиції силою зв'язку та часом життя частини та цілого. Композиція має більш міцний зв'язок, агрегація — більш слабкий. Приклад агрегації: кімната містить столи, столи можна винести з кімнати, і столи і кімната продовжать існувати. Кімната і стіл описуються різними класами.

Об'єкти-частини створюються в тому порядку, в якому вони оголошені у класі- цілому, який їх включає.

Створення об'єктів частин буде відбуватися до того, як буде створений об'єкт цілого.

При цьому можна або задавати початкові значення об'єкту частини, або не задавати початкові значення об'єкту частини, але при їх відсутності клас частин обов'язково повинен мати конструктор за замовчуванням. Якщо треба створити об'єкт-частину з використанням конструктора з параметрами, доцільно включити звернення до нього в список ініціалізації конструктора цілого.

Особливості конструювання класів при агрегації

Для включення об'єкта частини `m_part` класу `CPart` в клас `CWhole` є дві альтернативи:

- явно оголосити дане - `CPart m_part;`
- оголосити покажчик або посилання на тип частини - `CPart * m_p`
або `CPart &m_r`.

В даному випадку досліджується можливість включення в клас в якості членів-даних об'єктів іншого класу на прикладі, наведеному в лістингу.

Для ініціалізації об'єкта частини у списку ініціалізації конструкторів класу цілого явно викликаються конструктори класу частини.

В лістингу 2 приведено приклад оголошення об'єкту-частини через покажчик

Лістинг

```
#include "Station.h"

class CTrain          // опис рейсу
{
public:
    int m_num ;       // номер поїзда
    CStation* m_ps ;   // вказівник на станцію відправлення
    CTrain () ; CTrain ( int, char* ) ; CTrain ( CTrain& ) ;
    ~CTrain () ;
};

// реалізації класу CTrain
CTrain :: CTrain ( ) : m_num ( 0 ), m_ps ( new CStation ) { }
CTrain :: CTrain ( int num, char* p )
: m_num ( num ), m_ps ( new CStation ( p ) ) { }
CTrain :: CTrain ( CTrain& t )
: m_num ( t.m_num ), m_ps ( new CStation ( *t.m_ps ) ) { }
CTrain :: ~CTrain ( ) { delete m_ps ; }
```

Коли об'єкт іншого класу оголошується через вказівник , для його створення потрібно динамічне виділення пам'яті, яке і виконують конструктори класу CTrain . Деструкція класу руйнує об'єкт, виконуючи операцію delete для покажчика m_ps .

Результат виконання цієї версії програми буде ідентичний версії з явним оголошенням об'єкта.

Лістинг.Приклад оголошення частини через посилання

```
#include "Station.h"

class CTrain          // опис рейсу
```

```

{
public:
    int m_num ;           // номер поїзда
    CStation& m_rs; // посилання на станцію відправлення
    CTrain () ; CTrain ( int, char* ) ; CTrain ( CTrain& ) ;
    ~CTrain () ;
};

// Train.cpp реалізація класу
CTrain :: CTrain ( ) : m_num ( 0 ), m_rs ( * ( new CStation ) ) { }
CTrain :: CTrain ( int num, char* p )
: m_num ( num ), m_rs ( * ( new CStation ( p ) ) ) { }
CTrain :: CTrain ( CTrain& t )
: m_num ( t.m_num ), m_rs ( * ( new CStation ( t.m_rs ) ) ) { }
CTrain :: ~CTrain ( ) { delete &m_rs ; }

```

У прикладі показані зміни специфікації і реалізації класу CTrain. Решта вихідного коду залишився таким же, як у прикладі з явним оголошенням .

Коли об'єкт іншого класу оголошується через посилання , необхідно створювати вміст за адресою, якого поки не існує. Тому конструктори класу CTrain використовують розіменування покажчика, що повертається оператором динамічного виділення пам'яті new. У деструкції класу для руйнування створеного об'єкта операція delete отримує його адресу.

Результат виконання програми буде аналогічний версії з явним оголошенням об'єкта (лістинг 1) .

Головна функція за допомогою функції view () виведе наступні рядки:

Train number: 0 Departure station:

Train number: 2 Departure station: Kiev

Train number: 2 Departure station: Kiev

Першим буде виконуватися конструктор за замовчуванням класу CStation для створення члена-даного класу CTrain з ідентифікатором m_sd. Потім продовжує свою

роботу конструктор за замовчуванням класу CTrain, який створює об'єкт t, стан якого виводиться функцією view (). На наступному кроці виконання виділяється блок оперативної пам'яті, необхідний для розміщення об'єкта типу класу CTrain. Щоб створити цей об'єкт, викликається конструктор з параметром "Kiev" класу CStation і конструктор з параметрами класу CTrain завершує формування об'єкта. Стан створеного об'єкта виведено у другому рядку.

Нарешті, за допомогою виразу CTrain copy (* p) створюється копія тільки що створеного об'єкта, послідовно викликаючи конструктори копіювання класів cstation і CTrain. Стан копії також виводиться. В кінці головної функції для руйнування об'єкта, на який вказує p, виконується оператор delete, викликаючи деструктори класів CTrain і cstation. Перед виходом з програми автоматично будуть викликані деструктори ~ CTrain () і ~ CStation ().

Висновки

Коли об'єкт іншого класу оголошується через посилання, необхідно створювати вміст за адресою, якого поки не існує. Тому конструктори класу CTrain використовують розіменування покажчика, що повертається оператором динамічного виділення пам'яті new. У деструкції класу для руйнування створеного об'єкта операція delete отримує його адресу. Результат виконання програми буде аналогічний версії з явним оголошенням об'єкта.

Питання та завдання до контролю знань студентів

1 Для актуалізації опорних знань та перевірки попереднього матеріалу

1. Що таке агрегація класів?
2. Що таке композиція класів?
- 3 Як створюються об'єкти при агрегації?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Які є способи включення об'єкту-частки у об'єкт-ціле?

2 Як працюють конструктори та деструктори при агрегації?

Література

Довбуш, Г. Ф. Visual C++ на прикладах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под
ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

Тема

Відношення між класами. Діаграми класів

Мета заняття

Розглянути види відношень між класами та навчитись будувати
діаграми класів

Успадкування застосовується для того, щоб забезпечити похідний клас властивостями одного або більше базових. Частіше всього один з цих класів є основним, а інші забезпечують деякі додаткові властивості, тому вони називаються класами підмішування. По можливості класи підмішування мають бути віртуальними і створюватися за допомогою конструкторів без параметрів, що дозволяє уникнути деяких конфліктів.

Крім успадкування класи можуть бути пов'язані відношенням агрегації або композиції, враховуючи доцільність відношення об'єкт-ціле та об'єкт-частка. В такому випадку говорять про відношення "містить", тобто, об'єкт-ціле містить об'єкти-частки. В залежності від відношення класи отримують доступ до елементів один одного за правилами, відповідно, успадкування або агрегації.

Відношення між класами. Діаграми класів

Яким чином пов'язати між собою класи, треба вирішувати за результатами аналізу предметної області та проектування відношень. Наприклад, при розгляді об'єктів Людина - Студент ми не можемо говорити, що студент містить людину, або людина містить студента, але можемо говорити, що студент є людиною, таким чином, у випадку успадкування клас Студент отримає за правилами елементи класу Людина. Навпаки, у відношенні Обід та Фрукт ми не говоримо, що обід є фруктом, або фрукт є обідом, але говоримо, що обід може містити фрукти, тобто, в даному випадку доцільним відношення буде агрегація за вказівником (тому що обід може бути і без фруктів, в такому випадку вказівник буде порожній).

Множинне успадкування означає, що клас має декілька базових класів. Якщо у базових класах є однойменні елементи, при цьому може відбутися конфлікт ідентифікаторів, який усувається за допомогою операції доступу до зони видимості :

```

class monstr{
public: int get_health( );
};

class hero{
public: int get_health( );
};

class ostrich: public monstr, public hero{
...
};

int main( ){
ostrich A;
cout << A.monstr::get_health( );
cout << A.hero::get_health( );
}

```

Як видно з прикладу, для виклику методу `get_health` вимагається явно вказувати клас, в якому він описаний. Використання звичайної для виклику методу класу конструкції `A.get_health()` приведе до помилки, оскільки компілятор не в змозі розібратися, до методу якого з базових класів потрібно звернутися.

Якщо у базових класів є загальний предок, це приведе до того, що похідний від цих базових клас успадкує два екземпляри полів предка, що найчастіше є небажаним. Щоб уникнути такої ситуації, вимагається при спадкоємстві загального предка визначити його як віртуальний клас:

```

class monstr{
...
};

class daemon: virtual public monstr{
...
};

class lady: virtual public monstr{
...
};

class baby: public daemon, public lady{
...
};

```

Похідний клас, у свою чергу, сам може служити базовим класом. Такий набір пов'язаних класів традиційно називається ієрархією класів. Ієрархія найчастіше є деревом, але може мати і загальнішу структуру графа.

Доступність в похідному класі регулюється ключем доступу до базового класу, що вказується в оголошенні похідного класу. Цей ключ визначає вид спадкоємства : відкрите (public), захищене (protected) або закрите (private).

Відкрите спадкоємство зберігає статус доступу усіх елементів базового класу, захищене, - знижує статус доступу public елементів базового класу до protected, і нарешті, закрите - знижує статуси доступу public і protected елементів базового класу до private. Помітимо, що в C++ відношення між класами "is a" має місце тільки при відкритому спадкоємстві.

Діаграма класів є ключовим елементом в об'єктно-орієнтованому моделюванні. На діаграмі класи представлені в рамках, що містять три компоненти.

У верхній частині написано ім'я класу. Ім'я класу вирівнюється по центру і пишеться напівжирним шрифтом. Імена класів починаються з великої літери. Якщо клас абстрактний - то його ім'я пишеться напівжирним курсивом. Але ця домовленість не є обов'язковою.

Посередині розташовуються поля (атрибути) класу. Вони вирівняні по лівому краю і починаються з маленької літери.

Нижня частина містить методи класу. Вони також вирівняні по лівому краю і пишуться з малої літери.

Приклад наведений на рисунку 42.1.

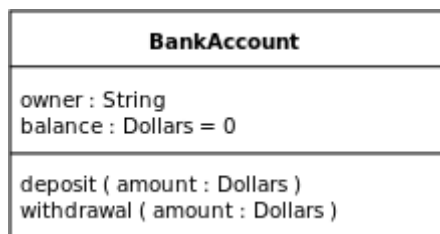


Рис. 42.1 – Клас з двома полями та методами

Взаємозв'язок - це особливий тип логічних відносин між сутностями, показаних на діаграмах класів і об'єктів. В UML представлені наступні види відносин:

Залежність позначає таке відношення між класами, що зміна специфікації одного класу може вплинути на роботу залежного класу, але не навпаки.

Асоціація показує, що об'єкти однієї сутності (класу) пов'язані з об'єктами іншої сутності таким чином, що можна переміщатися від об'єктів одного класу до іншого. Є загальним випадком композиції і агрегації.

Наприклад, клас Людина і клас Школа мають асоціацію, так як людина може вчитися в школі. Асоціації можна присвоїти ім'я «вчиться в».

Агрегація - це різновид асоціації при відношенні між цілим і його частинами. Як тип асоціації агрегація може бути іменованою. Одне ставлення агрегації не може включати більше двох класів (контейнер і вміст).

Агрегація зустрічається, коли один клас є колекцією або контейнером інших. Причому за замовчуванням, агрегацією називають агрегацію за посиланням, тобто коли час існування об'єктів класів частин не залежить від часу існування об'єкту класу цілого.

Графічно агрегація представляється порожнім ромбом на блоці класу, і лінією, що йде від цього ромба до вмісту класу.

Композиція - більш суворий варіант агрегації. Відома також як агрегація за значенням.

Композиція має жорстку залежність часу існування примірників класу контейнера і примірників містяться класів. Якщо контейнер буде знищений, то весь його вміст буде також знищено.

Графічно представляється як і агрегація, але з зафарбованим ромбиком.

Наприклад, кімната є частиною квартири, отже тут підходить композиція, тому що кімната без квартири існувати не може. А, наприклад, меблі не є невід'ємною частиною квартири, але в той же час, квартира містить меблі, тому слід використовувати агрегацію.

Приклади зображення відношень класів наведені на рисунках 42.2-42.3.



Рис. 42.2 – Нотація UML для відображення взаємозв'язку між класами на діаграмах

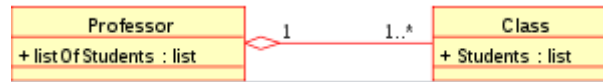


Рис. 42.3 – Приклад зображення агрегації на діаграмі

Висновки

Питання та завдання до контролю знань студентів

- 1 Для актуалізації опорних знань та перевірки попереднього матеріалу
 1. Як описуються та використовуються дружні класи?
 2. Як описуються та використовуються вкладені класи?
 3. Як описується включення об'єкту-частки у об'єкт-ціле?
- 2 Для узагальнення та перевірки засвоєного матеріалу на занятті
 1. Як можуть бути пов'язані між собою класи?
 2. Як в діаграмі класів зображується відношення успадкування?
 3. Як в діаграмі класів зображуються відношення агрегації та композиції?

Література

Лафоре, Р. Объектно-ориентированное программирование в С++ [Текст] / Р. Лафоре. – М. - СПб. : ПИТЕР, 2004.- 923 с.

Тема	Створення одновіконного візуального додатку
Мета заняття	Навчитись створювати одновіконні візуальні додатки

Створення одновіконного візуального додатку розглядається на прикладі середовища C++ Builder або Qt Creator.

Ознайомитись з середовищем, спроектувати візуальну форму.

Порядок виконання роботи.

1 В Embarcadero C++ Builder створити новий проект за допомогою меню File - > New, вибрати VCL Forms Application (або Windows VCL Application - C++ Builder).

2 Для виконання завдання використати різні види компонентів за сенсом: написи, редактори, списки, перемикачі, кнопки. За допомогою Палітри компонентів розмістити на візуальній формі необхідні компоненти, наприклад:

– TLabel — надпис;

– TEdit — однорядковий текстовий редактор для введення та редагування тексту під час виконання додатку;

– TMemo — багаторядковий текстовий редактор;

– TButton — кнопка для завдання дій керування додатком, таких як "Розрахувати", "Очистити" та ін.;

– TListBox — список, для вибору елементів зі списку, який можна задавати під час проектування або змінювати програмно;

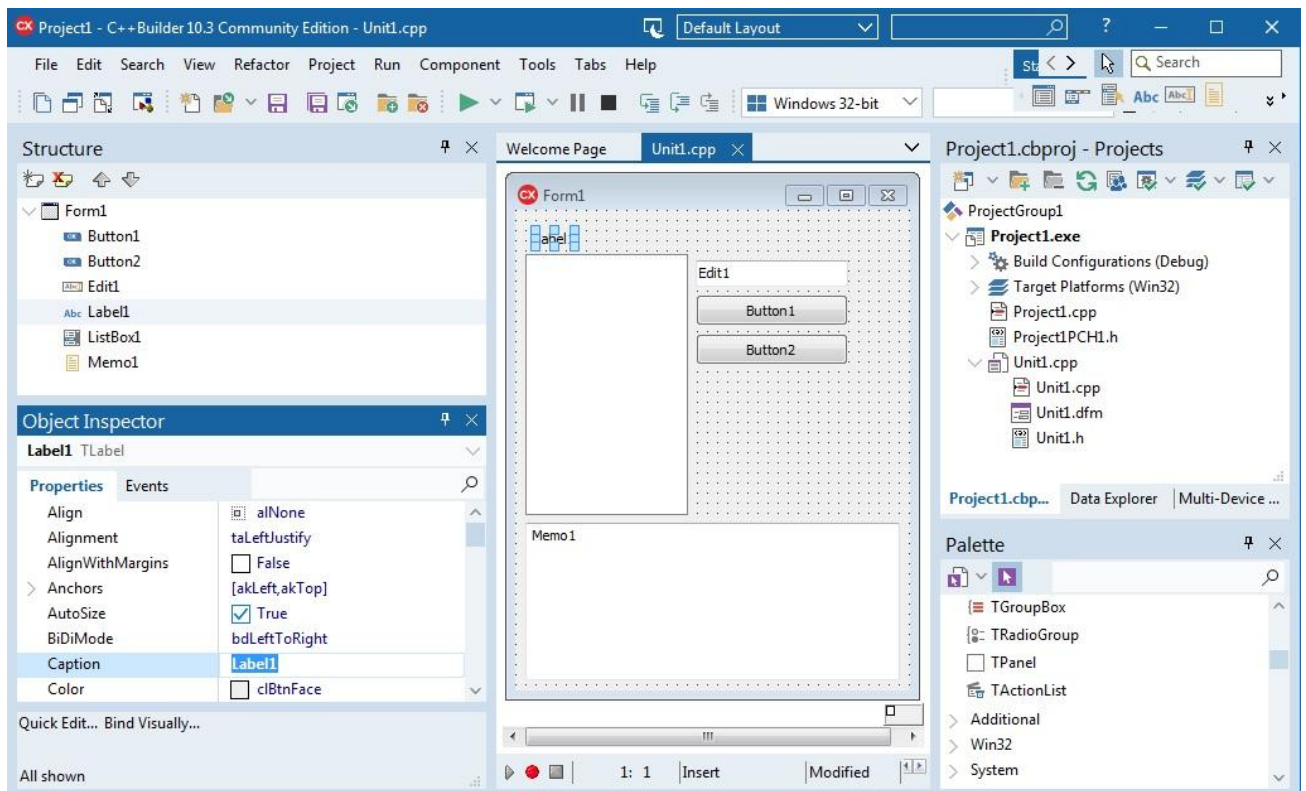
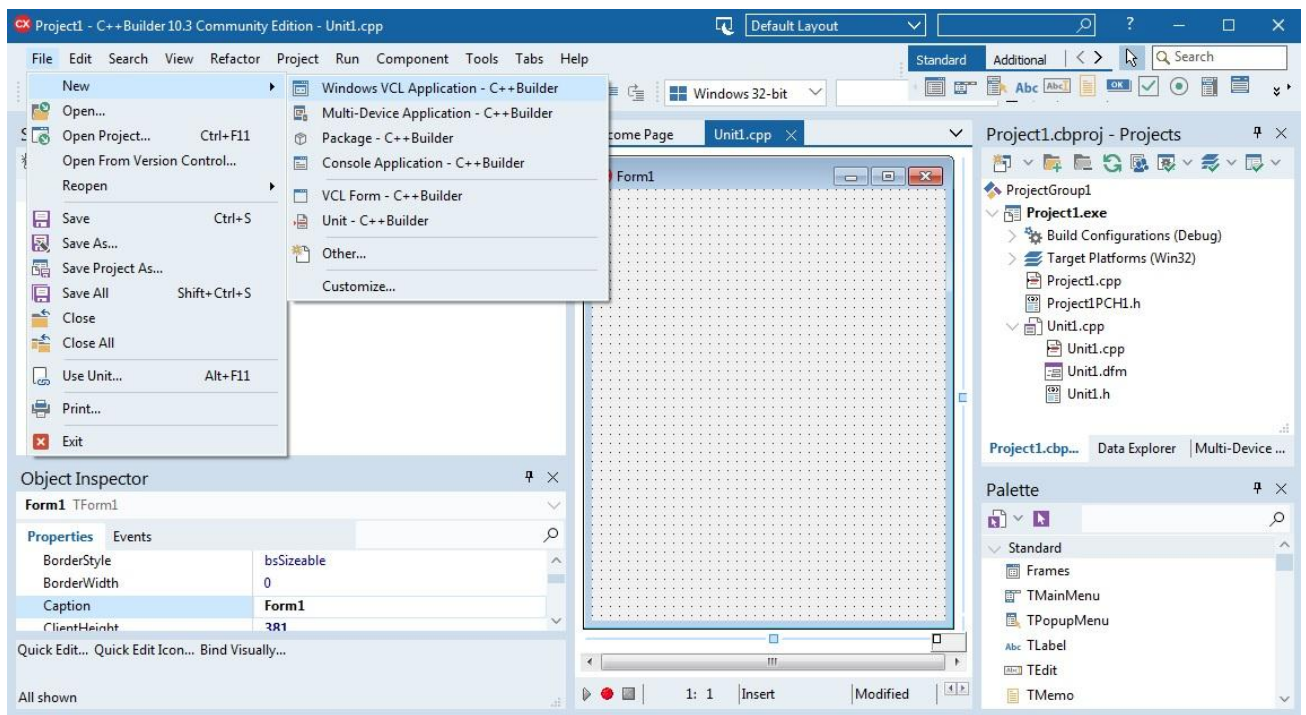
– TCheckBox — прапорець, використовується як незалежний перемикач (за бажанням);

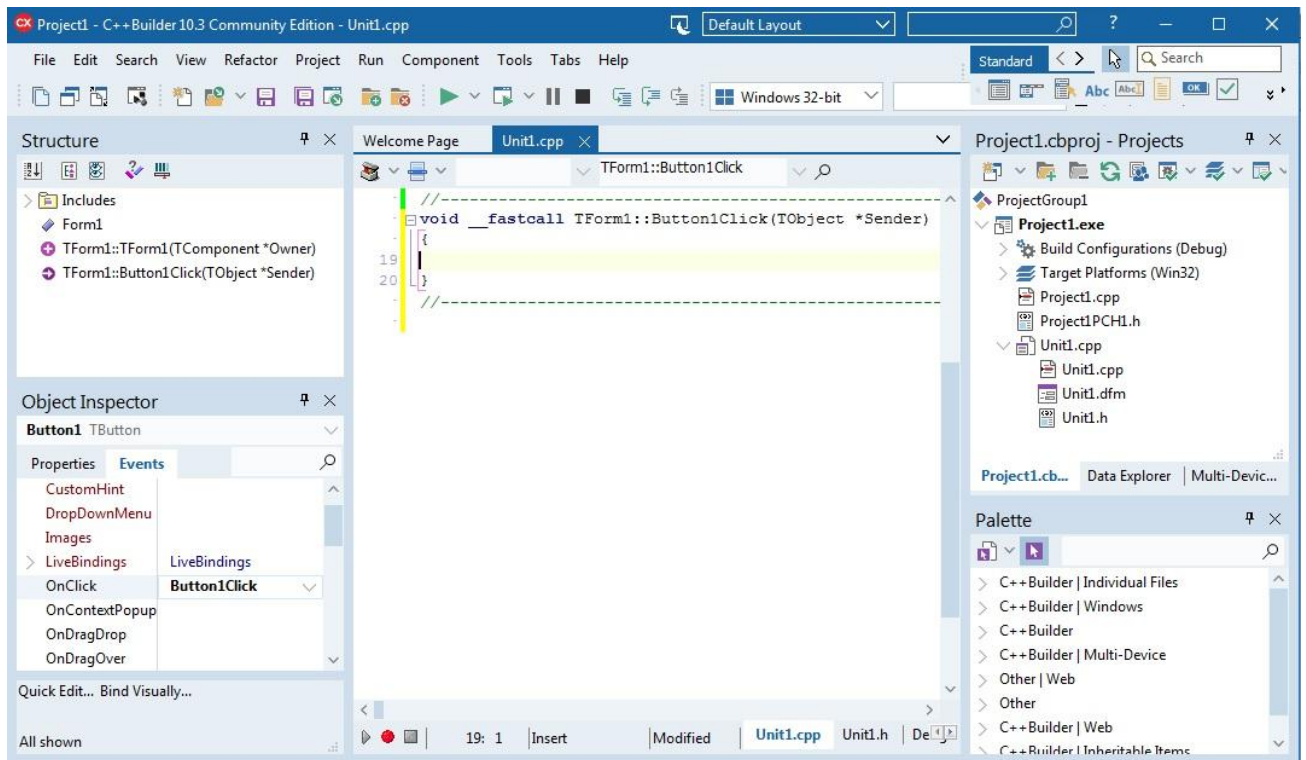
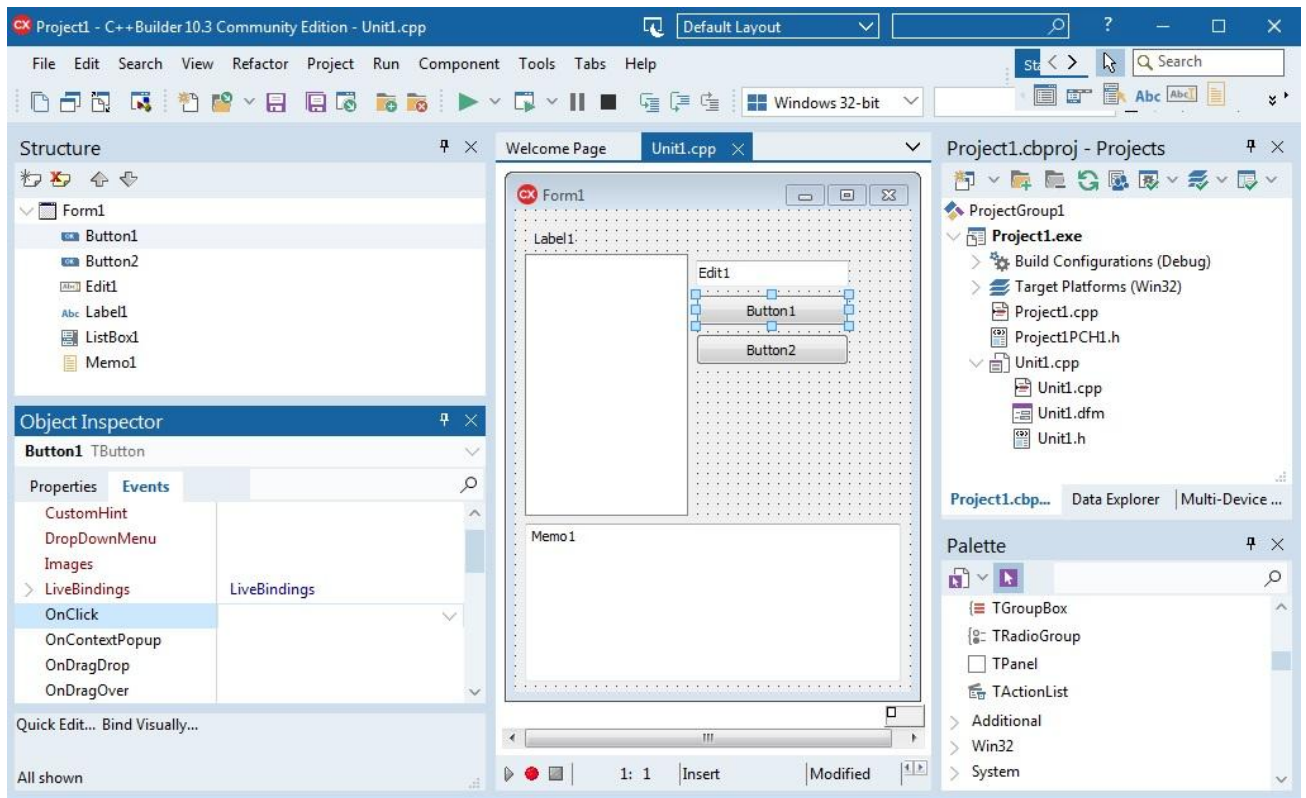
– TComboBox — випадаючий список — комбінація поля та списку, що дозволяє як вибирати елементи зі списку, який можна змінювати програмно, так і вводити текст у відповідному полі (за бажанням);

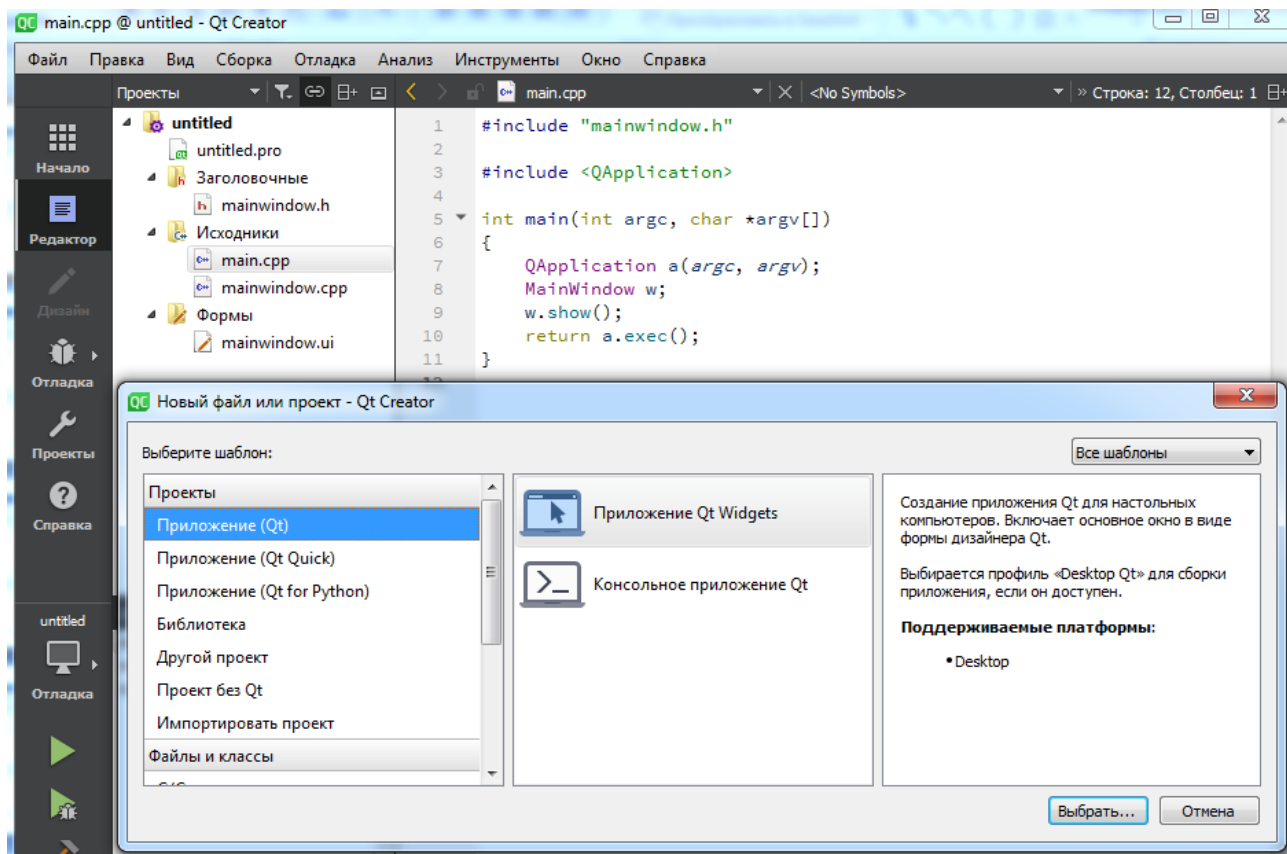
– TRadioGroup — група залежних перемикачів (за бажанням).

Щоб розмістити компонент на формі, потрібно клацнути по компоненту на Палітрі компонентів, а потім — в тому місці форми, де треба розмістити компонент.

З використанням Інспектора об'єктів задати значення властивостям візуальних об'єктів. Наприклад, для напису це властивість Caption="текст".







Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на занятті

1. Як створити візуальний додаток?
2. Охарактеризувати компоненти за сенсом: написи, редактори.
3. Охарактеризувати компоненти за сенсом: списки, перемикачі, кнопки.

Література

Лафоре, Р. Объектно-ориентированное программирование в С++ [Текст] / Р. Лафоре. – М. - СПб. : ПИТЕР, 2004.- 923 с.

Тема

Робота з одновіконним візуальним додатком

Мета заняття

Закріпити навички роботи з одновіконним візуальним додатком

Створення одновіконного візуального додатку розглядається на прикладі середовища C++ Builder або Qt Creator.

Для об'єктно-орієнтованих візуальних додатків доцільно розподіляти логіку інтерфейса, призначену для введення/виведення інформації, та логіку предметної області.

Порядок виконання роботи.

Доцільно проектувати одновіконний візуальний додаток та власний клас, що описує предметну область, в окремому модулі (не в класі візуального вікна) за предметною областю, для цього:

- проаналізувати постановку задачі за варіантом;
- сформулювати функції, які повинна виконувати програма, яка розробляється;
- виділити вхідну і вихідну інформацію, виявити типи даних та компоненти, які доцільно їм відповідають;
- спроектувати та розробити програму, рекомендовано підключити власний клас як окремий модуль, всю логіку за предметною областю винести в клас.

Введені значення рекомендовано хоча б в мінімальному обсязі перевіряти на допустимість.

Врахувати, що програма повинна допускати повторне введення значень та отримання результатів. Забезпечити, щоб на візуальній формі біло видно, які дані були введені, та який отриманий результат (наприклад, вносити в багаторядковий текстовий редактор).

```

TForm2 *Form2;
float sum;
//-----
void __fastcall TForm2::btAddClick(TObject *Sender)
{
    if ((lbGoods->ItemIndex==-1) || (edCount->Text.Length()==0)) {
        ShowMessage("Не всі поля заповнені");    }
    else {
        memBasket->Lines->Add(lbGoods->Items->Strings[lbGoods->ItemIndex]);
        int k=StrToInt(edCount->Text);
        switch (lbGoods->ItemIndex) {
            case 0: sum+=15*k; break; ↵
            case 1: sum+=40*k; break; ↵
            case 2: sum+=25*k; break; ↵
            case 3: sum+=10*k; break; ↵
            case 4: sum+=100*k; break; ↵
        }
        edSum->Text=FloatToStr(sum);    } }
//-----
void __fastcall TForm2::FormCreate(TObject *Sender) { sum=0; }
//-----
void __fastcall TForm2::btnBuyClick(TObject *Sender)
{ int k=0;
  if (sum>500) { k=3; } else if (sum>300) { k=2; } else if (sum>100) { k=1; }
  if (k>0) { sum=sum*(100-k)/100; }
  edPercent->Text=IntToStr(k);
  edPay->Text=FloatToStrF(sum, ffFixed, 5, 2);
  edCount->Clear();
  memBasket->Clear();
}

```

```

TForm2 *Form2;
float sum;
MyBasket o;
//-----
void __fastcall TForm2::btAddClick(TObject *Sender)
{
    if ((lbGoods->ItemIndex==-1) || (edCount->Text.Length()==0)) {
        ShowMessage("Не всі поля заповнені");    }
    else {
        int i=lbGoods->ItemIndex;
        memBasket->Lines->Add(lbGoods->Items->Strings[i]);
        o.goods[i].count=StrToInt(edCount->Text);
        o.sum+=o.goods[i].price*o.goods[i].count;
        edSum->Text=FloatToStr(o.sum);    } }
//-----
void __fastcall TForm2::FormCreate(TObject *Sender) { sum=0;
    for (int i=0; i < goodsCount; i++) {
        lbGoods->Items->Add(o.goods[i].name);
    }
}
//-----
void __fastcall TForm2::btnBuyClick(TObject *Sender)
{
    int k=0;
    o.Pay(k);
    edPercent->Text=IntToStr(k);
    edPay->Text=FloatToStrF(o.sum, ffFixed, 5, 2);
    edCount->Clear();
    memBasket->Clear();
}

```

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на занятті

1. Як розподіляється логіка у візуальному додатку?
2. Яка логіка реалізується в інтерфейсі?
3. Яка логіка реалізується в класі?

Література

Лафоре, Р. Объектно-ориентированное программирование в С++ [Текст] / Р. Лафоре. – М. - СПб. : ПИТЕР, 2004.- 923 с.

Тема	Використання власних класів в візуальному додатку
Мета заняття	Навчитись використовувати власні класи в візуальному додатку

Для об'єктно-орієнтованих візуальних додатків доцільно розподіляти логіку інтерфейса, призначену для введення/виведення інформації, та логіку предметної області. Логіка предметної області реалізується у власних класах, де описуються основні характеристики та поведінка об'єктів реального світу.

Порядок виконання роботи.

Додати в проєкт новий модуль. Будуть додані два файли: *.h, *.cpp, де відповідно проєктується специфікація та реалізація класів. Враховуючи, що об'єкт такого класу доцільно створювати в модулі реалізації візуального вікна, підключаємо його директивою include.

Для проєктування власного класу, що описує предметну область, в окремому модулі (не в класі візуального вікна):

- проаналізувати постановку задачі за варіантом;
- сформулювати функції, які повинна виконувати програма, яка розробляється;
- виділити вхідну і вихідну інформацію, виявити типи даних та компоненти, які доцільно їм відповідають;
- спроектувати та розробити програму, рекомендовано підключити власний клас як окремий модуль, всю логіку за предметною областю винести в клас.

```

struct basket {
    AnsiString name;
    float price;
    int count;
    basket() { count=0; }
};

const int goodsCount=5;
class MyBasket {
public:
    basket goods[goodsCount];
    float sum;
    MyBasket(): sum(0) {
        goods[0].name="макарони, 1 уп."; goods[0].price=15;
        goods[1].name="масло, 1 уп."; goods[1].price=40;
        goods[2].name="молоко, 1 пак."; goods[2].price=25;
        goods[3].name="хліб, 1 батон"; goods[3].price=10;
        goods[4].name="цукерки, 1 кг"; goods[4].price=100;
    }
    float Pay(int &k) {
        if (sum>500) { k=3; } else if (sum>300) { k=2; } else if (sum>100) { k=1; }
        if (k>0) { sum=sum*(100-k)/100; }
        return sum;
    }
};

```

```

TForm2 *Form2;
float sum;
MyBasket o;
//-----
void __fastcall TForm2::btAddClick(TObject *Sender)
{
    if ((lbGoods->ItemIndex==-1) || (edCount->Text.Length()==0)) {
        ShowMessage("Не всі поля заповнені");    }
    else {
        int i=lbGoods->ItemIndex;
        memBasket->Lines->Add(lbGoods->Items->Strings[i]);
        o.goods[i].count=StrToInt(edCount->Text);
        o.sum+=o.goods[i].price*o.goods[i].count;
        edSum->Text=FloatToStr(o.sum);    } }
//-----
void __fastcall TForm2::FormCreate(TObject *Sender) { sum=0;
    for (int i=0; i < goodsCount; i++) {
        lbGoods->Items->Add(o.goods[i].name);
    }
}
//-----
void __fastcall TForm2::btnBuyClick(TObject *Sender)
{
    int k=0;
    o.Pay(k);
    edPercent->Text=IntToStr(k);
    edPay->Text=FloatToStrF(o.sum, ffFixed, 5, 2);
    edCount->Clear();
    memBasket->Clear();
}

```

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на занятті

1. Як розподіляється логіка у візуальному додатку?
2. Як підключити модуль класу у візуальний додаток?
3. Яка логіка реалізується в класі?

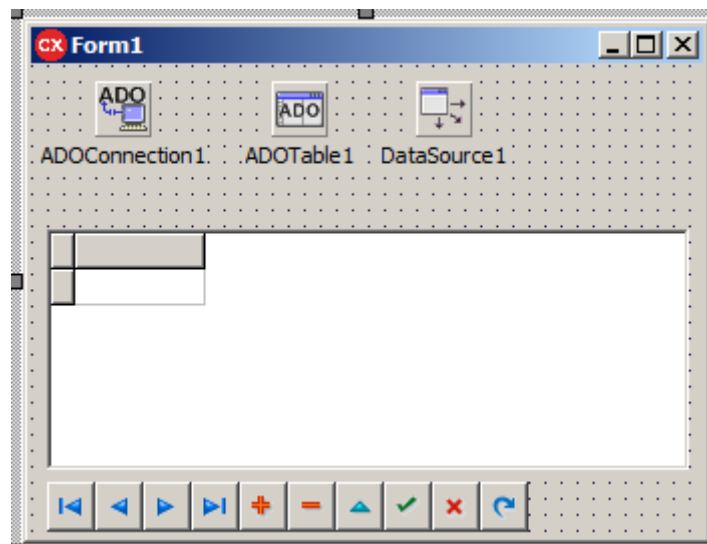
Література

Лафоре, Р. Объектно-ориентированное программирование в С++ [Текст] / Р. Лафоре. – М. - СПб. : ПИТЕР, 2004.- 923 с.

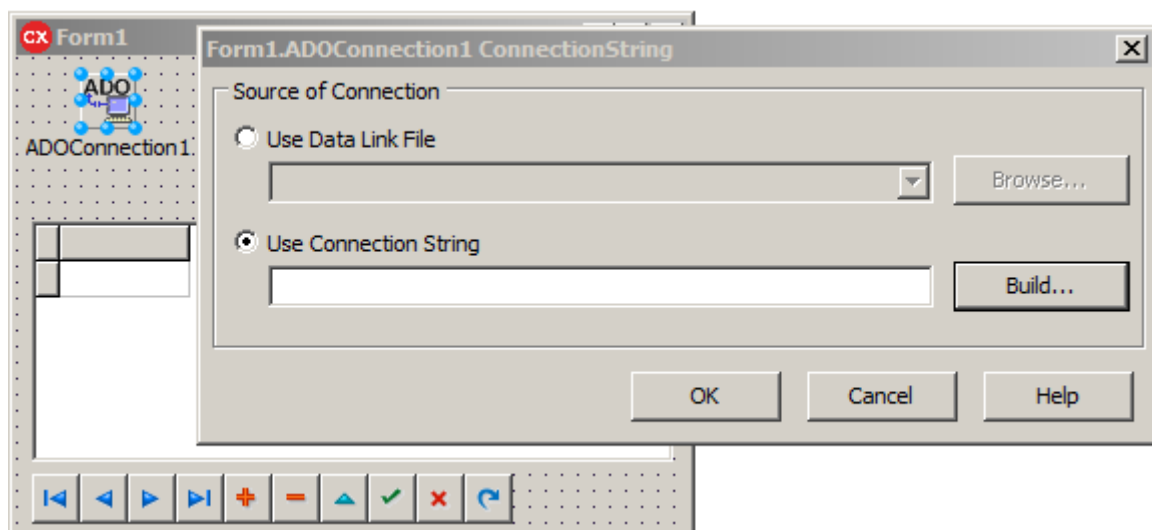
Тема	Створення підключення до бази даних у візуальному додатку
Мета заняття	Навчитись створювати підключення до бази даних у візуальному додатку

Для навчання можна використати БД, яка встановлена з Embarcadero
C:\Users\Public\Documents\Embarcadero\Studio\19.0\Samples\Data\dbdemos.mdb

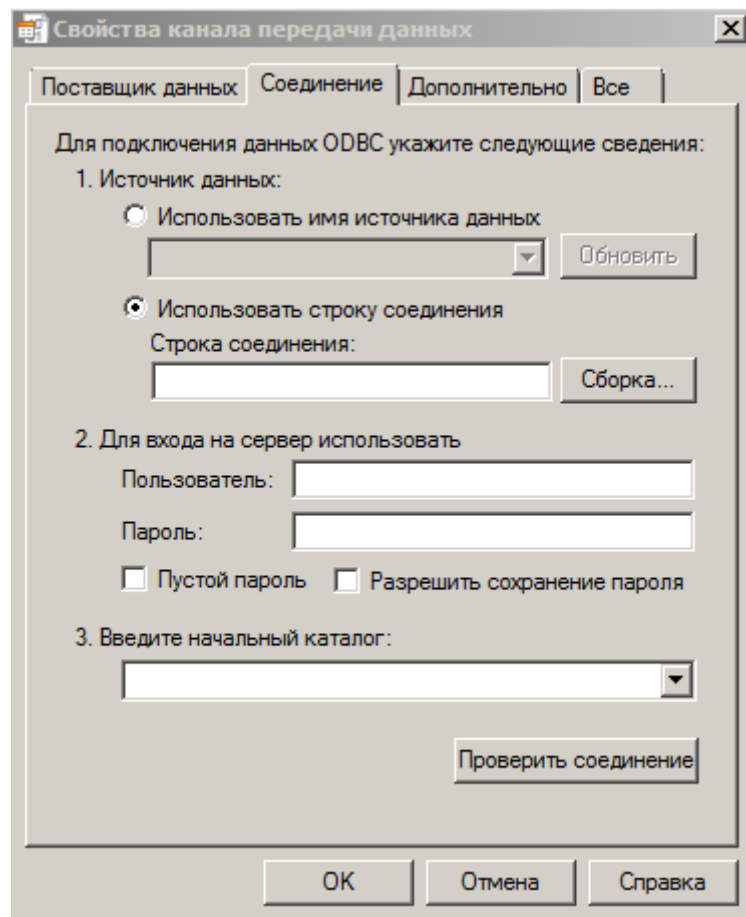
Розмістіть на формі компоненти TADOConnection, TADOTable, TDataSource, TDBGrid, TDBNavigator



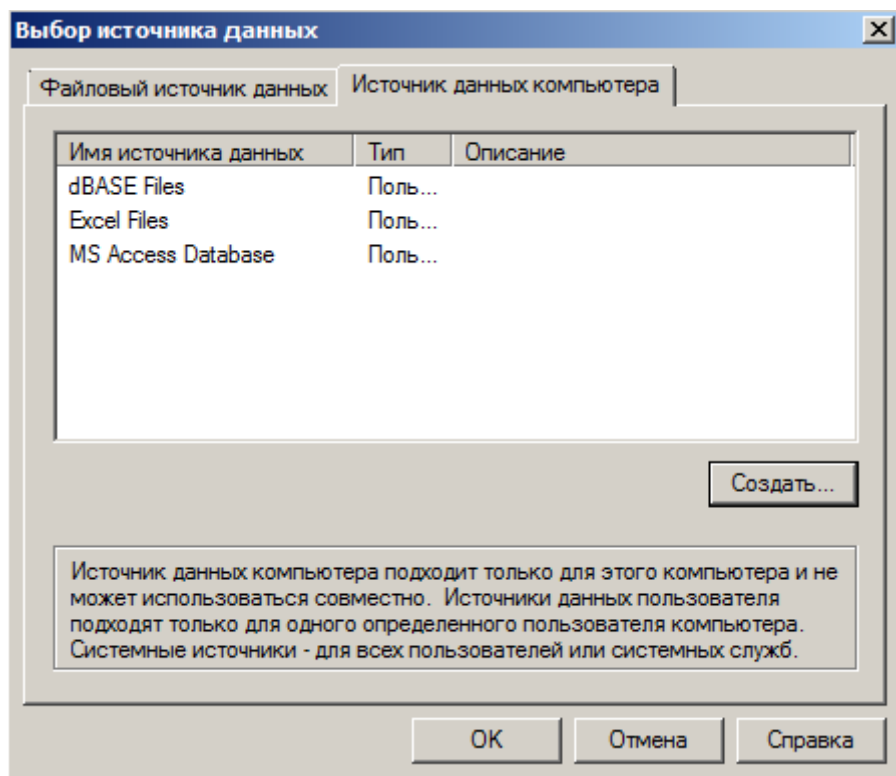
За допомогою Інспектора об'єктів встановіть властивість ADOConnection1->ConnectionString, як наведено нижче (Use Connection String (Build)).



Виберіть «Соединение», «Использовать строку соединения», «Сборка»



Виберіть «Источник данных компьютера», «Создать»



Microsoft Access Driver => Вказати повний шлях і ім'я файлу БД.

Перевірити з'єднання натисканням відповідної кнопки.

Рекомендовано встановити властивість ADOConnection1->LoginPrompt=False, щоб не було щоразу запиту логіна та паролю і встановити ADOConnection1->Active=True.

Для ADOTable1 встановити ADOTable1->Connection=ADOConnection1;

ADOTable1->TableName= вибрати необхідну таблицю

ADOTable1-> Active=True.

Встановити:

DataSource1->DataSet= ADOTable1;

DBGrid1-> DataSource= DataSource1;

DBNavigator1-> DataSource= DataSource1;

Для перебору логічного набору даних можна використати:

ADOTable1->First() – перейти на перший запис

ADOTable1->Next() – перейти на наступний запис

ADOTable1->Last() – перейти на останній запис

ADOTable1->Eof – істина, якщо кінець набору

ADOTable1->FieldByName(“ім’я поля”)->AsString (або AsInteger) – вибрати значення

Як редагувати запис по натисканню на кнопку форми:

ADOTable1->Edit();

ADOTable1->FieldByName(“ім’я поля”)->AsFloat = 1.5; // AsString (або AsInteger)

Поля...

ADOTable1->Post();

Як додати запис по натисканню на кнопку форми:

```
ADOTable1->Insert();
```

```
ADOTable1->FieldByName("ім'я поля")->AsFloat = 1.5; // AsString (або AsInteger)
```

Поля...

```
ADOTable1->Post();
```

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на занятті

1. Як підключити базу даних у візуальний додаток?
2. Які компоненти використати для підключення?
3. Як організувати виведення даних у візуальний компонент?

Література

Лафоре, Р. Объектно-ориентированное программирование в С++ [Текст] / Р. Лафоре. – М. - СПб. : ПИТЕР, 2004.- 923 с.

Тема	Технології візуального об'єктно-орієнтованого та подійно-орієнтованого програмування
-------------	---

Мета заняття	Розглянути технології візуального об'єктно-орієнтованого та подійно-орієнтованого програмування
---------------------	---

Візуальне програмування - спосіб створення програм шляхом маніпулювання графічними об'єктами замість написання коду в текстовому вигляді.

Візуальне програмування дозволяє програмувати, використовуючи графічні або символічні елементи, якими можна маніпулювати інтерактивним чином згідно з деякими правилами.

Технології візуального об'єктно-орієнтованого та подійно-орієнтованого програмування

Значна кількість сучасних мов програмування має розвинені візуальні засоби для розробки графічного інтерфейсу, причому здійснюється програмування розміщених на спеціальних формах об'єктів з налаштуванням їх властивостей і поведінки.

Візуальне програмування є в даний час однією з найбільш популярних парадигм програмування. Візуальне програмування полягає в автоматизованій розробці програм з використанням особливої діалогової оболонки. Розглядаючи системи візуального програмування, легко побачити, що всі вони базуються на об'єктно-орієнтованому програмуванні і є його логічним продовженням. Найбільш часто візуальне програмування використовується для створення інтерфейсу програм і систем управління базами даних.

При виборі одного з класів в спеціальних вікнах відображаються його локальні функції і змінні. Потім при виборі одного з методів на окремій панелі висвічується його код. Зазвичай в системі присутні засоби для додавання і видалення класів з проекту.

Структурною одиницею візуального програмування є компонент. Компонент являє собою різновид об'єкта, який можна перенести (агрегувати) в додаток зі

спеціальної Палітри компонент. Компонент має набір властивостей, які можна змінювати, не змінюючи вихідний код програми.

Компоненти бувають візуальними і невізуальними. Перші призначені для організації інтерфейсу з користувачем. Це різні кнопки, списки, статичний і редагований текст, зображення і багато іншого. Ці компоненти відображаються при виконанні розробляється. Невізуальні компоненти відповідають за доступ до системних ресурсів: драйверам баз даних, таймерам і т. д. Під час розробки вони відображаються своєю піктограмою, але при виконанні програми, як правило, невидимі.

Формою називається візуальний компонент, що володіє властивістю вікна Windows. При розробці на формі поміщаються необхідні компоненти (наприклад, елементи необхідного діалогу). Форм в додатку може бути декілька - по необхідному числу відкриваються при виконанні діалогу вікон, їх можна додавати і видаляти.

Клацнувши по зображенню компонента на формі, можна зробити його активним. Потім, переміщаючи за допомогою миші кордону кнопки і працюючи в вікні властивостей та подій, можна задати напис (наприклад, ОК) і / або графічну піктограму на кнопці, задати колір і інші параметри налаштування кнопки. Подвійне клацання по кнопці - і в початковому тексті форми з'явиться шаблон підпрограми (методу) потрібного типу реакції на клацання. Працюючи у вікні редактора тексту, можна оформити тіло підпрограми реакції кнопки на клацання.

Середовище дозволяє входити в вихідні тексти методів (підпрограми обробки подій, названих Events), наприклад, на натискання Enter, а також задавати початкові значення полів даних, названих Properties.

Початкові кроки технології візуального програмування визначаються оболонкою самого середовища візуального програмування. Спочатку створюються екранні форми, проводиться настройка їх властивостей шляхом заповнення окремих полів. На головну форму крім візуальних компонент наносяться невізуальні компоненти. Форми об'єднуються в єдиний проект. Далі відповідно до сценарію діалогу програмуються методи події основний і підлеглих форм.

Середовище візуального програмування, наприклад MS Visual Studio - це система об'єктно-орієнтованого візуального програмування.

В основі її лежить технологія візуального проектування і подієвого програмування, суть якої полягає в тому, що середовище розробки бере на себе більшу частину рутинної роботи, залишаючи програмісту роботу з конструювання діалогових вікон і функцій обробки подій.

Додатки (прикладні програми) Delphi є інтерактивними системами, в яких для організації взаємодії між користувачем і програмою використовуються методи (підпрограми), керовані подіями.

Основою об'єктно-орієнтованого програмування є об'єкт. Об'єкт можна визначити як сукупність даних (компонентів) і методів роботи з ними. Об'єктно-орієнтована програма - це сукупність об'єктів і способів їх взаємодії. Обмін інформацією між об'єктами проводиться за допомогою повідомлень. Повідомлення є результатом появи подій.

Подія - це відгук на зовнішній вплив. Суть програмування, керованого подіями, складається у відстеженні таких подій, які вимагають реакції додатки.

Об'єктно-орієнтоване програмування (ООП, OOP - Object Oriented Programming) дозволяє вирішувати складні завдання за допомогою об'єктів, необхідних для програмування в Windows. Використання елементів візуальної розробки автоматично веде до застосування ООП.

Об'єкти - це елементи, з яких будується додаток: форма, рамка, кнопка, мітка і ін. Об'єктом є і включається візуальний компонент (наприклад, кнопка) в тому вигляді, як він представлений під час поміщення його на форму і під час виконання програми. Об'єкти зображуються на екрані до виконання самої програми. Тому програмування - візуальне.

Основним вікном, що розробляється є форма. У процесі розробки програми при розміщенні об'єкта на формі (наприклад, кнопки) у візуальному середовищі основні параметри об'єкта (розмір, положення на екрані, колір та ін.) Відразу відображається у вигляді реального компонента на формі, а відповідний йому код автоматично записується в вихідний файл форми, який відображає об'єкт в процесі виконання програми.

Властивості - це атрибути (основні характеристики), які описують особливості об'єкта; наприклад, відображають такі характеристики, як колір, висота, ширина і

місце розташування об'єкта. На зовнішній вигляд об'єкта можна впливати (змінювати його) під час розробки та виконання програми, змінюючи його властивості.

Практично всі об'єкти реагують на певні події від миші і клавіатури.

Додатки Windows використовують методи (підпрограми) обробки подій для управління взаємодією між програмою і користувачем і для реакції на дії ОС. Програмний код, який пише програміст, буде забезпечувати реакцію на події. Підпрограма, яка реагує на подія, називається обробником події. Середовище працює з подіями шляхом виклику певних процедур - обробників подій. Якщо процедура не пов'язана з цією подією, то воно ігнорується і виконується стандартна реакція системи або не проводиться ніякої дії.

Висновки

Візуальне програмування багато в чому автоматизує працю програміста з написання програм.

Візуальне програмування - одна з найпопулярніших парадигм програмування на даний момент. Воно базується на технології ООП.

Середовище візуального програмування підтримує роботу засобів, за допомогою яких можна автоматично отримати документацію по структурі програми.

Основним елементом в засобах візуального програмування є компонент. Компоненти бувають візуальними і невізуальними.

Технологія візуального програмування полягає в наступному: створення екранних форм, нанесення візуальних і невізуальних компонент, програмування подій і методів віконних форм.

Питання та завдання до контролю знань студентів

1 Для актуалізації опорних знань та перевірки попереднього матеріалу

1. Як можуть бути пов'язані між собою класи?
2. Як в діаграмі класів зображується відношення успадкування?
3. Як в діаграмі класів зображуються відношення агрегації та композиції?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

1. Назвіть основні кроки технології візуального програмування.
2. Що таке компонент? Які бувають компоненти?
3. Опишіть функціональні можливості етапу проектування у візуальному середовищі.

Література

Хортон, А. Visual C++ 2005: базовый курс. [Текст] / А.Хортон. – М. : ООО

"И.Д. Вильямс", 2007. - 1152 с.

Лекція

з навчальної дисципліни "Об'єктно-орієнтоване програмування"

Тема Віртуальні функції. Механізм пізнього зв'язування

Мета заняття Ознайомитись з поняттям віртуальних функцій. Розглянути механізм пізнього зв'язування

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН
Лістинги з фрагментами програмного коду

Час – 2 години

Література

Глушаков, С.В. Язык программирования C++. [Текст] / С.В. Глушаков, А.В.Коваль, С.В.Смирнов. - Х.:Фолио, 2001. - 500 с.

Навчальні матеріали лекції

ВСТУП

У C++ реалізований механізм пізнього зв'язування, коли дозвіл посилань на метод відбувається не під час побудови об'єктного коду, а на етапі виконання програми в залежності від конкретного типу об'єкта методу. Цей механізм реалізований за допомогою віртуальних методів.

Віртуальні функції. Механізм пізнього зв'язування

Припустимо , що ми пишемо програму для зображення фігур на екрані. Загальні атрибути фігури представлені класом `shape` , а спеціальні атрибути - спеціальними класами `circle`, `triangle`... :

...

```
class shape {
    point center;
    color col;
    //...
public:
    void move(point to) { center = to; draw(); }
```

```

        point where() { return center; }
        virtual void draw() =0;
        virtual void rotate(int);
        //...
    };

    class circle: public shape {
    public:
        virtual void draw() { ... } //малювання фігури "коло"
        virtual void rotate(int) { ... } //поворот фігури "коло"
        //...
    };
...

```

Функції, які можна визначити, не знаючи точно певної фігури (наприклад, `move` і `where` , тобто, " пересунути центр" і "де центр"), можна описати як звичайні.

Решту функцій не можна визначити, не знаючи конкретного типу фігури, тому вони описуються як `virtual`, тобто такі, які повинні визначатися в похідному класі, наприклад: `void draw()`, `void rotate(int)`.

Для звичайних методів зв'язування виклику з конкретним методом відбувається на етапі побудови об'єктного коду. Для віртуальних методів — під час виконання програми. Наприклад:

```

int main() {
    shape *myfigure;
    int answer(0);
    cin>>i;
    if (i=1) {
        myfigure=new circle();
    } else {
        myfigure=new triangle();
    }
    myfigure->draw(); //виклик віртуального метода
    ...myfigure->where(); //виклик невіртуального метода

}

```

Подібний приклад з іншої предметної області:

```
class demon: public monstr {
    int radius;
public:
    void draw();
    void rotate(int i) {}
    //...
};
```

Робота з об'єктами найчастіше проводиться через покажчики. Вказівником на базовий клас можна присвоїти значення адреси об'єкта будь-якого похідного класу, наприклад:

```
int main() {
    / / Описується покажчик на базовий клас:
    monstr * p;
    / / Покажчик посилається на об'єкт похідного класу:
    if (...) {
        p = new daemon;
    } else {
        p = new ork;
    }
}
```

Виклик методів об'єкту відбувається відповідно до типу покажчика, а не фактичним типом об'єкта, на який він посилається, тому при виконанні оператора, наприклад,

```
p-> draw ();
```

якщо метод не віртуальний, то буде викликаний метод класу monstr, а не класу daemon, оскільки посилання на методи вирішуються під час компонування програми, . Цей процес називається раннім зв'язуванням. Щоб викликати метод класу daemon, можна використовувати явне перетворення типу покажчика:

```
(Daemon * p) -> draw ();
```

Це не завжди можливо, оскільки в різний час покажчик може посилатися на об'єкти різних класів ієрархії, і під час компіляції програми конкретний клас може бути невідомий. Як приклад можна привести функцію, параметром якої є вказівник на об'єкт базового класу. На його місце під час виконання програми може бути переданий покажчик на будь-який похідний клас. Інший приклад — зв'язний список покажчиків на різні об'єкти ієрархії, з яким потрібно працювати однотипно.

Для визначення віртуального методу використовується специфікатор `virtual`, який пишеться при оголошенні (в специфікації), наприклад:

```
virtual void draw (int x, int y, int scale, int position);
```

Розглянемо правила опису та використання віртуальних методів.

1. Якщо в базовому класі метод визначений як віртуальний, метод, визначений в похідному класі з тим же ім'ям і набором параметрів, автоматично стає віртуальним, а з відмінним набором параметрів - звичайним.

2. Віртуальні методи успадковуються, тобто перевизначати їх у похідному класі потрібно тільки при необхідності задати відрізняються дії. Права доступу при перевизначенні змінити не можна.

3. Якщо віртуальний метод перевизначений в похідному класі, об'єкти цього класу можуть отримати доступ до методу базового класу за допомогою операції доступу до області видимості.

4. Віртуальний метод не може оголошуватися з модифікатором `static`, але може бути оголошений як дружній.

5. Якщо в класі вводиться опис віртуального методу, він повинен бути визначений хоча б як чисто віртуальний (той, що не має реалізації в цьому класі).

Чистий віртуальний метод містить ознаку `= 0` замість тіла, наприклад:

```
virtual void f (int) = 0;
```

Чистий віртуальний метод повинен перевизначатися в похідному класі (можливо, знову як чисто віртуальний).

Якщо визначити метод draw в класі monstr як віртуальний, рішення про те, метод якого класу викликати, буде прийматися в залежності від типу об'єкта, на який посилається покажчик:

```
monstr * r, * p;  
r = new monstr; / / Створюється об'єкт класу monstr  
p = new daemon; / / Створюється об'єкт класу daemon  
r-> draw (1. 1, 1, 1); / / Викликається метод monstr:: draw  
p-> draw (1, 1, 1, 1); / / Викликається метод daemon:: draw  
p-> monstr:: draw (1, 1. 1, 1); / / Обхід механізму віртуальних  
методів
```

Якщо об'єкт класу daemon буде викликати метод draw не безпосередньо, а косвенно (тобто з іншого методу, визначеного в класі monstr), буде викликаний метод draw класу daemon.

Отже, віртуальним називається метод, посилення на який дозволяється на етапі виконання програми (переклад англійського слова virtual - в даному значенні всього лише «фактичний», тобто посилення дозволяється за фактом виклику).

Механізм пізнього зв'язування реалізує описане вище.

Висновки

Віртуальні функції пов'язані з успадкуванням класів.

Існують механізми зв'язування на стадії компіляції та на стадії виконання програми.

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на лекції

1. Що таке віртуальні функції?
- 2 Для чого призначені віртуальні функції?
- 3 Що означає механізм пізнього зв'язування?

Лекція

з навчальної дисципліни "Об'єктно-орієнтоване програмування"

Тема Поліморфізм. Віртуальні деструктори

Мета заняття Навчитись реалізовувати поліморфізм і використовувати віртуальні деструктори

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН
Лістинги з фрагментами програмного коду

Час – 2 години

Література

Глушаков, С.В. Язык программирования C++. [Текст] / С.В. Глушаков, А.В.Коваль, С.В.Смирнов. - Х.:Фолио, 2001. - 500 с.

Поліморфізм — одне з основних понять об'єктно-орієнтованого програмування. В загальному випадку під поліморфізмом розуміють можливість при однаковій специфікації мати різну реалізацію.

Поліморфізм поділяють на поліморфізм часу компіляції та поліморфізм часу виконання.

Прикладами поліморфізму часу компіляції можуть бути [перевантаження функцій](#), операторних функцій, так званий параметричний поліморфізм (використання шаблонів класів). Він так називається (поліморфізм часу компіляції або статичний поліморфізм або раннє зв'язування) через те, що іще до виконання програми однозначно відомо, в якому рядку програмного коду яка функція буде викликана, яка спеціалізація створена.

Виділяють ще поліморфізм часу виконання, при якому тільки під час виконання програми стане відомо, метод якого класу буде викликаний. Наприклад, якщо використовується ієрархія класів і методи з однаковою сигнатурою в ієрархії (перевизначені методи). Виконання програми може залежати від дій користувача і в залежності від цього буде викликаний метод того чи іншого класу.

Для поліморфізму часу виконання характерне використання віртуальних методів в ієрархії класів та створення динамічних об'єктів класів (з використанням вказівника).

Поліморфні об'єкти

Для кожного класу, що містить хоча б один віртуальний метод, компілятор створює таблицю віртуальних методів (vtbl), в якій для кожного віртуального методу записаний його адресу в пам'яті.

Адреси методів містяться в таблиці в порядку їх опису в класах. Адреса будь-якого віртуального методу має в vtbl одне і те ж зміщення для кожного класу в межах ієрархії.

Кожен об'єкт містить приховане додаткове поле посилання на vtbl, яке називається vptr. Воно заповнюється конструктором при створенні об'єкта (для цього компілятор додає в початок тіла конструктора відповідні інструкції).

На етапі компіляції посилання на віртуальні методи замінюються на звернення до vtbl через vptr об'єкта, а на етапі виконання в момент звернення до методу його адреса вибирається з таблиці. Таким чином, виклик віртуального методу, на відміну від звичайних методів і функцій, виконується через додатковий етап отримання адреси методу з таблиці.

Рекомендується робити віртуальними деструктори для того, щоб гарантувати правильне звільнення пам'яті з-під динамічного об'єкта, оскільки в цьому випадку в будь-який момент часу буде обраний деструктор, який відповідає фактичному типу об'єкта.

Деструктор передає операції delete розмір об'єкта, що має тип size_t. Якщо видаляється об'єкт класу, який є похідним, і в ньому не визначено віртуальний деструктор, переданий розмір об'єкта може виявитися неправильним.

Чіткого правила, за яким метод слід робити віртуальним, не існує. Можна тільки дати рекомендацію оголошувати віртуальними методи, для котрих є ймовірність, що вони будуть перевизначені в похідних класах.

Методи, які у всій ієрархії залишаються незмінними або ті, якими похідні класи користуватися не будуть, робити віртуальними немає сенсу. З іншого боку, при проектуванні ієрархії не завжди можна передбачити, яким чином будуть розширюватися базові класи (особливо при проектуванні бібліотек класів), а [оголошення](#) методу віртуальним забезпечує гнучкість і можливість розширення.

Для пояснення останньої тези уявімо собі, що виклик методу `draw` здійснюється з методу переміщення об'єкта. Якщо текст методу переміщення не залежить від типу переміщуваного об'єкту (оскільки принцип переміщення всіх об'єктів однаковий, а для відтворення викликається конкретний метод), перевизначати цей метод в похідних класах немає необхідності, і він може бути описаний як невіртуальний. Якщо метод `draw` віртуальний, метод переміщення зможе без перекомпіляції працювати з об'єктами будь-яких похідних класів - навіть тих, про які при його написанні нічого відомо не було.

Віртуальний механізм працює тільки при використанні покажчиків або посилань на об'єкти. Об'єкт, визначений через покажчик або посилання й утримуючий віртуальні методи, називається поліморфним.

В даному випадку поліморфізм полягає в тому, що за допомогою одного і того ж звернення до методу виконуються різні дії в залежності від типу, на який посилається вказівник в кожен момент часу.

Відповідно до типів методів процес установки взаємозв'язку об'єктів і методів може бути:

- раннім - під час компіляції - для статичних методів;
- пізнім - під час виконання програми (виклику методу) - для віртуальних методів.

Адреси віртуальних і динамічних методів при їх виклику (дозвіл посилань на методи) визначаються не під час компіляції, а під час виконання програми по спеціальній таблиці (пізніє скріплення).

Висновки

За умовчанням (без `virtual`) методи є статичними; їх адреса визначається під час компіляції проекту (раннє скріплення). Вони викликаються найшвидше. Транслятор дозволяє (формує) виклики статичних методів на етапі компіляції.

Питання та завдання до контролю знань студентів

1 Для актуалізації опорних знань та перевірки попереднього матеріалу

1. Яка послідовність виклику конструкторів в ієрархії класів?
2. Які особливості успадкування конструкторів, деструкторів та операцій присвоювання?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

1. Що таке поліморфізм?
 2. Для чого використовуються та як працюють віртуальні методи?
- 3 Що таке поліморфні об'єкти?

Література

Лафоре, Р. Объектно-ориентированное программирование в С++ [Текст] / Р. Лафоре. –
М. - СПб. : ПИТЕР, 2004.- 923 с.

Лекція

з навчальної дисципліни "Об'єктно-орієнтоване програмування"

Тема Абстрактні класи. Віртуальні класи

Мета заняття Ознайомитись з поняттями абстрактних класів, віртуальних класів

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН
Лістинги з фрагментами програмного коду

Час – 2 години

Література

1 Довбуш, Г. Ф. Visual C++ на примерах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

2 Глушаков, С.В. Язык программирования C++. [Текст] / С.В. Глушаков, А.В.Коваль, С.В.Смирнов. - Х.:Фолио, 2001. - 500 с.

Навчальні матеріали лекції

ВСТУП

Абстрактні класи, віртуальні класи використовуються при успадкуванні класів як базові.

Абстрактні класи

Клас, що містить хоч би один чисто віртуальний метод, називається абстрактним. Абстрактні класи призначені для представлення загальних понять, які передбачається конкретизувати в похідних класах.

Абстрактний клас може використовуватися тільки як базовий для інших класів – об'єкту абстрактного класу створювати не можна, оскільки прямий або непрямий виклик чисто віртуального методу призводить до помилки при виконанні.

При визначенні абстрактного класу необхідно мати на увазі наступне:

1. абстрактний клас не можна використати при явному приведенні типів, для опису типу параметра і типу значення, яке повертається функцією;

2. допускається оголошувати покажчики і посилання на абстрактний клас, якщо при ініціалізації не вимагається створювати тимчасовий об'єкт;

3. якщо клас, похідний від абстрактного, не визначає все чисто віртуальні функції, він також є абстрактним.

Таким чином, можна створити функцію, параметром якої є вказівник на абстрактний клас. На місце цього параметра при виконанні програми може передаватися покажчик на об'єкт будь-якого похідного класу. Це дозволяє створювати поліморфні функції, працюючі з об'єктом будь-якого типу в межах однієї ієрархії.

Віртуальні класи

В якості альтернативи операції дозволу зони дії при множинному успадкуванні, можна зажадати, щоб похідний клас містив тільки одну копію базового. Цього можна досягти, описавши базовий клас при спадкоємстві від нього як віртуальний за допомогою ключового слова `virtual`. Ось модифікація прикладу, яка робить клас A віртуальним базовим класом :

```
class A { public:
    int AData;
    void AFunc ();
    // . . . };

class B: public virtual A // A - віртуальний базовий клас.
{
}:

class C: public virtual A // A - віртуальний базовий клас.
{
    // ...
};

class D: public B, public C // Містить тільки одну копію A.
{
    // ...
};

int main(void) {
    D d;
```

```

d.AData = 0; // Тепер неоднозначності не виникає.
d.AFunc();
//
return 0;
}

```

Віртуальні базові класи - хитріша річ, ніж може здатися з першого погляду. Якщо, припустимо, конструктори "проміжних" класів В і С явно викликають у своїх списках ініціалізації якийсь конструктор з параметрами класу А, те знову виникає неоднозначність - який набір параметрів компілятор повинен використати при конструюванні тієї єдиної копії А, яка міститься в С?

Тому, якщо при конструюванні похідного класу повинен ініціалізуватися віртуальний базовий клас(нехай навіть він ініціалізувався непрямым чином через конструктори проміжних класів), в списку ініціалізації вимагається явно вказати ініціалізатор віртуального базового класу, приблизно так:

```

D: :D(...) : B( . . . ) , C( . . . ) , A( . . . ) {
// ... }

```

Все виглядає так, як якби віртуальний базовий клас був безпосереднім предком похідного класу D "в обхід" проміжних класів ієрархії. Тому, якщо потрібно виклик конструктора віртуального базового класу, останній обов'язково має бути присутнім в списку ініціалізації похідного класу, навіть якщо реально не виникає ніякої неоднозначності. Така ситуація існує, наприклад, у бібліотеці *Object Windows Library (OWL)*, яка має ієрархію із складним успадкуванням і віртуальними базовими класами. Конструктор головного вікна додатка там повинен визначатися так:

```

TMyWindow::TMyWindow(TWindow *parent, const char *title):
TFrameWindow(parent, title), TWindow(parent, title) {
// ... }

```

TWindow — віртуальний базовий клас, тому він повинен ініціалізуватися окремо, хоча це виглядає абсолютно зайвим, оскільки TFrameWindow, проміжний клас, сам викликає конструктор TWindow з тими ж параметрами.

Висновки

Абстрактний клас може використовуватися тільки як базовий для інших класів – об'єкту абстрактного класу створювати не можна, оскільки прямий або непрямий виклик чисто віртуального методу призводить до помилки при виконанні.

Щоб похідний клас містив тільки одну копію базового, цього можна досягти, описавши базовий клас при спадкоємстві від нього як віртуальний

Питання та завдання до контролю знань студентів

1 Для актуалізації опорних знань та перевірки попереднього матеріалу

1. Що таке віртуальні функції?
- 2 Для чого призначені віртуальні функції?
- 3 Що означає механізм пізнього зв'язування?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

1. Що таке абстрактний клас, для чого він використовується?
- 2 Що таке віртуальний клас, для чого він використовується?

Лекція

з навчальної дисципліни "Об'єктно-орієнтоване програмування"

Тема Поліморфні об'єкти

Мета заняття Ознайомитись з поняттям поліморфних об'єктів

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН
Лістинги з фрагментами програмного коду

Час – 2 години

Література

1 Довбуш, Г. Ф. Visual C++ на прикладах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

2 Глушаков, С.В. Язык программирования C++. [Текст] / С.В. Глушаков, А.В.Коваль, С.В.Смирнов. - Х.:Фолио, 2001. - 500 с.

Поліморфізм — одне з основних понять об'єктно-орієнтованого програмування. В загальному випадку під поліморфізмом розуміють можливість при однаковій специфікації мати різну реалізацію.

Поліморфізм поділяють на поліморфізм часу компіляції та поліморфізм часу виконання.

Прикладами поліморфізму часу компіляції можуть бути [перевантаження функцій](#), операторних функцій, так званий параметричний поліморфізм (використання шаблонів класів). Він так називається (поліморфізм часу компіляції або статичний поліморфізм або раннє зв'язування) через те, що іще до виконання програми однозначно відомо, в якому рядку програмного коду яка функція буде викликана, яка спеціалізація створена.

Виділяють ще поліморфізм часу виконання, при якому тільки під час виконання програми стане відомо, метод якого класу буде викликаний. Наприклад, якщо використовується ієрархія класів і методи з однаковою сигнатурою в ієрархії (перевизначені методи). Виконання програми може залежати від дій користувача і в залежності від цього буде викликаний метод того чи іншого класу.

Для поліморфізму часу виконання характерне використання віртуальних методів в ієрархії класів та створення динамічних об'єктів класів (з використанням вказівника).

Поліморфні об'єкти

Для кожного класу, що містить хоча б один віртуальний метод, компілятор створює таблицю віртуальних методів (vtbl), в якій для кожного віртуального методу записаний його адресу в пам'яті.

Адреси методів містяться в таблиці в порядку їх опису в класах. Адреса будь-якого віртуального методу має в vtbl одне і те ж зміщення для кожного класу в межах ієрархії.

Кожен об'єкт містить приховане додаткове поле посилання на vtbl, яке називається vptr. Воно заповнюється конструктором при створенні об'єкта (для цього компілятор додає в початок тіла конструктора відповідні інструкції).

На етапі компіляції посилання на віртуальні методи замінюються на звернення до vtbl через vptr об'єкта, а на етапі виконання в момент звернення до методу його адреса вибирається з таблиці. Таким чином, виклик віртуального методу, на відміну від звичайних методів і функцій, виконується через додатковий етап отримання адреси методу з таблиці.

Рекомендується робити віртуальними деструктори для того, щоб гарантувати правильне звільнення пам'яті з-під динамічного об'єкта, оскільки в цьому випадку в будь-який момент часу буде обраний деструктор, який відповідає фактичному типу об'єкта.

Деструктор передає операції delete розмір об'єкта, що має тип `size_t`. Якщо видаляється об'єкт класу, який є похідним, і в ньому не визначено віртуальний деструктор, переданий розмір об'єкта може виявитися неправильним.

Чіткого правила, за яким метод слід робити віртуальним, не існує. Можна тільки дати рекомендацію оголошувати віртуальними методи, для котрих є ймовірність, що вони будуть перевизначені в похідних класах.

Методи, які у всій ієрархії залишаються незмінними або ті, якими похідні класи користуватися не будуть, робити віртуальними немає сенсу. З іншого боку, при проектуванні ієрархії не завжди можна передбачити, яким чином будуть розширюватися базові класи (особливо при проектуванні бібліотек класів), а [оголошення](#) методу віртуальним забезпечує гнучкість і можливість розширення.

Для пояснення останньої тези уявімо собі, що виклик методу `draw` здійснюється з методу переміщення об'єкта. Якщо текст методу переміщення не залежить від типу переміщуваного об'єкту (оскільки принцип переміщення всіх об'єктів однаковий, а для відтворення викликається конкретний метод), перевизначати цей метод в похідних класах немає необхідності, і він може бути описаний як невіртуальний. Якщо метод `draw` віртуальний, метод переміщення зможе без перекомпіляції працювати з об'єктами будь-яких похідних класів - навіть тих, про які при його написанні нічого відомо не було.

Віртуальний механізм працює тільки при використанні покажчиків або посилань на об'єкти. Об'єкт, визначений через покажчик або посилання й утримуючий віртуальні методи, називається поліморфним.

В даному випадку поліморфізм полягає в тому, що за допомогою одного і того ж звернення до методу виконуються різні дії в залежності від типу, на який посилається вказівник в кожен момент часу.

Відповідно до типів методів процес установки взаємозв'язку об'єктів і методів може бути:

— раннім - під час компіляції - для статичних методів;

— пізнім - під час виконання програми (виклику методу) - для віртуальних методів.

Адреси віртуальних і динамічних методів при їх виклику (дозвіл посилань на методи) визначаються не під час компіляції, а під час виконання програми по спеціальній таблиці (пізніє скріплення).

Висновки

За умовчанням (без `virtual`) методи є статичними; їх адреса визначається під час компіляції проекту (раннє скріплення). Вони викликаються найшвидше. Транслятор дозволяє (формує) виклики статичних методів на етапі компіляції.

Питання та завдання до контролю знань студентів

1 Для актуалізації опорних знань та перевірки попереднього матеріалу

1. Яка послідовність виклику конструкторів в ієрархії класів?
2. Які особливості успадкування конструкторів, деструкторів та операцій присвоювання?

2 Для узагальнення та перевірки засвоєного матеріалу на лекції

1. Що таке поліморфізм?

2. Для чого використовуються та як працюють віртуальні методи?

3 Що таке поліморфні об'єкти?

Література

Лафоре, Р. Объектно-ориентированное программирование в С++ [Текст] / Р. Лафоре. –

М. - СПб. : ПИТЕР, 2004.- 923 с.

Тема. Розробка об'єктно-орієнтованого програмного забезпечення

Мета: навчитися розробляти об'єктно-орієнтоване програмне забезпечення

При розробці об'єктно-орієнтованого програмного забезпечення потрібно описати визначення та реалізацію нового класу або класів, при необхідності, пов'язаних простим успадкуванням або відношенням агрегації. Є завдання, де необхідно спроектувати базу даних та виконати її підключення в проект, можна використати спеціальні компоненти для підключення джерел даних. Слід пам'ятати, що візуальні та невізуальні компоненти бібліотек теж описуються пов'язаними відповідними відношеннями класів.

Для демонстрації роботи потрібно запрограмувати модуль головної екранної форми.

У проекті повинні бути продемонстровані різні способи створення об'єктів, при наявності масивів об'єктів.

Програма повинна демонструвати використання всіх своїх функцій і методів класу.

Для установки значень окремих полів об'єктів можна використовувати методи, у тому числі конструктори.

Заповнення полів об'єкта може виконуватися з екрану.

Введені значення потрібно перевіряти на допустимість.

Для виведення закритих даних об'єкту потрібно реалізувати методи, відкриті дані можна виводити безпосередньо на екран.

Теоретичні рекомендації щодо розробки програми

Використання середовища Embarcadero C++ Builder

Створюємо новий проект File -> New -> Windows VCL Application - C++ Builder, задаємо ім'я проекту та шлях його збереження (рекомендовано створити власну папку).

З використанням Палітри компонентів (Palette) мишею розміщуємо на візуальній формі необхідні компоненти: надписи (TLabel), однорядкові редактори для введення-виведення даних (TEdit), кнопки (TButton) та ін. Щоб розмістити компонент на формі, слід клацнути по ньому на Палітрі компонентів (в рядку з зображенням лупи можна виконати

пошук за назвою компоненту), а потім клацнути в тому місці форми, де треба розмістити компонент.

У форми можна змінити текст верхнього заголовка, якщо клікнути на формі мишею і в Інспекторі об'єктів в полі Caption ввести потрібний текст.

Призначення основних рекомендованих до використання на етапі виконання практики компонентів (приклади використання під час виконання програми наведені нижче):

- TButton (Кнопка) — використовується для реалізації кнопок керування додатком, таких як "Додати", "Розрахувати" та ін. Оброблювана подія OnClick (в Інспекторі об'єктів на вкладинці Events);

- TCheckBox (Прапорець) — незалежний перемикач, аналізую стан якого можна керувати додатком;

- TEdit (Однорядковий редактор) — використовують для текста, який можна редагувати під час виконання. Подія, яку рекомендується оброблювати при виконанні завдань OnChange (подія, яка виникає при зміні тексту в компоненті). Для додавання коду в обробник події для візуального компоненту, слід кліком миші виділити потрібний елемент керування та задати реакцію на подію. Це виконується в Інспекторі об'єктів на вкладинці Events. В результаті буде сформирована функція обробки події (наприклад, OnChange);

- TComboBox (Поле зі списком) — представляє собою комбінацію поля редагування та списку, дозволяє вибирати елементи зі списку, який під час виконання може змінюватися програмно, і набирати текст в поле введення. Подія, яка рекомендована до обробки OnSelect (вибір елементу списку). Колекція елементів списку задається з використанням властивості Items (можна задавати як під час проектування, так і програмно), індекс вибраного елементу списку - ItemIndex (-1 - нічого не вибрано), Text - поле рядка редагування в ComboBox (при необхідності);

- TListBox (Список) — слугує для вибору користувачем елементу списку, який під час виконання може змінюватися програмно;

- TGroupBox (Група) — слугує для об'єднання в групу декількох залежних перемикачів або інших візуальних компонентів. Надпис зверху GroupBox можна змінити з використанням властивості Caption;

- TRadioButton (Перемикач) — слугує для задання залежного перемикача в складі групи;

- TLabel (Надпис) — слугує для відображення тексту, який не редагується та використовується для надпису для іншого елемента керування. Текст задається в полі Caption;

TProgressBar - слугує для відображення послідовності виконання довготривалого процесу.

В завданнях, де потрібно використовувати підключення до бази даних, рекомендовано створити базу даних MySQL з таблицями згідно завдання. Для підключення бази даних в візуальний додаток рекомендовано встановити програму MySQLConnector. В програмі "Администратор источников данных ODBC" додати джерело даних користувача MySQL, задати ім'я, вказати сервер, ім'я БД, при необхідності логін і пароль.

У візуальному додатку для підключення доцільно розмістити на формі компонент TADOConnection, в Інспекторі об'єктів встановити властивість ConnectionString, вказати джерело даних користувача з ім'ям, яке було створене вище, і після цього встановити поле Connection true.

Для роботи з конкретною таблицею БД або з запитом доцільно використовувати компоненти TADOTable та TADOQuery відповідно, для чого задати властивість Connection=ADOConnection1. В TADOTable додатково вказується поле TableName, а в TADOQuery - SQL. Властивість Active=true відкриває набори.

Для того, щоб вивести інформацію з БД на візуальну форму, рекомендовано виконати наступне. Розмістіть на формі TDataSource, вкажіть властивість DataSet=ADOTable1 або ADOQuery1. На формі розмістіть TDBGrid та TDBNavigator, задайте їм властивості DataSource=DataSource1.

В DBGrid з'являться дані з таблиці БД або відповідного запиту. За допомогою DBNavigator можна виконувати навігацію та модифікацію.

ADOQuery1->SQL->Text="UPDATE TABLE1 SET FIELD1=VALUE1 WHERE ID=VALUE2";
//Приклад збереження без використання навігатора.

Приклади з фрагментами програмного коду наведені в лістингу.

Лістинг – Фрагменти програмного коду

```
//-----  
void __fastcall TForm1::Button1Click(TObject *Sender) //Обробник події OnClick
```

```

{
    String s=Edit1->Text; //Отримати введені дані з Edit1
або
    Edit1->Text="Новий текст";

    ListBox1->Items->Add("Новий рядок в списку");//Додати рядок
    ListBox1->Items->Clear();");//Очистити

    CheckBox1->Checked=true; //Встановити позначку "v" в незалежний перемикач

    ComboBox1->Items->Add("Новий рядок в випадяючому списку");

    ProgressBar1->Min=1;
    ProgressBar1->Max=10;
    ProgressBar1->StepBy(1); //Пересунути повзунок
}

//-----
void __fastcall TForm1::Edit1Change(TObject *Sender) //Обробка події OnChange
для Edit1
{

}

//-----

```

При проектуванні додатку слід пам'ятати про розподілення функціональних обов'язків між різними класами та формами, для чого потрібно провести делальний аналіз предметної області. Класи можуть "знати" про інтерфейси один одного, але всі інші члени класів повинні бути прихованими згідно правил інкапсуляції та підвищення рівня абстракції програми.

Література

1 Довбуш, Г. Ф. Visual C++ на примерах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

2 Хортон, А. Visual C++ 2005: базовый курс. [Текст] / А.Хортон. – М. : ООО "И.Д. Вильямс", 2007. - 1152 с.

Додатки на основі об'єктно-орієнтованого підходу

Програма повинна демонструвати використання всіх функцій і методів.

Для установки значень окремих полів об'єктів потрібно використовувати конструктори з параметрами та методи встановлення значень даних (дані класу повинні бути закритими).

Виведення значень об'єктів повинно здійснюватися методами класу.

Робота з компонентами вводу та відображення текстових даних.

Використання компонентів-списків List Box, Combo Box, Edit Box, Static Text.

Вимоги до інтерфейсу.

Компонент або його властивість	Призначення
List Box	Виведення на екран списку
Static Text	Виведення пояснювальних написів
Edit Box	Введення початкових даних та виведення тексту
Button	Обробки події клацання

Програма повинна:

- забезпечити введення даних, перелічених у варіанті завдання, а також вибір значень і найменувань зі списку, сформованого на етапі проектування програми;
- забезпечити контроль даних, що вводяться;
- видавати користувачеві повідомлення про некоректні дані;

- виконати повторення введення необхідних значень стільки разів, скільки побажає користувач;

- виконати розрахунок необхідних сумарних характеристик, порівняння результатів і тому подібне та вивести результати на екран.

Створення групи вибору, використання властивості Enabled для керування роботою програми, робота з текстовими файлами.

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на лекції

Як створюються додатки на основі об'єктно-орієнтованого підходу?

Література

1 Довбуш, Г. Ф. Visual C++ на примерах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

2 Хортон, А. Visual C++ 2005: базовый курс. [Текст] / А.Хортон. – М. : ООО "И.Д. Вильямс", 2007. - 1152 с.

Навчимось складати та відлагоджувати програми з пов'язаними між собою класами на прикладі реалізації предметної області "Бібліотека".

Клас "Книга" містить елементи, що описують книгу. Клас "Каталог" містить інформацію щодо переліку книг і функціонал пошуку книги, додавання, сортування в каталозі тощо. Необхідно розподілити обов'язки між трьома класами: CBook, CCatalog - звичайні класи, а функціонал, запропонований для класу CMenu, реалізувати на формі у вигляді кнопок тощо (за натисканням звертатися до каталогу або книги), на цій же формі спроектувати інтерфейс для введення та виведення.

Створимо класи згідно завдання, для цього слід:

- проаналізувати постановку задачі;
- сформулювати функції, які повинна виконувати програма;
- виділити вхідну і вихідну інформацію, виявити типи даних, які доцільно їм відповідають;
- спроектувати та розробити програму.

Створити власні класи: візуальна форма (описує користувацький інтерфейс, реалізовує функціонал наведеного нижче для зразка класу CMenu), клас CCatalogue (описує масив із 100 вказівників на об'єкти типу CBook) та клас CBook.

В лістингах наведені фрагменти програмного коду для класів CMenu (зразок для функціоналу візуальної форми), CCatalogue, CBook та програмного коду головної функції як приклад, як можна було б реалізувати в консольному додатку.

Слід звернути увагу, що у цьому прикладі в клас CBook доданий новий метод check(), який перевіряє збіг значення параметра методу і значення члена-даного m_title. Метод використовується об'єктом класу CCatalogue при пошуку книги в каталозі за назвою. Об'єкт класу CCatalogue не в змозі самотійно виконати перевірку на збіг значення змінної key з назвою книги m_title, тому що назва належить книзі (*m_pBook [i]), а не каталогу (m_p), якому відома лише адреса книги (m_pBook [i]). Тільки сама книга знає свою назву і може відповісти, чи збігається вона з key. Така організація взаємодії програмних об'єктів menu, *m_p і *m_pBook[i] робить програму об'єктно-орієнтованою, коли обов'язки розподіляються між об'єктами і кожен з них несе персональну відповідальність за їх виконання.

Реалізацію наведеного коду доцільно запропонувати в візуальному додатку C++ Builder або іншому (мовою C++) з дотриманням функціоналу наведених класів в цілому та незначною зміною інтерфейсів класів за необхідністю. Введення та виведення виконується з використанням інтерфейсу.

```
// Зразок програмного коду головної функції, як це було б в консольному
додатку
    CMenu menu ;menu.showMenuAndSelect ( ) ;

// Menu.h Специфікація класу меню CMenu, як це було б в консольному додатку
#include "Catalogue.h"
class CMenu {
public:
    CCatalogue* m_p ;           // вказівник на каталог книг
    CMenu ( ) ; ~CMenu ( ) ;    // конструктор і деструктор
    void showMenuAndSelect ( ) ; // активізація меню
private:
    void find ( ) const ;       // пошук книги за назвою
    void view ( ) const ;       // перегляд стану каталогу
    void ins ( ) ;              // додавання книги
    void del ( ) ;              // видалення книги за назвою
    void sort ( ) ;             // сортування в каталозі за автором
    void align ( int ) const ;  // вирівнювання при виведенні
} ;

// Menu.cpp Реалізація класу CMenu, як це було б в консольному додатку

#include <iostream>
#include <iomanip>
using namespace std ;
#include "Menu.h"
const static int N ( 512 ) ;
CMenu :: CMenu ( ) : m_p ( new CCatalogue ) {} // конструктор
CMenu :: ~CMenu ( ) { delete m_p ; }          // деструктор
```

```

void CMenu :: showMenuAndSelect ( ) { // активізація меню
    char choice ;
    do {
        cout << "\n1. Find and look up" ;
        cout << "\n2. Insertion" ;
        cout << "\n3. Deletion" ;
        cout << "\n4. Sorting" ;
        cout << "\n5. View" ;
        cout << "\n0. Exit\n" ;
        cout << "Please, your choice -> " ;
        cin >> choice ; cin.ignore ( ) ;
        if ( m_p -> getNumberOfBook ( ) )
            switch ( choice )
            {
                case '1': find ( ) ; break ;
                case '2': ins ( ) ; break ;
                case '3': del ( ) ; break ;
                case '4': sort ( ) ; break ;
                case '5': view ( ) ; break ;
            }
        else
            if ( ! m_p -> getNumberOfBook ( ) && choice == '2' )
                ins ( ) ;
    } while ( choice != '0' ) ;
}

void CMenu :: find ( ) const{ // пошук по ключу
    string title ;
    cout << "\nEnter Title -> " ; getline ( cin, title ) ;
    int iKey = m_p -> find( title ) ;
    if ( iKey != -1 )
    {
        cout << "Author: " ;
        cout << m_p -> m_pBook [ iKey ] -> getAuthor( ) << endl ;
        cout << "Title" ;
        cout << m_p -> m_pBook [ iKey ] -> getTitle( ) << endl ;
        cout << "Year: " ;
        cout << m_p -> m_pBook [ iKey ] -> getYear( ) << endl ;
    }
    else cout << title << " - NOT FOUND\n" ;
}

void CMenu :: ins ( ) // додавання
{
    string author, title ; int year ;
    cout << "\nEnter Title to add -> " ; getline ( cin, title ) ;
    if ( m_p -> find ( title ) < 0 )
    {
        cout << "\nEnter Author to add -> " ;
        getline ( cin, author ) ;
        cout << "\nEnter Year to add -> " ; cin>> year ;
        if ( m_p -> ins ( author, title, year ) )
            cout << "\nINSERTION IS COMPLETED!\n";
    }
    else cout << title << " - INSERTION IS IMPOSSIBLE\n" ;
}

void CMenu :: del ( ) // видалення

```

```

{
    string title ;
    cout << "\nEnter Title -> " ; getline ( cin, title ) ;
    if ( m_p -> del ( title ) )      cout << "\nDELETION IS COMPLETED!\n";
    else cout << title << " - DELETION IS IMPOSSIBLE\n" ;
}
void CMenu :: sort ( )                // сортування
{    m_p -> sort ( ) ;      cout << "\nSORTING IS COMPLETED!\n" ;      }
void CMenu :: align ( int n ) const    // вирівнювання
{    cout.width ( n ) ; cout.setf ( ios_base::left ) ;      }

// Catalogue.h  Специфікація класу CCatalogue
#pragma once
static const int MAX_SIZE = 100 ;
#include "Book.h"
class CCatalogue
{
    int m_q ;                // фактична кількість книг
public:
    CBook* m_pBook [ MAX_SIZE ] ; // масив вказівників на книги
public:
    CCatalogue ( ) ; ~CCatalogue ( ) ; // конструктор і деструктор
    bool ins ( const string&, const string&, int ) ; // додавання
    bool del ( const string& ) ; // видалення
    int find ( const string& ) const ; // пошук
    void sort ( ) ; // сортування
    int getNumberOfBook ( ) const ; // повернення кількості книг
} ;

// CCatalogue.cpp Реалізація класу CCatalogue
#include "Catalogue.h"
CCatalogue :: CCatalogue ( ) : m_q ( 0 ) { } // конструктор
CCatalogue :: ~CCatalogue ( ) // деструктор
{    for ( int i = 0; i < m_q; i++ ) delete m_pBook [ i ] ; }
// повернення фактичної кількості книг
int CCatalogue :: getNumberOfBook ( ) const { return m_q ; }
/*    додавання нової книги
1 перевірити наявність книги в масиві;
2 якщо книга вже є в масиві, повернути false
   якщо книги немає ->
       - створити новий об'єкт і занести в масив;
       - збільшити кількість книг в масиві і повернути true */
bool CCatalogue ::
ins ( const string& author, const string& title, int year )
{
    if ( find ( title ) != -1 ) return false ;
    m_pBook [ m_q++ ] = new CBook ( author, title, year ) ;
    return true ;
}
/*    видалення книги з масиву
1 перевірити наявність книги в масиві
2 якщо книги немає в масиві, повернути false
   якщо книга знайдена ->
       - звільнити знайдений об'єкт;
       - стиснути масив;
       - зменшити кількість книг в масиві і повернути true.*/
bool CCatalogue :: del ( const string& key )

```

```

{
    int i = find ( key ) ;    if ( i < 0 )    return false ;
    delete m_pBook [ i ] ;
    while ( i < m_q - 1 ) m_pBook [ i++ ] = m_pBook [ i + 1 ] ;
    m_q-- ;    return true ;
}
/*    пошук книги за назвою
1 перевірити назву кожної книги;
2 якщо є співпадання, повернути індекс книги
   якщо немає співпадань, повернути -1.    */
int CCatalogue :: find ( const string& key ) const
{
    for ( int i = 0; i < m_q; i++ )
        if ( m_pBook [ i ] -> check ( key ) )    return i ;
    return -1 ;
}

// Book.h - специфікація класу CBook
#pragma once
#include <string>
using namespace std ;
class CBook
{
public:
    CBook ( ) ; CBook ( const string&, const string&, int ) ;
    ~CBook ( ) ;
private:
    string m_author ;    // автор
    string m_title ;    // назва
    int m_year ;    // рік видання
public:
    // методи встановлення значень
    void setAuthor ( const string & ) ;
    void setTitle ( const string & ) ;
    void setYear ( const int ) ;
    // методи повернення значень
    string getAuthor ( ) const { return m_author ; }
    string getTitle ( ) const { return m_title ; }
    int getYear ( ) const { return m_year ; }
    // метод перевірки назви книги
    bool check ( const string& ) const ;
} ;

// Book.cpp - реалізація класу CBook
#include "Book.h"
CBook :: CBook ( ) : m_year ( 0 ), m_title ( "" ), m_author ( "" ) {}
CBook :: CBook ( const string& author, const string& title, int year )
: m_year ( year ), m_title ( title ), m_author ( author ) { }
CBook :: ~CBook ( ) { }
// встановити автора
void CBook :: setAuthor ( const string& author ) { m_author = author ; }
// встановити назву
void CBook :: setTitle ( const string& title ) { m_title = title ; }
// встановити рік видання
void CBook :: setYear ( const int year ) { m_year = year ; }
// перевірка книги за назвою
bool CBook :: check ( const string& key ) const

```

```
{    if (m_title == key ) return true ;    return false ; }
```

Класи в програмі пов'язані між собою, відстежується послідовність створення об'єктів, ініціалізація полів.

Питання та завдання до контролю знань студентів

Для узагальнення та перевірки засвоєного матеріалу на лекції

Як організовано взаємодію об'єктів?

Як розподілений функціонал між класами?

Який клас за що відповідає і як обмінюється інформацією з іншим класом?

Література

Довбуш, Г. Ф. Visual C++ на примерах [Текст] / Г. Ф. Довбуш, А. Д. Хомоненко; под ред. А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2007. -528 с.

ОРІЄНТОВНІ ТЕОРЕТИЧНІ ПИТАННЯ ДО ОКР

1. Що таке конструктор класу? Які види конструкторів Вам відомі? Навести приклади опису та використання конструкторів.
2. Для чого призначені і як описуються і шаблони класів? Навести приклади оголошення шаблону класу та створення відповідного об'єкту.
3. Що таке дружні функції і для чого вони використовуються? Навести приклад використання дружньої функції.
4. Що таке об'єкт класу? Як встановлюються значення полів об'єкту? Привести приклади встановлення значень полів об'єктів одного й того ж класу, оголошених за допомогою імені та покажчика.
5. Що означають елементи опису класу private, protected та public? Навести приклад використання цих елементів.
6. Опишіть послідовність роботи конструкторів та деструкторів в ієрархії класів. Наведіть приклади.
7. Що означає поняття інкапсуляції? Для чого і як потрібно приховувати інформацію? Навести приклади.
8. Що означає поняття поліморфізму? Навести приклади.
9. Як описується конструктор з параметрами і для чого він призначений? Навести приклад використання конструктору з параметрами.
10. Що таке об'єкт класу? Навести приклади встановлення значень для закритих полів об'єкту.
11. Що таке дружні функції і для чого вони використовуються? Навести приклад використання дружньої функції.
12. Що таке операторні функції-друзі класу? Як вони використовуються?
13. Що означає просте поодиноке успадкування? Привести приклад. Як називаються класи, пов'язані успадкуванням? Які члени класу не успадковуються?

- 14.Що означає відкрите успадкування? Привести приклад. Як успадковуються члени базового класу в цьому разі?
- 15.Як оголошується операторні функції-члени класу? Привести основні формати оголошення.
- 16.Що таке перевантаження функцій? Навести приклади.
- 17.Що означає поняття поліморфізму? Навести приклади.
- 18.Для чого потрібні дружні функції та дружні класи? Навести приклади.
- 19.Що таке перевантаження операторів? Навести приклади.
- 20.Що означає закрите успадкування? Привести приклад. Як успадковуються члени базового класу в цьому разі?
- 21.Що є членами класу? Що являють собою операторні функції-члени класу? Навести приклад.
- 22.Що означає поняття абстракції? Навести приклади.
- 23.Що означає поняття поліморфізму? Навести приклади.
- 24.Що означає захищене успадкування? Привести приклад. Як успадковуються члени базового класу в цьому разі?
- 25.Як створюються об'єкти? Привести приклади встановлення значення полів об'єктів за допомогою ініціалізації в конструкторах та за допомогою методів класу.
- 26.Що таке конструктор класу? Які види конструкторів Вам відомі? Навести приклади використання конструкторів.
- 27.Що означає захищене успадкування? Привести приклад. Як успадковуються члени базового класу в цьому разі?
- 28.Що таке дружні функції і для чого вони використовуються? Навести приклад використання дружньої функції.
- 29.Що означає множинне успадкування? Привести приклади. Які конфлікти можуть бути пов'язані з використанням множинного успадкування?
- 30.Як оголошується операторні функції-члени класу? Привести основні формати оголошення.
- 31.Що таке перевантаження операторів? Навести приклади.
- 32.Як пов'язані класи при використанні агрегації або композиції? Навести приклади.
- 33.Для чого призначені статичні дані та статичні методи? Як вони використовуються?
- 34.Шаблони класів. Для чого. Приклад оголошення та об'єкта

Для чого призначені методи класу? Навести приклади опису методу класу.

ПРИКЛАДИ ДРУГОГО та ТРЕТЬОГО ПИТАННЯ

1. Прокоментувати наведений фрагмент програмного коду та дати відповіді на такі запитання:

2.1. Яке ім'я має клас, фрагмент опису якого приведено?

2.2. У якому файлі проекту розміщується приведений фрагмент?

2.3. Перелічити дані та методи класу, які зустрічаються у фрагменті.

2.4. Яке призначення має кожний рядок програмного коду?

```
class CPoint
{
public:
    CPoint();
    virtual ~CPoint();
    int m_x, m_y ;
    CPoint ( int , int );
    friend CPoint& operator ++ ( CPoint& ) ;
    friend CPoint& operator ++ ( CPoint&, int ) ;
};
```

3. Оголосити клас із ім'ям REIS, що містить наступні поля:

– назва кінцевого пункту маршруту;

– час відправлення;

час відправлення – об'єкт класу TIM.

Створити об'єкт класу REIS за допомогою конструктору з параметрами. Змінити дані об'єкту за допомогою методів класу. Вивести на екран значення даних за допомогою методів класу.