А. І. Купін, І. О. Музика

МЕРЕЖНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ
ПРАКТИКУМ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДВНЗ «КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»

А. І. Купін, І. О. Музика

МЕРЕЖНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

ПРАКТИКУМ
НАВЧАЛЬНИЙ ПОСІБНИК

Рекомендовано
для студентів вищих навчальних закладів

Кривий Ріг
2015

УДК 004.77
К 92
ББК 32.973.26-018+33.2

Рекомендовано до друку вченою радою
ДВНЗ «Криворізький національний університет»
(протокол №6 від 27.01.2015 р.)

Рецензенти:

В. В. Ткачов, завідувач кафедри автоматизації та комп'ютерних систем Національного гірничого університету, д-р техн. наук, професор

В. С. Ситніков, завідувач кафедри комп'ютерних систем Одеського національного політехнічного університету, д-р техн. наук, професор

Г. Ф. Кривуля, завідувач кафедри автоматизації проектування обчислювальної техніки Харківського національного університету радіоелектроніки, д-р техн. наук, професор

Купін А. І., Музика І. О.

К 92 Мережні інформаційні технології. Практикум. Навч. посіб. – Кривий Ріг : Видавець ФО-П Чернявський Д. О., 2015. – 238 с.
ISBN 978-966-7250-18-9

Даний навчальний посібник присвячено вивченню мережних інформаційних технологій у галузі розробки клієнт-серверного програмного забезпечення, розподілених інформаційних систем та веб-додатків засобами Microsoft .NET. Висвітлені питання дослідження та розробки мережних інформаційних систем із застосуванням засобів та технологій: C#, OOP, .NET Framework, TCP/UDP Sockets, ADO.NET, WinFroms, ASP.NET, HTML, CSS, Java Script.

Посібник призначений для студентів спеціальностей 7.05010201, 8.05010201 «Комп'ютерні системи та мережі», а також для широкого кола фахівців, які цікавляться технологіями побудови мережних інформаційних систем.

ББК 32.973.26-018+33.2

ISBN 978-966-7250-18-9

ЗМІСТ

ВСТУП

Сучасний стан підготовки фахівців за напрямом «Комп'ютерна інженерія» у вищих навчальних закладах вимагає досить динамічних змін майже кожні 3-4 роки. Адже інформаційні технології (ІТ) є однією з нових та найбільш прогресивних галузей у світі. Для забезпечення конкурентоздатності вітчизняних випускників на міжнародному ринку праці актуальним залишається питання цілісності та структури фахових дисциплін, зокрема дисципліни «Мережні інформаційні технології». Це обґрунтовано не лише значним попитом роботодавців, але й досить високим рівнем заробітної плати у даній сфері.

Галузь мережних інформаційних технологій можна умовно розподілити на дві великі сфери: апаратне забезпечення комп'ютерних мереж та програмне забезпечення різноманітних мережних інформаційних систем [13]. Також значна частка вмінь майбутнього ІТ-спеціаліста повинна охоплювати Web-розробку, мережні операційні системи та програмування клієнт-серверних додатків різного призначення, зокрема із використанням систем керування базами даних. Програмування інтернет-додатків на сьогодні вимагає від спеціаліста не тільки ґрунтовних знань таких мов та технологій, як HTML, CSS, PHP, ASP.NET, JavaScript, Flash, а й вмінь освоювати нові системи керування контентом (CMS), фреймворки, платіжні системи та ін.

У даному посібнику представлено матеріали про розробку програмного забезпечення мережних інформаційних систем. У якості інструментарію для розробки обрано платформу Microsoft .NET та мову програмування C#, яка сьогодні користується значним попитом. Посібник містить, окрім опису конструкцій мови, вибіркову інформацію про архітектуру .NET та базову функціональність деяких класів. Детальний опис цих елементів можна зробити лише в

багатотомному виданні, тому значний обсяг інформації та обмежений розмір посібника зумовили деяку конспективність викладок.

Основним завданням даного посібника є формування у майбутніх ІТ-спеціалістів системи спеціальних знань і навичок оволодіння сучасними мережними технологіями та їх практичним використанням для розробки програмних додатків, інформаційних систем та розподілених баз даних.

Для вдалого засвоєння курсу читачеві необхідно володіти навичками програмування на платформі операційної системи Windows, а також мати досвід використання однієї з сучасних мов програмування: C++, Java, Object Pascal, Visual Basic.

Посібник має вузьке спрямування. Перші дві теми – це вступ до алгоритмічної мови C# та опис необхідних елементів .NET для написання простих Windows-проектів у середовищі Visual Studio. Наступні теми присвячені розробці клієнт-серверних додатків з використанням протоколів TCP та UDP, доступу до баз даних на базі технології ADO.NET, розробці розподілених інформаційних систем за допомогою фреймворку WCF. Останні дві теми підручника мають дослідницький характер і розкривають підходи до оцінки інформаційних потоків між підрозділами підприємства та моделі оптимальної архітектури розподілених баз даних.

Кожна тема містить перелік завдань для самостійного виконання. Завдання розподілено за рівнями складності для зручності оцінювання результатів роботи студента.

Посібник призначено для студентів спеціальностей 7.05010201, 8.05010201 «Комп'ютерні системи та мережі», а також для широкого кола фахівців, які цікавляться технологіями побудови мережних інформаційних систем.

ТЕМА 1. ОСНОВИ РОЗРОБКИ МЕРЕЖНИХ ІНФОРМАЦІЙНИХ СИСТЕМ ЗАСОБАМИ МОВИ C#

Мета: розробити елементарні програми на мові C# з використанням базових конструкцій мови, типів змінних, засобів роботи з консоллю.

Технологія .NET – порівняно нова технологія для розробки Web-додатків і програмного забезпечення, орієнтованого на операційну систему Windows. За допомогою .NET можна також розробляти програмне забезпечення для портативних комп'ютерів і мобільних телефонів. Корпорація Microsoft – безумовний лідер у розвитку операційних систем і технологій програмування – відзначає .NET як головну платформу розробки програмного забезпечення на найближчі роки. Цей факт зумовлює для сучасного програміста необхідність володіти архітектурою та алгоритмічними мовами .NET.

Мова C# – алгоритмічна мова, розроблена для написання програм у середовищі .NET. На формування мови C# значний вплив зробили Java та C++. Мова C# поза технологією .NET не існує. Мова ґрунтується на типах даних базової бібліотеки .NET. Для виконання C#-програм необхідне загальномовне середовище виконання (CLR) – віртуальна машина [3, 6, 11].

Базова бібліотека .NET містить тисячі класів та десятки тисяч методів, більшість з яких мають декілька реалізацій. Цю інформацію неможливо запам'ятати, тому при розробці проектів під .NET необхідний онлайн доступ до довідкової системи MSDN Library.

Ключові терміни платформи .NET

CLR (Common Language Runtime) – середовище виконання .NET. Можна розглядати як код, який завантажує програму, виконує та надає їй усі необхідні служби.

CTS (Common Type System) – спільна система типів даних .NET. Розроблена для забезпечення сумісності адаптованих до .NET алгоритмічних мов. CTS надає також правила для означення нових типів даних.

CLS (Common Language Specification) – загальна специфікація алгоритмічних мов. CLS є підмножиною CTS і мінімальним набором стандартів, який повинні підтримувати усі компілятори для .NET.

Керований код (Managed Code) – довільний код, розроблений для виконання в середовищі CLR. Код, який виконується безпосередньо під операційною системою і не потребує .NET платформи, називають некерованим.

IL (Intermediate Language, MSIL) – проміжна мова, на якій написано код, якщо він призначений для завантаження та виконання середовищем .NET. При обробці керованого коду компілятор генерує код на IL, а CLR виконує завершальну стадію компіляції в машинні коди безпосередньо перед виконанням.

Простір імен (Namespace) – логічна схема групування типів відповідної функціональності. Простори імен мають ієрархічну структуру.

Складений модуль (Assembly) – модуль, який містить компільований керований код. На відміну від виконуваних EXE чи DLL файлів, складені модулі містять також *метадані:* інформацію про модуль та всі визначені в ньому типи, методи тощо. Складений модуль може бути приватним (доступним для використання однією аплікацією) або розподіленим (доступним для використання багатьма аплікаціями).

Глобальний кеш модулів (Global Assembly Cashe, GAC) – місце на диску, де зберігаються розподілені складені модулі.

Відображення (Reflection) – технологія, яка передбачає програмний доступ до метаданих складеного модуля.

Компіляція Just-In-Time (JIT) – процес виконання завершальної стадії компіляції з IL у машинний код. Передбачає компіляцію не коду загалом, а лише його частин за необхідністю.

Маніфест (Manifest) – область складеного модуля, що містить метадані.

Домен додатка (AppDomain) – спосіб, за допомогою якого CLR дає змогу різним програмам виконуватися в одному і тому ж просторі процесів.

Прибирання сміття (Garbage collection) – механізм очищення пам'яті, реалізований у .NET.

Процес компіляції та запуску програми

Скомпільований код програми не містить інструкцій асемблера, а лише інструкції на IL. Ці інструкції розташовані у складеному модулі [11, 18, 21, 27]. Сюди ж долучають метадані:

- опис типів даних;
- методи всередині складеного модуля;
- простий кеш (будується на основі вмісту складеного модуля та може бути використаний для перевірки його цілісності);
- інформація про версії складеного модуля;
- інформація про необхідні зовнішні складені модулі;
- інформація про привілеї, необхідні для виконання коду складеного модуля.

Пакет програм збирається зі складених модулів. Один з цих модулів є виконуваним і містить точку входу основної програми, а інші є бібліотеками. .NET завантажує виконуваний модуль, перевіряє його цілісність та метадані, а також рівень привілеїв користувача на відповідність потребам складеного модуля.

На цьому ж етапі CLR також робить перевірку *безпеки коду за типом пам'яті* (Memory Type Safety). Код вважають безпечним за типом пам'яті лише тоді, коли він звертається до пам'яті способами, які може контролювати середовище CLR. Якщо CLR не впевнене у безпеці коду за типом пам'яті, то (залежно від локальної політики безпеки) може відмовити у виконанні коду.

Для виконання коду CLR утворює *процес* операційної системи Windows і зазначає область застосування, в якій розташовано головний *потік* програми. CLR вибирає першу частину коду, який необхідно виконати, компілює її з мови IL на мову асемблера та виконує з відповідного потоку програми. Коли під час виконання трапляється новий метод, він компілюється у виконуваний код. Процес компіляції цього методу відбувається лише один раз. У процесі виконання коду CLR відстежує стан пам'яті та періодично запускає процедуру прибирання «сміття», тобто звільнення пам'яті від об'єктів, на які відсутні вказівники у коді.

Коротко розглянемо елементи середовища програмування Microsoft Visual Studio 2012.

Типи проектів Microsoft Visual Studio

Новий проект можна створити за допомогою меню *File | New | Project* (або натисненням кнопки *New Project*). У діалоговому вікні потрібно вибрати тип проекту та мову програмування. Для C# означені такі типи проектів (конкретний набір проектів залежить від вибору, здійсненого в процесі налаштування середовища).

Таблиця 1.1

Типи проектів Microsoft Visual Studio

Тип проекту	Стартовий код
Windows	
Windows Application	Порожня форма
Class Library	Бібліотека класів, яку можна використовувати в довільному коді, призначеному для платформи .NET
Windows Control Library	Клас .NET, який можна активізувати іншим кодом .NET. Має інтерфейс користувача (подібно компонентам ActiveX)

Тип проекту	Стартовий код
Web Control Library	Елемент керування. Активізується сторінками ASP.NET для генерування HTML-коду представлення елемента керування при перегляді у вікні браузера
Console Application	Програмний код консольного додатка
Windows Service	Служба, яка працює у фоновому режимі Windows
Empty Project	Проект Windows Application без стартового коду
Crystal Reports Windows Application	Порожня форма проекту для генератора звітів Crystal Reports
Office	
Проекти для Microsoft Excel та Word	
Smart Device	
Проекти для Pocket PC, Smartphone та Windows CE	
Database	
Проект для SQL Server	
Web Site	
ASP.NET Web Site	Web-сайт на основі ASP.NET
ASP.NET Web Service	Клас Web-служби
Empty Web Project	Проект ASP.NET Web Site без стартового коду.
Project From Existing Code	
Project From Existing Code	Нові файли для порожнього проекту. Використовують для конвертування існуючого коду C# у проект .NET.

Для початку роботи необхідно завантажити Microsoft Visual Studio останньої версії та запустити його. У даній темі будуть створюватися консольні додатки, як показано на рисунку 1.1.

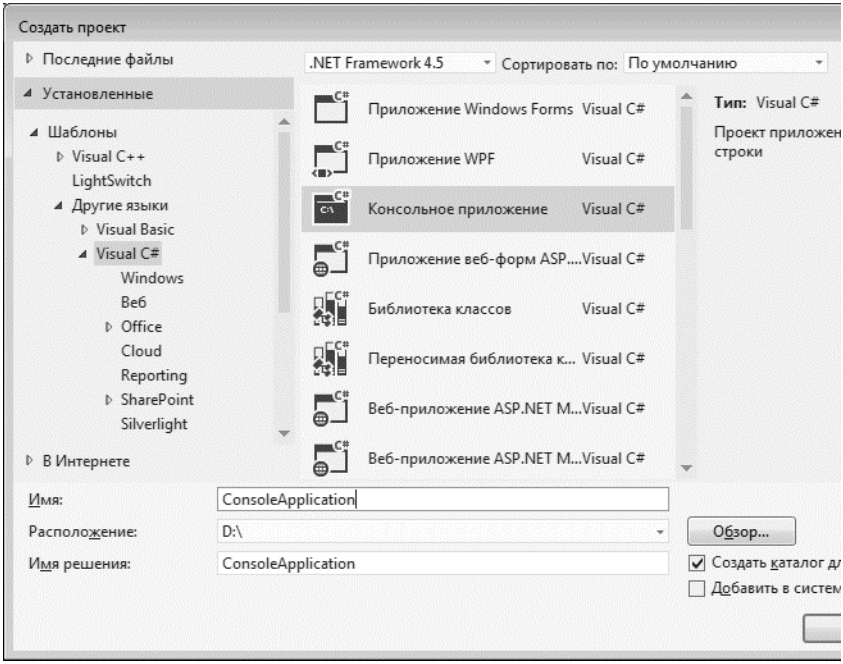


Рис. 1.1. Діалог створення нового проекту

При створенні нового проекту Visual Studio утворює папку проекту такої структури (рисунок 1.2).

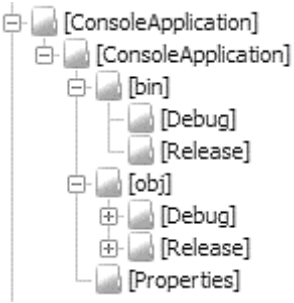


Рис. 1.2. Структура консольного проекту

Каталоги *bin* та *obj* призначені для розташування компільованих і тимчасових файлів [11, 21, 36, 40].

Основний каталог *ConsoleApplication* (назва каталогу відповідатиме назві проекту) призначений винятково для Visual Studio і містить інформацію щодо проекту. У процесі розробки в проект можна додавати нові каталоги.

Рішення та проекти Visual Studio

Рішення (*Solution*) – це набір усіх проектів, який утворює програмне забезпечення для поставленої задачі.

Проект (*Project*) – це набір усіх файлів вихідного коду та ресурсів, які компілюються в один складений модуль (assembly). У простіших проектах складений модуль – це один модуль.

Структура рішення відображається у вікні провідника рішень *(Solution Explorer*). Якщо створити консольний проект, то *Solution Explorer* відображатиме приблизно структуру, як показано на рисунку 1.3.

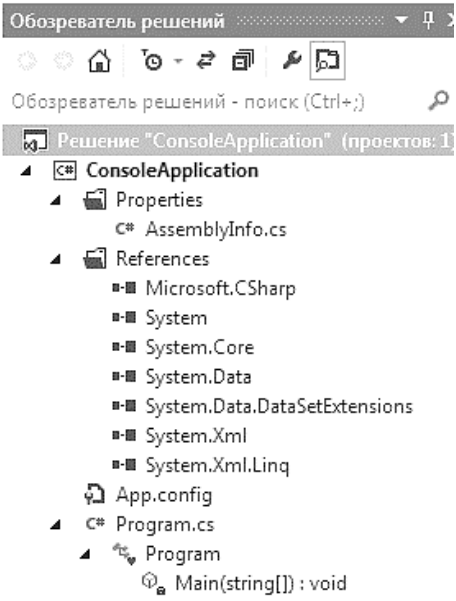


Рис. 1.3. Структура рішення *ConsoleApplication*

Рішення може містити проекти на різних алгоритмічних мовах, адаптованих до .NET. Натиснення правої клавіші миші на довільному пункті активізує контекстне меню, орієнтоване на особливості обраного вузла.

Перегляд та написання коду

Visual Studio містить великий набір інструментів для розробки проектів. Ці інструменти – вікна, які мають декілька режимів розташування та встановлення розмірів (табл. 1.2).

Таблиця 1.2

Режими вікон у середовищі Visual Studio

Режим вікна	Розташування та розміри
Плаваючий *(Floating)*	Окреме вікно із заданими користувачем розмірами.
Прикріплений *(Dockable)*	Вікно розташовується усередині деякого іншого вікна-контейнера. Контейнери можна міняти, перетягуючи вікно за його заголовок. Для прикріпленого вікна контейнер утворює сторінку табуляції. Вікно займає усю вільну область контейнера.
Табульований документ *(Tabbed Document)*	Вікно розташоване усередині вікна Редактора, утворює відповідну закладку. Займає всю вільну область контейнера або ділить область Редактора навпіл за горизонталлю або за вертикаллю.

Вікно можна зробити невидимим *(Hide)*, що є аналогом закритого. Для активізації невидимого вікна його потрібно знайти у списку меню *View*. Контейнер можна зробити прихованим *(Auto hide*). У цьому випадку вікно ховається за одним із країв екрана і з’являється при наближенні курсора миші.

Редактор Visual Studio містить усі стандартні можливості редакторів тексту, форм, ресурсів та інших елементів проекту (рис. 1.4). Блоки коду (класи, члени класу, цикли, набори однотипних рядків та ін.) розглядаються як елементи дерева. Їх можна згорнути або розгорнути, акцентуючи увагу лише на необхідному коді. За допомогою директив `#region` та `#endregion` можна формувати свої гілки дерева перегляду коду. Редактор коду використовує технологію IntelliSense. Зокрема, виводиться контекстний список можливих елементів після крапки наприкінці назви класу. Цей список можна також активізувати комбінацією клавіш Ctrl+Space. Ctrl+Shift+Space активізує список та опис параметрів методів. Редактор проводить часткову синтаксичну перевірку коду. Синтаксичні помилки підкреслюються хвилястою лінією. При наведенні курсора миші на підкреслене слово Visual Studio виводить віконце з описом помилки.

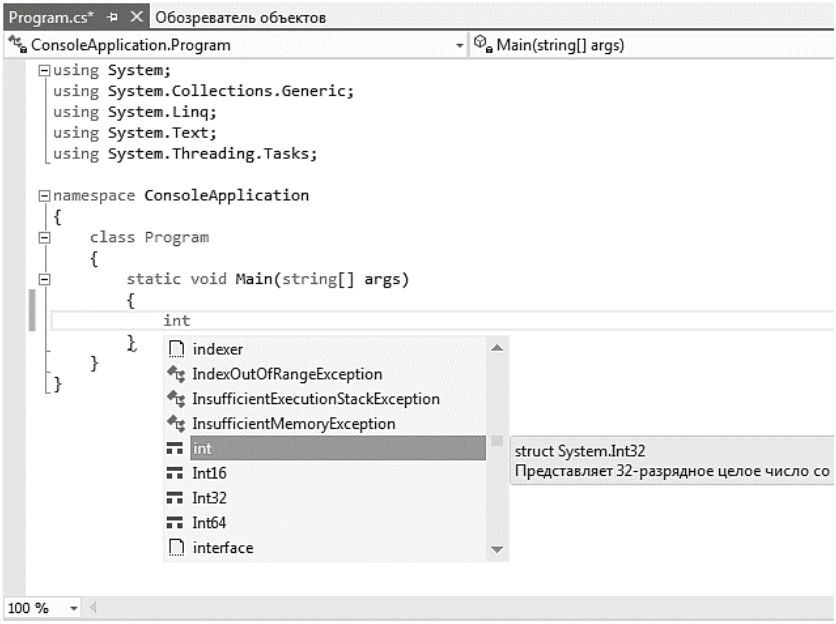


Рис. 1.4. Редактор коду з підказкою синтаксису IntelliSense

Вікно інструментів (*ToolBox*) містить згруповані за категоріями компоненти .NET, які використовують при розробці застосувань. Компоненти перетягують у програму за допомогою миші. Можна додавати власні категорії елементів (контекстне меню *Add Tab*). Елементи ActiveX та компоненти COM долучають опцією меню *Customize ToolBox*. Вікно властивостей (*Properties*) відображає та дає змогу редагувати значення властивостей і подій (*events*) активного (виокремленого) керуючого елемента (компонента). Властивості та події можна впорядкувати за категоріями чи алфавітом. Виокремлений елемент супроводжується коротким описом.

Вікно класів *(Class View*) дає ієрархічний список просторів імен, класів та об'єктів проекту. Список можна сортувати та групувати за категоріями. Контекстне меню для активного елемента списку містить опцію Go To Definition (F12) – перехід до означення активного елемента в коді програми (рис. 1.5).

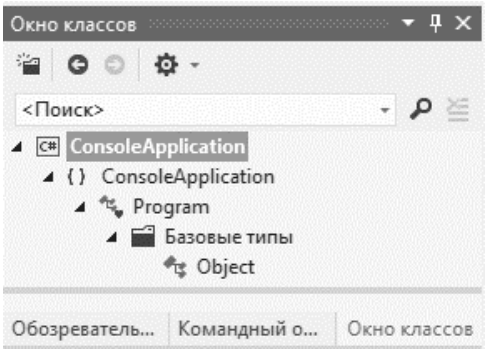


Рис 1.5. Вікно класів проекту *ConsoleApplication*

Браузер об'єктів *(Object Browser)* містить ієрархічний список класів проекту. Причому, на відміну від вікна Class View, передбачає перегляд просторів імен та класів у всіх складених модулях, які використовує проект. Вікно реалізоване подібно до файлового провідника: ліва панель

показує дерево класів, а права – члени класу та опис синтаксису (рис. 1.6).

Для перегляду COM-об'єктів доцільно використовувати програму OLEVIEW або інтерфейси .NET обгортки над COM-об'єктом, автоматично згенерованої середовищем розробки.

Серверний провідник *(Server Explorer)* надає інформацію про комп'ютер: з'єднання з базами даних, служби, Web-служби, запущені процеси, журнали подій та інше. Server Explorer з'єднаний з вікном властивостей *Properties*: вибір елемента ініціює налаштування вікна *Properties* на показ властивостей цього елемента.

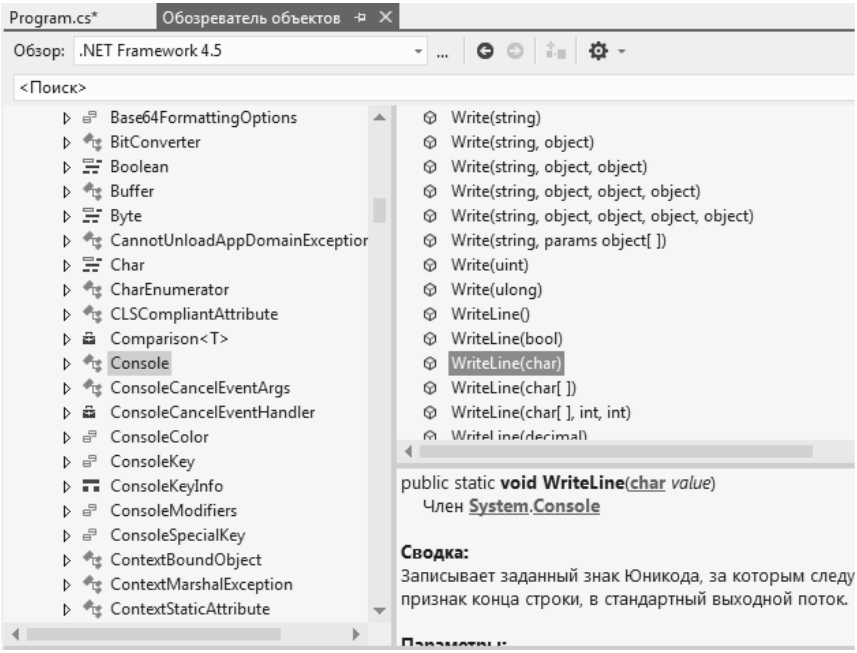


Рис. 1.6. Вікно браузера об'єктів

Розробка проекту

При створенні проекту Visual Studio .NET автоматично генерує дві конфігурації: *Debug* та *Release*. Головною відмінністю конфігурації *Debug* від *Release* є те, що оптимізація

коду не проводиться, а у виконавчі файли додається інформація відлагодження. Очевидно, що відлагоджений продукт повинен розповсюджуватися у конфігурації *Release*. Visual Studio дає змогу також утворювати власні конфігурації. Для вибору конфігурації використовують опцію меню *Debug | Set Active Configuration*, а для редагування – *Project | Properties*. Процес розробки проекту загалом містить етапи проектування, кодування (написання програм) та відлагодження (пошук і виправлення помилок).

Одним з головних інструментів відлагодження є точки переривання (*Break Points*) процесу виконання програми. У Visual Studio для точок переривання можна задавати умови (опція меню *Debug | Break Points*), зокрема:

– задати лічильник, за умови досягнення яким заданого значення здійснити переривання;

– здійснити переривання через певну кількість проходів;

– додати точку переривання для змінної (спрацьовує за умови зміни значення).

Важливим є також дослідження стану об’єктів у довільній точці виконання програми. З цією метою використовують такі вікна:

– *Autos* – відстежує декілька останніх змінних, доступ до яких здійснювався в процесі виконання програми;

– *Locals* – локальні змінні методу, який виконується;

– *Watch* 1, *Watch* 2, ... – явно задані змінні.

У процесі відлагодження корисним може бути вікно *Call Stack* (стек викликів), яке містить упорядкований перелік методів, які виконуються в поточний момент часу, а також значення аргументів цих методів. Вікно Exceptions (винятки) дає змогу зазначити, які дії виконати при генеруванні обраного винятку. Можна обрати і варіант продовження виконання, і варіант переходу до відлагодження. В останньому випадку виконання програми призупиняється, а відлагоджувач переходить до відповідного оператора `throw`. Меню *Debug* містить значну кількість опцій активізації інших корисних інструментів відлагодження програми [22, 34, 38].

Проста консольна програма

Розглянемо код:

```
using System; // використання простору імен System
namespace ConsoleApplication // простір імен
{
    class Program // головний клас програми
    {
        // початок програми
        static void Main(string[] args)
        {
            // виведення строки тексту в консоль
            Console.WriteLine("Hello World!");
            // затримка зчитування клавіші
            Console.ReadKey();
        }
    }
}
```

У C# кожен оператор закінчується символом «`;`». Для об'єднання операторів у блоки використовують фігурні дужки: `{...}`. Текст, розташований між наборами символів `/*` та `*/`, є коментарем і компілятором ігнорується. Текст, розташований після символів `//` і до кінця рядка, також є коментарем. Весь код програми повинен міститися в класі або в іншому означенні типу. Класи (та інші типи) на платформі .NET організовуються у *простори імен (namespaces).* Якщо простір імен не зазначено, то клас належить неіменованому загальному простору імен. Директива `using` дає вказівку компілятору, що неописані в поточному просторі імен класи потрібно шукати в поданому списку просторів імен.

Кожен виконуваний файл C# повинен мати точку входу – метод `Main`. Цей метод може не повертати результат (тип `void`) або повертати ціле число (тип `int`). Означення методів у C# таке:

```
[модифікатори] тип_результату НазваМетоду
( [параметри])
{
// тіло методу
}
```

Тут квадратні дужки позначають необов'язкові елементи. Модифікатори використовують для встановлення рівня доступу до методу.

Типи даних

C# підтримує загальну систему типів CTS. Кожен тип CTS є класом і володіє методами, корисними для форматування та перетворення типів.

Дані програми зберігаються у *стеку.* Стек зберігає дані фіксованої довжини, наприклад, цілі значення. Якщо деяка функція *A* активізує функцію *B*, то всі змінні, які є локальними щодо функції *A*, зберігаються у стеку. Після виконання коду функції *B* керування повертається функції *A*. У цьому випадку зі стека зчитуються збережені значення локальних змінних.

Дані змінної довжини зберігаються в динамічно розподіленій пам'яті *(Heap Allocation*). Динамічну пам'ять використовують також для збереження даних, тривалість життя яких повинна бути довшою за тривалість життя методу, у якому їх означено [6, 24, 30].

C# ділить свої типи даних на дві категорії залежно від місця зберігання. *Змінні типів за значенням* зберігають свої дані в стеку, а *змінні типів за посиланням* – у динамічній пам'яті. Операція присвоєння одній змінній за значенням іншої змінної за значенням утворює дві різні копії одних і тих же даних у стеку. Операція присвоєння одній змінній за посиланням іншої змінної за посиланням спричинює виникнення двох посилань на одну й ту ж ділянку пам'яті, тобто реально дані не дублюються. У C# базові типи даних, такі, як `bool` і `long`, є типами за значенням. Здебільшого складні типи C#, у тім числі й класи, означені користувачем, є

типами за посиланням. Зауважимо, що на відміну від C++, де тип `struct` є специфічним класом, у C# тип `struct` є типом за значенням. У C# ціле число можна оголосити з використанням типу CTS:

```
System.Int32 x;
```

Однак це виглядає неприродно для програміста C. Тому у C# означені псевдоніми *(aliases)* типів CTS. C# має 15 базових типів: 13 типів за значенням та 2 типи (`string` та `object`) за посиланням.

Типи за значенням. C# вимагає, щоб кожна змінна за значенням була явно ініціалізована початковим значенням до того, як її використовуватимуть в операціях.

Таблиця 1.3

Перелік типів за значенням

Тип C#	Тип .NET Framework	Діапазон	Опис та розмір
Цілі типи			
`sbyte`	`System.SByte`	-128 – 127	8-розрядне знакове ціле число
`byte`	`System.Byte`	0 – 255	8-розрядне ціле число без знака
`short`	`System.Int16`	-32768 – 32767	16-розрядне знакове ціле число
`ushort`	`System.UInt16`	0 – 65535	16-розрядне ціле число без знака
`int`	`System.Int32`	-2 147 483 648 – 2 147 483 647	32-розрядне знакове ціле число

Тип C#	Тип .NET Framework	Діапазон	Опис та розмір
`uint`	`System.UInt32`	0 – 4 294 967 295	32-розрядне ціле число без знака
`long`	`System.Int64`	-9223372036854775808 – 9223372036854775807	64-розрядне ціле число зі знаком
`ulong`	`System.UInt64`	0 – 18446744073709551615	64-розрядне ціле число без знака
Числа з плаваючою крапкою			
`float`	`System.Single`	$\pm 1{,}5*10^{-45}$ – $\pm 3{,}4*10^{38}$	32-розрядне дійсне число
`double`	`System.Double`	$\pm 5{,}0*10^{-324}$ – $\pm 1{,}7*10^{308}$	64-розрядне дійсне число
`decimal`	`System.Decimal`	$(-7{,}9*10^{28} – 7{,}9*10^{28}) / / (10^{0–28})$	128-розрядне фінансове число
Логічний тип			
`bool`	`System.Boolean`	`false, true`	8-бітне логічне значення
Символьний тип			
`char`	`System.Char`	U+0000 – U+FFFF	16-розрядний символ Unicode

Змінним цілого типу можна присвоювати значення у десятковій та шістнадцятковій системах числення. Остання вимагає префікс `0x`:

```
long x = 0x1ab;
```

Можна використовувати явно типізовані значення:

```
uint ui = 12U;
long l = 12L;
ulong ul = 12UL;
double f = 1.2F;
decimal d = 1.20M;
```

Десятковий тип `decimal` призначений для забезпечення високої точності фінансових операцій. Змінні логічного типу можуть набувати лише значення `false` або `true`. Значення змінних символьного типу – це символи Unicode. У коді символи розташовують між одинарними лапками. Наприклад: `‘a’, ‘c’, ‘\u0041’, ‘\x0041’`. Останні два значення – це символи, задані своїми номерами. Передбачено також зведення типів: `(char) 65`.

Існують ще спеціальні символи:

`\'` – одинарна лапка
`\"` – подвійна лапка
`\\` – зворотний слеш
`\o` – null-значення
`\e` – увага
`\b` – повернення назад на 1 символ
`\f` – подання сторінки
`\n` – новий рядок
`\r` – повернення каретки
`\t` – символ табуляції
`\v` – вертикальна табуляція

Типи за посиланням. Нехай існує деякий клас `myClass`. Розглянемо рядок коду

```
myClass objMyClass;
```

Цей код формує посилання (тобто резервує `sizeof(int)` байт для розташування адреси) на ще неутворений об'єкт `objMyClass`. Посилання `objMyClass` матиме значення `null`. У C# утворення екземпляра об'єкта за посиланням вимагає ключового слова `new`:

```
objMyClass = new myClass();
```

Даний код утворює об'єкт у динамічній пам'яті та його адресу записує в `objMyClass`. C# містить два базових типи за посиланням:

– `object` (тип CTS `System.Object`) – кореневий тип, який успадковують усі інші типи CTS (у тім числі типи за значенням);

– `string` (тип CTS `System.String`) – рядок символів Unicode.

У C# тип `object` є стартовим типом-предком, від якого беруть початок усі внутрішні та всі визначені користувачем типи. Цей тип реалізує низку базових універсальних методів: `Equals(), GetHashCode(), GetType(), ToString()` та інші. Класам користувача, можливо, доведеться замінити реалізації деяких із цих методів, використовуючи об'єктно-орієнтований принцип перекриття (*overriding*).

При оголошенні змінної за посиланням типу `object` або похідного від `object` класу цій змінній початково надається значення `null` (за умови, що ця змінна є частиною класу). Якщо ж неініціалізована змінна оголошена в межах методу, у програмі виникне помилка на етапі компіляції.

Об'єкт `string` дає змогу зберігати (в динамічній пам'яті) набір символів Unicode. Зауважимо, що коли одну рядкову змінну присвоїти іншій, то в результаті одержимо два посилання на один і той же рядок у пам'яті. Якщо ж подальший код вносить зміни в один із цих рядків, то утворюється новий об'єкт `string`, водночас інший рядок залишається без змін. Рядкові літерали розташовуються у подвійних лапках `"..."`. Рядки C# можуть містити ті ж символи, що й тип `char`. Символ

«\», якщо він не призначений для означення спеціального символу, подвоюється:

```
string filepath = "C:\\Program Files\\F.cs";
```

Рядковому літералу може передувати символ @, який дає вказівку всі символи трактувати за принципом «як є»:

```
string filepath = @"The file
    C:\Program Files\F.cs is C#-file";
```

Значення цього рядка містить також всі пробіли перед C:\.

Структури та переліки

Структура – це подібний до класу тип даних за значенням [11, 18, 24]. Оскільки структура розташовується в стеку, вона утворюється більш ефективно, ніж клас. До того ж копіюється простим присвоєнням. Структура ініціалізується відразу після свого оголошення, а всі поля встановлюються у значення за замовчуванням (0, `false`, `null`). Однак компілятор не дає змоги копіювати одну структуру в іншу до її ініціалізації за допомогою ключового слова `new`.

Приклад:

```
public struct Student
{
    public string FirstName;
    public string LastName;
    public string Group;
}

Student st1, st2;
st1 = new Student;
st1.FirstName = "Леся";
st1.LastName = "Басюк";
st1.Group = "КСМ-13-1";
st2 = st1;
```

У C# структура може виконувати більшість функцій класу (однак не підтримує наслідування реалізацій). Оскільки екземпляри структур розташовані в стеку, доцільно їх використовувати для представлення невеликих об’єктів. Це одна з причин, за якої їх застосовують для реалізації всіх інших типів даних за значенням у платформі .NET.

Перелік – це означуваний користувачем цілий тип даних. Переліки мають тип за значенням. При оголошенні переліку вказується набір допустимих значень, які можуть набувати екземпляри переліку:

```
public enum Light
{
    Green = 0,
    Yellow = 1,
    Red = 2
}

Light l;
l = Light.Red;
```

Масиви

Масив – це колекція змінних однакового типу, звернення до яких відбувається з використанням загального для всіх імені [3, 6, 11].

Для оголошення одновимірного масиву використовують такий синтаксис:

```
тип [] ім'я_масиву = new тип [розмір];
```

Наприклад,

```
int[] arrInt;
int[] arrInt = new int[10];
```

Масиви у C# є 0-базованими, тобто перший елемент масиву має індекс 0:

```
arrInt[0] = 100;
```

Для встановлення розміру масиву використовують число, константу або змінну. Установити розмір можна також наданням значень елементам масиву при утворенні масиву:

```
string[] str = new string{"1-й", "2-й"};
```

Масиви є класами у C# і володіють певними властивостями та методами. Наприклад,

```
//повертає розмірність масиву
int L = arrInt.Length;
//повертає розмірність заданого виміру
//для багатовимірних масивів
int L = arrInt.GetLength(0);
```

До масивів застосовують статичні методи класу `Array`. Наприклад, елементи масиву можна:

```
//сортувати у порядку зростання
Array.Sort (arrInt);
//змінити напрям сортування
Array.Reverse(arrInt);
```

У C# підтримуються багатовимірні масиви двох типів: прямокутні та ортогональні.

Роботу з прямокутними масивами демонструють такі приклади:

```
int[,] arr2Int = new int[5,10];
string[,] studList = {{"Олександр","Сенько"},
                      {"Галина","Вітковська"},
                      {"Петро","Петровський"}
};
```

```
int[,,] arr3lnt = new int[5,10,5];
arr3lnt[0,0,0] = 100;
```

В ортогональних масивах кожен вимір може мати свою довжину:

```
int[] [] a = new int[3][];
a[0] = new int [5];
a[1] = new int [3];
a[2] = new int [10];
```

Типи внутрішніх масивів не обов'язково збігаються з оголошеними:

```
int [][,] b = new int[3] [,];
b[0] = new int[5,2];
```

На відміну від прямокутних масивів кожен індекс в ортогональних масивах виокремлено набором квадратних дужок:

```
a[0][0] = 100; b[0][0, 0] = 100;
```

Перетворення типів

C# дає змогу присвоїти значення змінних одного типу змінним деяких інших типів. Це присвоєння може бути явним або неявним.

Виконати неявні перетворення для цілих типів можна лише тоді, коли перетворюються цілі у більш крупніші цілі або цілі без знаку в цілі зі знаком такого ж розміру. В інших випадках компілятор генерує помилку. Наприклад,

```
byte b1 = 10;
byte b2 = 5;
byte b = b1 + b2; //помилка компіляції
int i = b1 + b2;  //коректне неявне перетворення
```

Довільне ціле число можна перетворювати в типи `float, double` та `decimal`. Однак можлива втрата точності для перетворень із `int, uint, long` або `ulong` до `float` та з `long` або `ulong` до `double.`

Дозволеними є також неявні перетворення з типу `float` у тип `double` та з типу `char до ushort, int, uint, long, ulong, float,double i decimal.`

Явне перетворення типів використовують з метою уникнення помилки компіляції при неявному перетворенні. Типовий синтаксис:

```
long l = 1000;
int i = (int)l;
```

Якщо значення, яке явно перетворюється, лежить за межами діапазону значень типу призначення, то помилка не виникає, а результат перетворення залежатиме від конкретного типу. Для перевірки коректності перетворення використовують оператор `checked`:

```
int i = checked((int)4);
```

Якщо значення l виходить за межі значень типу `int`, то оператор `checked` згенерує виняткову ситуацію (переповнення). Типи за значенням допускають перетворення лише у числові типи, типи `enum` і тип `char`. Не можна безпосередньо перетворити тип `bool` до довільного іншого, і навпаки.

Тип рядка перетворюється в інші типи (і навпаки) за допомогою відповідних методів .NET. Оскільки кожен клас C# є нащадком класу `object`, то він успадковує або має власну реалізацію методу `ToString()`:

```
int i = 1; string s = i.ToString();
```

Для переводу рядкового значення у числове або значення типу `bool` використовують метод `Parse`:

```
string s = "1"; int i = Int32.Parse(s);
```

Широкий набір методів перетворення типів надає також клас `System.Convert.`

Упакування та розпакування типів

Упакування (boxing) та розпакування дає змогу перетворювати типи за значенням у типи за посиланням і навпаки.

Приклад упакування:

```
int i = 10;
object obj = i;
```

Приклад розпакування:

```
int j = (int)obj;
```

Розпаковувати можна лише змінну, яка попередньо була упакована. Змінна, у яку розпаковують об’єкт, повинна мати достатній розмір для розташування усіх байтів змінної, яку розпаковують.

Змінні та константи

Змінні оголошуються у C# із використанням такого синтаксису:

```
[модифікатори] тип_даних ідентифікатор;
```

Якщо в одному виразі оголошуються та ініціалізуються кілька змінних, то всі вони матимуть однаковий тип і модифікатори. Змінні з однаковими іменами не можуть оголошуватися двічі в одній області видимості. Ідентифікатори чутливі до регістру символів. Ідентифікатори повинні

починатися літерою або символом «_». Ідентифікаторами не можуть бути ключові слова C#. Якщо необхідно використати ключове слово як ідентифікатор, то перед ним ставлять символ @. Цей символ дає вказівку компілятору сприймати такі символи як ідентифікатор, а не ключове слово. Ідентифікатори можна записувати і за допомогою символів Unicode `\uxxxx`. Наприклад, `\u050fidentifier`.

Змінні з однаковими ідентифікаторами можна використовувати за умови їхнього розташування у різних областях видимості:

```
for (int i = 0; i < 10; i++) {...};
for (int i = 0; i < 50; i++) {...};
```

Дві змінні з однаковими ідентифікаторами можна використовувати у спільній області видимості лише у випадку, коли одна змінна – поле класу, а друга – локальна змінна, оголошена в методі класу, наприклад:

```
public class A
{
    private int j = 10;
    public int Method()
    {
        int j = 20;
        return j + this.j;
    }
}
```

Ключове слово `this` вказує, що елемент (змінна `j`) належить поточному екземпляру класу.

Константи – це змінні, описані з використанням модифікатора `const`, наприклад:

```
const int a = 10;
```

Константи є подібними до статичних полів лише для читання із такими розбіжностями:

– локальні змінні та поля можуть бути оголошені константами;

– константи ініціалізуються в оголошенні;

– значення константи повинно бути обчислюваним під час компіляції, отож константу не можна ініціалізувати виразом із використанням змінної;

– константа завжди є статичною.

Скорочений запис операцій

Операції інкремента та декремента (`++ та --`) можуть стояти як до, так і після змінної. Вирази `x++` та `++x` еквівалентні виразу `x = x + 1`. Однак префіксний оператор (`++x`) збільшує значення `x` до його використання у виразі. Навпаки, постфіксний оператор (`x++`) збільшує значення x після обчислення виразу. Наприклад:

```
bool b1, b2;
int x = 0;
b1 = ++x == 1; // b1 = true
b2 = x++ == 2; // b2 = false
```

Префіксний і постфіксний оператори декремента поводять себе аналогічно, однак зменшують операнд. Інші скорочені оператори (`+= -= *= /= %= |= ^= <<= =>>`) вимагають двох операндів. Їх використовують для зміни значення першого операнда шляхом виконання над ним арифметичної, логічної чи бітової операції. Наприклад, оператори `x += y;` та `x = x + y;` є еквівалентними.

Таблиця 1.4

Перелік операцій C#

Категорія	Операції
Арифметичні	`+ - * / %`
Логічні	`& | ^ ~ && || !`
Конкатенація рядків	`+`
Інкремент та декремент	`++ --`

Категорія	Операції
Побітовий зсув	`<< >>`
Порівняння	`== != < > <= >=`
Присвоєння	`= += -= *= /= %= |= ^= <<= =>>`
Доступ до членів (для об'єктів)	`.`
Індексування масивів та індексаторів	`[]`
Перетворення типу	`()`
Умовна (тернарна операція)	`?:`
Створення об'єкта	`new`
Інформація про тип	`sizeof is typeof`
Керування винятками переповнення	`checked unchecked`
Розіменування та адресація	`* -> & []`

Зазначимо, що чотири із цих операцій (`sizeof, *, ->, &`) доступні лише в «небезпечному коді» (`unsafe` – код, для якого C# не виконує перевірку безпечності за типом).

Таблиця 1.5

Пріоритет операцій C#

Група	Операції
Унарні	`() . [] x++ x-- new typeof sizeof checked unchecked + - ! ++x –x` та операції приведення типів
Множення / ділення	`* / %`
Додавання / віднімання	`+ -`
Операції побітового зсуву	`>> <<`
Відношення	`< > <= >= is`
Порівняння	`== !=`
Побітовий AND	`&`
Побітовий XOR	`|`
Побітовий OR	`^`
Логічний AND	`&&`

Група	Операції
Логічний OR	`||`
Тернарний оператор	`?:`
Присвоєння	`= += -= *= /= %= |= ^= <<= =>>`

Тернарний оператор ?:

Тернарний оператор `?:` є скороченою формою конструкції `if...else`. Його назва вказує на використання трьох операндів. Оператор обчислює умову та повертає одне значення у випадку, якщо умова істинна, та інше – у протилежному випадку. Синтаксис оператора:

```
умова ? значення_істина : значення_хибність
```

Наприклад:

```
string s = (x >= 0 ? "Додатне" : "Від'ємне");
```

Оператори checked та unchecked

Використання оператора `checked` ми вже демонстрували, розглядаючи перетворення типів. Зауважимо, що перевірку на переповнення можна увімкнути відразу для всього коду, виконавши компіляцію програми з опцією `/checked`. Власне щодо цього випадку може виникнути потреба відміни перевірки на переповнення деякого коду, для чого й використовують оператор `unchecked`.

Оператор sizeof

Розмір (у байтах), необхідний для збереження у стеку змінної за значенням, можна визначити за допомогою оператора `sizeof`:

```
int L;
unsafe
```

```
{
    L = sizeof(double);
}
```

У результаті змінна L набуде значення 8. Зауважимо, що оператор `sizeof` можна використовувати лише в блоках коду, який не є безпечним. За замовчуванням компілятор C# не сприймає такий код. Тому його підтримку необхідно активізувати або використанням опції командного рядка компілятора `/unsafe`, або встановленням у значення `true` пункту *Allow unsafe code blocks* на сторінці властивостей проекту.

Умовні оператори

C# має два умовні оператори: `if` та `switch`.

Синтаксис оператора `if` такий:

```
if (умова)
    оператор(и)
[else
    оператор(и)]
```

Квадратні дужки позначають необов'язкову частину оператора `if`. *Умова* повинна повертати результат логічного типу: `true` або `false`. Якщо потрібно виконати декілька операторів, то їх об'єднують у блок (розташовують у фігурних дужках `{...}`).

Оператор `switch...case` призначений для вибору одного з варіантів подальшого виконання програми з декількох. Синтаксис оператора такий:

```
switch (вираз)
{
    case константа:
        оператор(и)
        оператор переходу
```

```
    [default:
        оператор(и)
        оператор переходу]
}
```

Якщо *вираз* дорівнюватиме значенню однієї з позначок `case`, то виконуватиметься наступний за цією позначкою код. Зауважимо, що цей код не обов'язково оформляти як блок. Однак він повинен завершуватися оператором переходу (зазвичай, це оператор `break`). Єдиний виняток – кілька позначок `case` поспіль:

```
switch (s)
{
    case "A":
    case "D":
        i = 1;
        break;
    case "C":
        i = 2;
        break;
    default
        i = 0;
        break;
}
```

Константа може бути константним виразом. Дві позначки `case` не можуть набувати однакового значення.

Цикли

C# реалізує чотири типи циклів. Цикл `for` має такий синтаксис:

```
for ([ініціалізатор]; [умова]; [ітератор])
    оператор(и)
```

Тут *ініціалізатор* – вираз, який обчислюється до початку виконання циклу (зазвичай, тут ініціалізується локальна змінна, яку використовують як лічильник циклу), *умова* – вираз, який обчислюється перед виконанням кожної ітерації циклу (наприклад, перевірка того, що лічильник циклу менший за деяке значення), *ітератор* – вираз, який виконується після кожної ітерації циклу (наприклад, зміна лічильника циклу).

Кожен (або всі) із цих елементів може бути пропущений.

Цикл for є циклом із передумовою. Таким є й цикл `while`:

```
while (умова)
    оператор(и)
```

Цикл `do...while` є прикладом циклу з постумовою:

```
do
    оператор(и)
while (умова);
```

Оскільки *умова* перевіряється після виконання тіла циклу, то хоча б одна ітерація виконається обов'язково. Усі розглянуті цикли завершують свої ітерації, якщо *умова* набуде значення `false`.

C# пропонує ще один механізм циклів – `foreach`:

```
foreach (тип ідентифікатор in вираз)
    оператор(и)
```

Тут *тип* – тип ідентифікатора, *ідентифікатор* – змінна циклу, що представляє елемент колекції або масиву, а *вираз* – об'єкт колекції або масив (або вираз над ними).

Цикл `foreach` дає змогу проводити ітерацію по кожному об'єкту в контейнерному класі, який підтримує інтерфейс `IEnumerable`. До контейнерних класів належать масиви C#, класи колекцій у просторі імен `System.Collection` та означені користувачем класи колекцій:

```
int even = 0, odd = 0;
int[] arr = new int [] {0,1,2,5,7,8,11};
foreach (int i in arr)
{
    if (i%2 == 0)
        even++;
    else
        odd++;
}
```

Зауважимо, що значення об'єкта в колекції всередині циклу `foreach` змінювати не можна.

Оператори переходу

Програма C# може містити позначки – ідентифікатор із двокрапкою. Оператор `goto` дає змогу передати керування рядку програми з позначкою:

```
goto позначка;
```

Не можна передавати керування у блок коду, за межі класу, а також не можна вийти з блоку `finally`, розташованого після блоків `try...catch`. Оператор `goto` може використовуватися для переходів усередині оператора `switch`. У цьому випадку синтаксис такий:

```
goto case константа;
goto default;
```

Оператор `break` використовують для виходу з коду позначки в операторі `switch`, а також для виходу із циклів `for`, `foreach`, `while` та `do...while`.

У циклах також можна використовувати оператор `continue`, який перериває поточну ітерацію та ініціює наступну.

Оператор `return` використовують для припинення роботи поточного методу та передачі керування в метод, який його активізував. Якщо метод повертає значення, то `return` повинен повернути значення відповідного типу:

```
return [вираз];
```

Приклади виконання завдань

Завдання №1

Розглянемо програму, яка узагальнює наведені вище теоретичні відомості про основні конструкції, керуючі оператори та синтаксис мови C#.

```
using System;

namespace ConsoleApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Hello World!!!");
      Console.WriteLine("Натисність будь-яку клавішу");
      Console.ReadKey(); // Очікування введення

      int x = 5;
      double y = 20.2;
      // Виведення числових типів
      Console.WriteLine("У мене було {0} яблук та
                        {1} гривень", x, y);
      Console.WriteLine("Натисність будь-яку клавішу");
      Console.ReadKey();

      // Виведення рядкових змінних
      string str = "У мене було 6 яблук та 40 гривень";
      Console.WriteLine(str);
      Console.ReadKey();
```

```
        // Об’явлення одновимірного масиву
        int[] intArray = new int[10];
         // Об’явлення двовимірного масиву
        int[,] intDoubleArray = new int[10, 10];
        // Ініціалізація одновимірного масиву
        // (приклад циклу for)
        for (int i = 0; i < 10; i++)
        {
            intArray[i] = i;
        }
        // Ініціалізація двовимірного масиву
        // (приклад циклу for)
        for (int i = 0; i < 10; i++)
        {
          for (int j = 0; j < 10; j++)
          {
            intDoubleArray[i, j] = i + j;
          }
        }
        // Сума чисел одновимірного масиву
        // (приклад циклу while)
        int count = 0;
        int sum = 0;
        while (count < intArray.Length)
        {
          sum = sum + intArray[count];
          count++;
        }
        Console.WriteLine("Сума елементів масиву рівна
                           Sum={0}", sum);
       Console.WriteLine("Натисність будь-яку клавішу");
        Console.ReadKey();
    }
  }
}
```

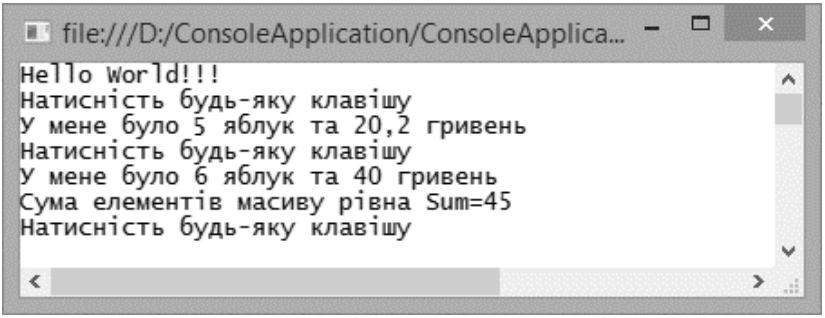


Рис. 1.4. Результат роботи програми №1

Завдання №2

Реалізувати програму обчислення суми квадратів двох цілих чисел, які вводить користувач.

```
using System;
namespace ConsoleApplication
{
  class Program
  {
    static void Main(string[] args)
    {
      string str;

      Console.Write("Введіть змінну X: ");
      str = Console.ReadLine();
      int X = Convert.ToInt32(str);

      Console.Write("Введіть змінну Y: ");
      str = Console.ReadLine();
      int Y = Convert.ToInt32(str);

      Console.WriteLine("X^2+Y^2 = {0}", X*X + Y*Y);
      Console.ReadKey();
    }
  }
}
```

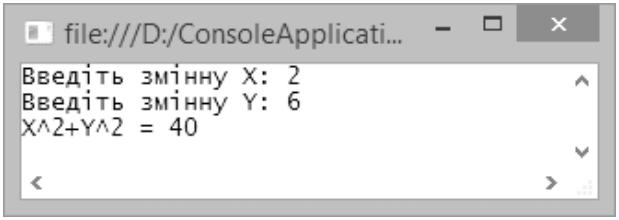


Рис. 1.5. Результат роботи програми №2

Як бачимо, для введення даних користувачем була використана проміжна рядкова змінна `string str;` яка потім за допомогою класу `Convert` конвертувалася в ціле число. Таке перетворення є найбільш уразливим місцем у даній програмі, оскільки у випадку введення букви замість цілого числа виникне помилка та буде згенеровано виняток `System.FormatException`.

Завдання для самостійного виконання

Відповідно до варіанту та обраного рівня складності виконати такі завдання.

№	Рівень завдань		
	Початковий	Базовий	Високий
1	Відкомпілювати та протестувати аудиторні завдання	Завдання №1	Завдання №1
2	–	Завдання №2	Завдання №2
3	–	–	Завдання №3

Завдання №1

№ варіанта	Завдання
1	Вивести на консоль своє ім’я як текст та групу як строкову змінну
2	Реалізувати програму зведення в куб добутку двох цілих чисел

№ варіанта	Завдання
3	Вивести на консоль своє прізвище як змінну строкового типу та свій вік як змінну типу `int`
4	Реалізувати програму зведення в квадрат частки двох цілих чисел
5	Створити дві змінні типу `int`, знайти їх добуток та суму, вивести результат на консоль
6	Створити дві змінні типу `float`, знайти їх середнє арифметичне, результат вивести на консоль
7	Реалізувати програму зведення в квадрат суми двох цілих чисел
8	Створити дві змінні типу `double`, знайти їх суму та різницю, результат вивести на консоль
9	Вивести на консоль своє ім’я та прізвище як текст
10	Створити дві змінні типу `double`, знайти їх суму та вивести на консоль
11	Реалізувати програму обчислення добутку двох дійсних чисел
12	Реалізувати програму обчислення частки двох дійсних чисел
13	Реалізувати програму зведення цілого числа в квадрат
14	Реалізувати програму зведення в куб цілого числа
15	Реалізувати програму зведення в куб суми двох цілих чисел
16	Реалізувати програму зведення в квадрат добутку двох цілих чисел
17	Створити дві змінні типу `string`, склеїти їх та вивести на консоль
18	Реалізувати програму зведення в квадрат суми двох дійсних чисел
19	Реалізувати програму зведення в квадрат різниці двох дійсних чисел

№ варіанта	Завдання
20	Реалізувати програму зведення в квадрат добутку двох дійсних чисел
21	Реалізувати програму зведення в квадрат частки двох дійсних чисел
22	Створити дві змінні типу `int`, знайти їх різницю, якщо вона більше за нуль, вивести її квадрат, якщо менша за нуль, то її модуль
23	Реалізувати програму зведення в квадрат різниці двох цілих чисел
24	Реалізувати програму зведення в куб різниці двох цілих чисел
25	Створити дві змінні типу `double`, знайти їх суму та вивести на консоль
26	Реалізувати програму зведення в куб частки двох цілих чисел
27	Реалізувати програму зведення в куб суми двох дійсних чисел
28	Реалізувати програму зведення в куб різниці двох дійсних чисел
29	Реалізувати програму зведення в куб добутку двох дійсних чисел
30	Реалізувати програму зведення в куб частки двох дійсних чисел

Завдання №2

№ варіанта	Завдання
1	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), знайти їх суму та добуток
2	Створити, ініціалізувати масив з 10 цілих чисел, відсортувати його за зростанням та вивести на консоль

№ варіанта	Завдання
3	Створити, ініціалізувати масив з 20 цілих чисел, знайти їх середнє арифметичне
4	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), вивести на консоль усі числа, більші за число 10
5	Створити, ініціалізувати масив з 10 цілих чисел, відсортувати його за спаданням та вивести на консоль
6	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), вивести на консоль усі числа, менші за число 15
7	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), знайти кількість чисел більших за 10 та вивести отримане число на консоль
8	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), вивести на консоль усі числа, менші за число 20, але більші за 10
9	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), знайти добуток чисел, які менші за середнє арифметичне чисел масиву
10	Створити, ініціалізувати масив з 10 чисел типу `double` у проміжку (0..25), знайти суму чисел більших за 15 і добуток чисел менших за 10
11	Створити, ініціалізувати масив з 8 цілих чисел у проміжку (0..15), знайти їх суму та частку
12	Створити, ініціалізувати масив з 10 цілих чисел, відсортувати його за спаданням та вивести на консоль
13	Створити, ініціалізувати масив з 15 цілих чисел, знайти їх середнє геометричне
14	Створити, ініціалізувати масив з 12 цілих чисел у проміжку (0..35), вивести на консоль усі числа, більші за число 15

№ варіанта	Завдання
15	Створити, ініціалізувати масив з 15 цілих чисел, відсортувати його за спаданням та вивести на консоль
16	Створити, ініціалізувати масив з 15 цілих чисел у проміжку (0..40), вивести на консоль усі числа, менші за число 25
17	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..50), знайти кількість чисел більших за 40 та вивести отримане число на консоль
18	Створити, ініціалізувати масив з 15 цілих чисел у проміжку (0..18), вивести на консоль усі числа, менші за число 15, але більші за 5
19	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), знайти добуток чисел, які менші за середнє арифметичне чисел масиву
20	Створити, ініціалізувати масив з 10 чисел типу `double` у проміжку (0..50), знайти суму чисел більших за 15 і добуток чисел менших за 12
21	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (10..50), знайти їх суму та добуток
22	Створити, ініціалізувати масив з 10 цілих чисел, відсортувати його за зростанням та вивести на консоль
23	Створити, ініціалізувати масив з 20 цілих чисел, знайти їх середнє арифметичне
24	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (5..25), вивести на консоль усі числа, більші за число 12
25	Створити, ініціалізувати масив з 10 цілих чисел, відсортувати його за спаданням та вивести на консоль

№ варіанта	Завдання
26	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (5..35), знайти добуток чисел, які менші за середнє арифметичне чисел масиву
27	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), знайти кількість чисел більших за 10 та вивести отримане число на консоль
28	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (10..60), вивести на консоль усі числа, менші за число 20, але більші за 10
29	Створити, ініціалізувати масив з 10 цілих чисел у проміжку (0..25), вивести на консоль усі числа, менші за число 15
30	Створити, ініціалізувати масив з 15 чисел типу `double` у проміжку (2..5), знайти суму чисел більших за 3 і добуток чисел менших за 4

Завдання №3

Дано матрицю розміру M×M, що заповнена цілими числами у проміжку (-10, 10). Необхідно:

№ варіанта	Завдання
1	Відсортувати за зростанням головну діагональ матриці
2	Знайти суму непарних рядків матриці
3	Знайти добуток парних стовпців матриці
4	Відсортувати побічну діагональ матриці за спаданням
5	Знайти усі елементи матриці, що більші за нуль, та знайти їх суму
6	Відсортувати перший та останній стовпець матриці за спаданням
7	Знайти максимальний елемент кожного стовпця матриці

№ варіанта	Завдання
8	Знайти мінімальній елемент кожного рядка матриці
9	Відсортувати за спаданням парні стовпці матриці
10	Знайти максимальний та мінімальний елемент матриці, та їх суму
11	Відсортувати за зростанням побічну діагональ матриці
12	Знайти суму парних рядків матриці
13	Знайти добуток непарних стовпців матриці
14	Відсортувати побічну діагональ матриці за зростанням
15	Знайти усі від'ємні елементи матриці та знайти добуток їх модулів
16	Відсортувати перший та останній стовпець матриці за спаданням
17	Знайти мінімальний елемент кожного стовпця матриці
18	Знайти максимальний елемент кожного рядка матриці
19	Відсортувати за зростанням парні стовпці матриці
20	Знайти максимальний та мінімальний елемент матриці та їх добуток
21	Знайти усі елементи матриці, що більші за нуль, та знайти їх суму
22	Відсортувати перший та останній стовпець матриці за спаданням
23	Знайти максимальний елемент кожного стовпця матриці
24	Знайти мінімальній елемент кожного рядка матриці
25	Відсортувати за спаданням парні стовпці матриці.
26	Знайти максимальний та мінімальний елемент матриці та їх суму

№ варіанта	Завдання
27	Відсортувати за зростанням побічну діагональ матриці
28	Знайти суму парних рядків матриці
29	Знайти добуток непарних стовпців матриці
30	Відсортувати побічну діагональ матриці за зростанням

Контрольні питання

1. Коротко охарактеризуйте платформу .NET Framework.
2. Які типи даних існують у мові C#?
3. Чим відрізняються типи за значенням від типів за посиланням?
4. Як перетворити строкову змінну у ціле число?
5. Чим відрізняється мова C# від C++?
6. Перелічіть основні бінарні операції та пріоритет їх виконання.
7. Для чого призначені простори імен?
8. Наведіть синтаксис основних операторів керування.
9. Як відкомпілювати файл програми на мові C# у командному рядку?
10. Чим відрізняються настроювання проекту у режимі компіляції *Debug* та *Release*?

ТЕМА 2. ОБ'ЄКТНО-ОРІЄНТОВАНИЙ ПІДХІД ДО СТВОРЕННЯ МЕРЕЖНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

Мета: розробити програми з використанням класів C#, реалізувати перевантаження методів, успадкування класів та дослідити принципи інкапсуляції.

Класи та структури

Алгоритмічна мова C# є цілковито об'єктно-орієнтованою. Програма на C# – це набір класів та маніпулювання ними [3, 11, 21, 22, 33-37].

Класи та структури дуже подібні між собою. Основні розбіжності між ними:

– об'єкти класів мають тип даних за посиланням, а структури – за значенням;

– структури не підтримують спадковість;

– для структури не можна оголосити конструктор за замовчуванням;

– для структури не можна оголосити деструктор;

– неможливо використати ініціалізатори для надання значень полям.

Оскільки екземпляри структур розташовані у стеку, доцільно їх використовувати для представлення невеликих об'єктів. Зокрема, клас `Point`, який розглядатимемо нижче, з успіхом можна було б реалізувати як структуру. Вибір класу зумовлено лише бажанням продемонструвати на простій логічній побудові більше понять класу та структури.

Розглянемо поняття класу на такому прикладі:

```
public class Point
{
  //статичне поле
  private static uint count = 0;
  //поля
  public double x = 0;
```

```
  public double y = 0;
  //властивість лише для читання
  public static uint Count
  {
    get {return count;}
  }
  //перевантажений метод
  public override string ToString()
  {
    return "(" + x + "," + y + ")";
  }
  // метод
  public void Set(double x, double y)
  {
    this.x = x; this.y = y;
  }
  //конструктор без параметрів
  public Point ()
  {
    count++;
  }
  //конструктор з параметрами
  public Point(double x, double y)
  {
    count++;
    Set(x, y);
  }
  //конструктор копій
  public Point(Point a): this(a.x, a.y)
  {
  }
  //деструктор
  ~Point( )
  {
    count--;
  }
}
```

За допомогою модифікатора класу `partial` оголошення класу можна розбити на декілька частин, які можна розташувати як в одному, так і в різних файлах.

Члени класу

Дані та функції всередині класу називають членами класу. Дані класу – це *поля, константи* та *події,* тобто ті члени класу, які містять дані для класу.

Поле – це довільна змінна, оголошена на рівні класу: `count, x, y`. Якщо поле оголошене як статичне (`static`), то воно єдине для всього класу і жоден екземпляр об'єкта класу не матиме окремого екземпляра цього поля. У протилежному випадку для кожного екземпляра об'єкта цього класу утворюються власні незалежні екземпляри полів.

Клас `Point` описує точку: кожен екземпляр класу – це точка на площині з координатами `x` та `y`. Статичне поле `count` містить кількість утворених і активних в поточний момент часу екземплярів класу.

Подія – це член класу, який дає змогу об'єкту надіслати повідомлення про початок певної події (зміна значення поля, взаємодія з користувачем тощо). Клієнт (код, який використовує об'єкт класу) може містити код обробки події.

Функції класу – це ті члени класу, які забезпечують функціональність для роботи з даними класу. Вони містять *методи, властивості, конструктори, деструктори, оператори* та *індексатори* [1, 3, 6, 11].

Модифікатори змінних

Модифікатори змінних описують ті чи інші особливості оголошуваних змінних. Перелічимо модифікатори змінних: `internal, new, private, protected, public, readonly, static, const`.

Модифікатори застосовують лише до полів (членів класів), однак не до локальних змінних. Змінна може мати декілька модифікаторів.

Модифікатор `new` використовують лише в класах, які є похідними від іншого, поля якого потрібно приховати.

Такі модифікатори утворюють групу модифікаторів доступу:

– `public` – змінна доступна скрізь як поле типу, якому вона належить;

– `internal` – змінна доступна лише поточному складеному модулю;

– `protected` – доступ до змінної відкритий лише з класу, якому вона належить, або з похідних класів;

– `protected internal` – те ж саме, що й `protected`, однак доступ лише з поточного складеного модуля;

– `private` – доступ до змінної лише з класу, у якому її оголошено.

За замовчуванням поле є членом екземпляра класу. Тобто кожен екземпляр класу має власне поле (виділену ділянку пам'яті) із відповідним ідентифікатором. Якщо до оголошення поля додати модифікатор `static`, то це поле буде єдиним для всіх екземплярів класу.

Модифікатор `readonly` робить змінну доступною лише для читання. Її значення можна встановити при першому оголошенні (для статичних змінних) або в конструкторі для типу, до якого вона належить (для нестатичних змінних).

Методи

Види функцій класу визначаються синтаксисом оголошення. Синтаксис оголошення методів у C# такий:

```
[модифікатори] тип_результату НазваМетоду([параметри])
{
    //тіло методу
}
```

Модифікатори методу аналогічні модифікаторам змінних. Додатково можна використовувати модифікатори, зазначені в таблиці 2.1.

Таблиця 2.1

Модифікатори методів класу

Модифікатор	Опис
`virtual`	Метод може бути перевизначений у дочірньому класі
`abstract`	Віртуальний метод, який визначає сигнатуру методу, однак не містить його реалізації. За наявності абстрактних методів екземпляр класу не може бути утворений
`override`	Метод перевизначає успадкований віртуальний або абстрактний метод
`sealed`	Метод перевизнчає успадкований віртуальний метод і забороняє його переозначення у дочірніх класах. Використовують разом із `override`
`extern`	Метод реалізований поза програмою на іншій мові

Клас `Point` містить перекритий (`override`) метод `ToString` класу `object`. Для активізації методу потрібно вказати ім'я об'єкта, для якого активізують метод, а після крапки – ім'я методу та список аргументів у дужках:

```
Point a = new Point(1, 1);
string s = a.ToString();
```

Для активізації статичного методу потрібно зазначити назву класу, а не ім'я об'єкта. Статичний метод може працювати лише зі статичними полями класу. Якщо метод активізується всередині класу, то назву класу не використовують.

Аргументи методів можуть передаватися *за посиланням* або *за значенням.* За замовчуванням параметри передаються за значенням, тобто метод одержує копію значення аргументу. Якщо параметр має тип даних за значенням, то довільні зміни цього параметра всередині методу ніяк не вплинуть на значення аргументу-оригіналу. Якщо ж параметр має тип за

посиланням (масив, клас), то метод працюватиме безпосередньо з даними аргументу, оскільки передана копія значення – адреса розташування об'єкта. Зауважимо, що рядки не поводять себе як типи за посиланням, тому зміни у рядку, що відбулися всередині методу, не зачіпають оригіналу.

Якщо потрібно змінні за значенням передати в метод за посиланням, використовують ключове слово `ref` перед типом параметра. Слово `ref` повинно вказуватися і при виклику методу. Довільна модифікація змінної методом викликає відповідні зміни аргументу-оригіналу.

Перед передачею значень методу всі змінні-аргументи повинні бути ініціалізованими, інакше компілятор C# видасть повідомлення про помилку. Обійти це обмеження можна шляхом використання ключового слова `out` перед типом параметра. Слово `out` необхідно зазначити і при виклику методу. Усередині методу параметрові, позначеному як `out`, необхідно присвоїти значення.

Властивості

Властивості використовують з метою зробити виклик методу подібним на поле. Оголошують їх подібно до поля. Однак додатково після оголошення у фігурних дужках розташовують блок коду для контролю даних та реалізації потрібної функціональності. Цей блок може мати два методи доступу: аксесор `get` та аксесор `set`.

Клас `Point` має властивість `Count`, яка повертає кількість утворених екземплярів класу. Розширимо опис цієї властивості:

```
public static uint Count
{
    get
    {
        return count;
    }
```

```
        set
        {
            if (value >= 0)
                count = value;
        }
    }
```

Зауважимо, що аксесору `set` неявно передається параметр з іменем `value` такого ж типу, як і властивість. Аналогічно, значення типу властивості повинен повертати аксесор `get`.

Оскільки властивість подібна до поля, то й активізацію її здійснюють подібно:

```
uint cnt = Point.Count;  // аксесор get
Point.Count = cnt;       // аксесор set
```

Якщо властивість містить лише аксесор `get`, то її використовують тільки для читання значення. Якщо властивість містить лише аксесор `set`, то її використовують тільки для запису значення.

Конструктори

Конструктори – це методи класу, які використовуються разом з оператором `new` для утворення об'єкта. Якщо клас не містить власного конструктора, то компілятор утворить конструктор за замовчуванням (без параметрів). Конструкторів може бути кілька. Вони відрізнятимуться кількістю і типом параметрів, однак матимуть одну й ту ж назву – ім'я класу. Клас `Point` містить три конструктори. Перший із них збільшує лічильник утворених об'єктів, а другий додатково ініціалізує поля x та y. Третій конструктор `public Point (Point a)` належить до так званих *конструкторів копій,* оскільки ініціалізує екземпляр класу на основі значень іншого екземпляра цього ж класу. Для утворення об'єкта можна використати довільний з конструкторів класу, проте лише один.

Конструктор не може повертати значення, отож тип результату не вказують при визначенні конструктора. Якщо конструктор оголошений як `private`, то його можна використовувати лише всередині класу, а якщо як `protected` – то в класах-нащадках. Обмеження доступу до конструктора може бути корисним, наприклад, для методів `Clone(), Copy()` або для аналогічних методів класу, яким потрібно утворювати інші екземпляри цього класу. Очевидно, що клас повинен мати хоча б один конструктор з доступом `public`, якщо передбачається утворення об'єктів цього класу клієнтським кодом.

Клас може також мати статичний конструктор без параметрів. Такий конструктор буде використано лише один раз. Його можна використати для ініціалізації статичних змінних. Наприклад,

```
static Point()
{
    count = 0;
}
```

Статичний конструктор не має модифікатора доступу, оскільки він активізується лише середовищем .NET при завантаженні класу. Зауважимо, що клас може водночас мати конструктор екземплярів без параметрів і статичний конструктор. Це єдиний випадок, коли може бути два методи класу з однаковими назвами та однаковим списком параметрів. Конструктор може викликати інший конструктор цього ж класу. Для цього випадку існує спеціальний синтаксис:

```
public Point(): this(0,0)
{
    // додатковий код
}
```

Такий синтаксис повідомляє компілятору, що у випадку використання конструктора спочатку потрібно виконати інший конструктор цього класу з переданими йому аргументами після

ключового слова this, а потім виконати код (якщо є) зазначеного конструктора.

Ключове слово this вказує, що використовують елемент поточного класу. Якщо ми використаємо ключове слово base, то дамо вказівку шукати відповідний елемент у базовому класі.

Деструктори

Деструктор класу використовують для виконання дій, необхідних при знищенні екземпляра класу. У нашому прикладі класу Point деструктор зменшує лічильник екземплярів класу.

Як і конструктор, деструктор має те ж ім'я, що й клас, однак з префіксом тильда (~): ~Point. Деструктор не має параметрів та не повертає результат. Явним чином деструктори у C# не викликають. Виконання коду деструктора ініціюється механізмом прибирання «сміття». Отож не можна передбачити, коли буде виконано код деструктора.

Ініціювати негайне прибирання «сміття» можна за допомогою методу Collect() об'єкта .NET System.GC, який реалізує «прибиральника». Однак цим доцільно користуватися лише у випадку, коли ви впевнені в необхідності такого кроку. Якщо при знищенні екземпляра класу необхідно негайно звільнити ресурси, зайняті екземпляром класу (наприклад, закрити файл), або надіслати повідомлення іншим об'єктам, доцільно утворити спеціальні методи. Типові назви таких методів – Close та Dispose. Клієнтський код повинен явно активізувати ці методи. І це є недоліком такого підходу.

Інший варіант звільнення ресурсів – використання оператора using. У цьому випадку клас повинен успадковувати інтерфейс IDisposable, означений у просторі імен System:

```
class Point: IDisposable
{
    // інші члени класу
    public void Dispose ()
```

```
    {
        // код
    }
}
```

Перевантаження операцій

Нехай задано точки $A(x_1, y_1)$ та $B(x_2, y_2)$. Точку $C(x, y)$ назвемо сумою точок A та B, якщо $x = x_1 + x_2$, $y = y_1 + y_2$. Для додавання точок у класі `Point` можна утворити метод. Наприклад:

```
public Point Add(Point p)
{
    //код
};
```

Тоді додавання виглядатиме так:

```
C = A.Add(B);
```

Якщо потрібно додати декілька точок, то вираз ускладниться. Значно зручніше використовувати звичний нам синтаксис для простих типів:

```
C = A + B;
```

C# дає змогу перевантажувати операції. Наприклад, до означення класу `Point` можна додати такий код:

```
public static Point operator + (Point p1, Point p2)
{
    return new Point(p1.x + p2.x, pl.y + p2.y);
}
```

Операція оголошується аналогічно методу, за винятком того, що замість імені методу пишуть ключове слово `operator` і знак операції. Тепер, якщо `A`, `B` та `C` мають тип `Point`, то ми можемо записати:

```
C = A + B;
```

Перевантажимо тепер операцію множення на число.

```
public static Point operator * (double a, Point p)
{
    return new Point(a * p.x, a * p.y);
}

public static Point operator * (Point p, double a)
{
    return a * p;
}
```

Для операції множення ми утворили два варіанти перевантаження, щоб компілятор коректно сприймав код, наприклад, `10*A` та `A*10`. Зауважимо, що перевантаження операцій `+, -, *` та `/` використовуються компілятором для реалізації операцій `+=, -=, *=` та `/=` відповідно. C# дає змогу перевантажувати лише операції, зазначені в таблиці 2.2.

Таблиця 2.2

Операції, які дозволяють перевантаження

Категорія	Операції
Арифметичні	`+ - * / % ++ --`
Бітові	`<< >> & | : ! ~ true false`
Порівняння	`== != < >= <= >`

Перевантаження методів

Перевантаження методів використовують у тому випадку, коли потрібно, щоб клас виконував деякі дії, але при цьому існувало кілька способів передачі інформації методу, який виконує завдання. Ми вже демонстрували перевантаження методів на прикладі конструкторів класу `Point` – одна назва за різних наборів параметрів.

Наведемо ще один приклад – перевантаження методу `ToString`:

```
public string ToString(string format)
{
    return String.Format(format, x, y);
}
```

Тепер можна записати код:

```
Point p = new Point(1,1);
string s = p.ToString();               //s = "x=1 y=1"
s = p.ToString("x:{0} y:{1}", x, y);  //s = "x:1 y:1"
```

Кількість перевантажених методів не обмежена. Тобто клас може містити багато методів з одним іменем, однак вони повинні вирізнятися кількістю, порядком або типом параметрів. Очевидно, що не доцільно давати однакове ім'я методам, які виконують зовсім різні задачі.

Приклад використання об'єктів класу

Розглянемо клас, який реалізує функціональність роботи з масивом точок. Наведемо код початкового варіанта такого класу:

```
public class Points
{
    private readonly uint count;
    protected Point[] points;
    public uint Count
    {
        get
        {
            return count;
        }
    }
    public override string ToString()
    {
        string Result = "";
        for (int i = 0; i < count; i++)
```

```
            Result += points[i] + " ";
        return Result;
    }
    public Points(uint count)
    {
        this.count = count;
        points = new Point[count];
        for (int i = 0; i < count; i++)
            points[i] = new Point();
    }
    public Points(Points pts): this(pts.count)
    {
        for (int i = 0; i < count; i++)
            points[i] = pts.points[i];
    }
}
```

Клас `Points` містить масив точок `points` (елементів типу `Point`). Розмірність масиву задається параметром конструктора і встановлюється при утворенні об'єкта класу:

```
points = new Point[count];
```

Зверніть увагу, що цей код виокремить пам'ять для розташування `count` вказівників на об'єкти `Point`, а не власне об'єктів. Тим більше, що самих об'єктів ще не існує:

```
for (int i = 0; i < count; i++)
    points[i] = new Point();
```

Властивість `Count` повертає значення розмірності. Оскільки клас містить конструктори, то конструктор за замовчуванням не утворюється. Тому для утворення об'єкта класу можна використати або конструктор з параметром, який задає розмірність, або конструктор копій.

Індексатори

Індексатори дають змогу здійснювати доступ до об'єкта так, ніби він є масивом. Індексатори означуються приблизно так, як властивості – з використанням функцій `get` та `set`. Однак замість імені індексатора використовують ключове слово `this`.

Якщо `ps` – об'єкт типу `Points`, то для доступу до точки з індексом 0 ми повинні використовувати синтаксис: `ps.points[0]`. Значно елегантніше було б застосувати `ps[0]`, однак для цього потрібно додати індексатор.

Щоб оголосити індексатор для класу `Points`, додамо до його опису такий код:

```
public Points this[int i]
{
  get
  {
    if (i >= 0 && i < count)
      return points[i];
    else
      throw new IndexOutOfRangeException("Вихід
      за допустимий діапазон індексів"+ i);
  }
  set
  {
    if (i >= 0 && i < count)
      points[i]=value;
    else
      throw new IndexOutOfRangeException("Вихід
      за допустимий діапазон індексів"+ i);
  }
}
```

Тепер для змінної `ps` типу `Points` ми можемо використати код:

```
string s = "x0=" + ps[0].x + " y0=" + ps[0].y;
```

Індексатори не є обмежені одномірними масивами та цілочисельними індексами. Для індексаторів можна застосовувати цикли `for`, `do` та `while`, однак не можна написати цикл `foreach`, оскільки він працює лише з колекціями, а не з масивами.

Інтерфейси

Інтерфейс – це список оголошень методів, властивостей, подій та індексаторів. Оголошення інтерфейсу подібне до класу, однак не містить модифікаторів доступу для членів і реалізацій. Інтерфейс не може мати конструкторів. Отож об'єкт інтерфейсу не можна утворити.

Наприклад, інтерфейс `IEnumerator` із простору імен `System.Collections` оголошений так:

```
interface IEnumerator
{
    //властивість
    object Current {get;}
    //методи
    bool MoveNext();
    void Reset();
}
```

Кажуть, що клас підтримує інтерфейс, якщо він містить реалізацію усіх оголошень інтерфейсу. Зокрема, клас підтримує інтерфейс `IEnumerator`, якщо він містить реалізацію властивості `Current` і методів `MoveNext` і `Reset`. За домовленістю назва інтерфейсу починається літерою I.

У попередньому пункті ми оголосили індексатор для класу `Points` і зазначили, що до нього не можна застосувати цикл `foreach`. Для того щоб клас `Points` підтримував колекції, він повинен виконати наперед оголошену домовленість: містити метод з назвою `GetEnumerator`, який повертає об'єкт деякого класу з підтримкою інтерфейсу `IEnumerator`. Це

правило формалізується інтерфейсом IEnumerable, оголошеним у просторі імен System.Collections так:

```
public interface IEnumerable
{
    IEnumerator GetEnumerator();
}
```

Перед тим як додати підтримку цього інтерфейсу до класу Points, утворимо допоміжний клас PointsEnum:

```
class PointsEnum: IEnumerator
{
  int location = -1;
  Points points;
  //конструктор класу
  public PointsEnum(Points points)
  {
    this.points = points;
    location = -1;
  }
  //реалізація членів інтерфейсу
  IEnumerator public object Current
  {
    get
    {
      if (location < 0 || location >= points.Count)
     throw   new   InvalidOperationException("Некоректний
індекс");
      return points[location];
    }
  }
  public bool MoveNext()
  {
    ++location;
    return (location >= points.Count) ? false:true;
  }
```

```
public void Reset()
  {
    location = -1;
  }
}
```

Код `class PointsEnum: IEnumerator` вказує, що клас підтримує інтерфейс `IEnumerator`, тобто містить реалізацію його членів. Якщо необхідно, щоб клас підтримував декілька інтерфейсів, то після двокрапки треба перелічити назви цих інтерфейсів, розділені комами. І, відповідно, реалізувати всі члени цих інтерфейсів. Клас `PointsEnum` працює з об'єктом типу `Points`, який передається параметром конструктора.

Додамо тепер до класу `Points` підтримку інтерфейсу `IEnumerable`:

```
public class Points : IEnumerable
{
    public IEnumerator GetEnumerator()
    {
        return new PointsEnum(this);
    }
}
```

Код `new PointsEnum(this)` утворює об'єкт типу `PointsEnum`, а метод повертає значення типу `IEnumerator`. Оскільки клас `PointsEnum` підтримує цей інтерфейс, то протиріччя тут не буде.

Тепер компілятор не заперечуватиме проти використання `foreach`:

```
Points q = new Points(5);
string s = "";
foreach (Point p in q) s += p.ToString();
```

Успадкування класів

Побудуємо клас, який представлятиме геометричний трикутник. Трикутник визначається трьома точками на площині. Використаємо клас `Points`, який дає змогу будувати множину точок і проводити деякі дії над ними.

```
public class Triangle: Points
{
    public Triangle(double x1, double y1, double x2,
                    double y2, double x3, double y3):
                    base(3)
    {
        points[0].Set(x1, y1);
        points[1].Set(x2, y2);
        points[2].Set(x3, y3);
    }
    public Triangle (Point p1, Point p2, Point p3):
                     base (3)
    {
        points[0] = p1;
        points[1] = p2;
        points[2] = p3;
    }
    public override string ToString()
    {
        return "трикутник " + base.ToString();
    }
}
```

Код `class Triangle:Points` оголошує, що клас `Triangle` успадковує клас `Points`. Це означає, що `Triangle` має всі компоненти класу `Points`: поля `count`, `points`, метод `ToString` та індексатор. Окрім того, оскільки клас `Points` успадковує клас `object` (як і всі типи C#), то клас `Triangle` має також усі компоненти класу `object`.

Класи `object` і `Points` є класами-предками для класу `Triangle`. Клас `Points` (клас, який зазначено після двокрапки в оголошенні нового класу) називають *базовим* або

батьківським класом для класу `Triangle`. Клас `Triangle` називають *похідним* або *дочірнім* для класу `Points`. А для класів `object` і `Points` він буде *нащадком.* Похідний клас може безпосередньо використовувати всі члени базового класу, якщо вони означені з модифікаторами `protected` або `public`.

Однак похідний клас не наслідує конструкторів базового класу (проте може використати, як це зроблено в нашому прикладі). Єдиний виняток – це конструктор за замовчуванням, який викликається конструктором за замовчуванням похідного класу. Якщо похідний клас наслідує усі члени предків, то об'єкт цього класу містить підмножину, яку можна розглядати як об'єкт деякого класу-предка:

```
Triangle T = new Triangle(1,1,2,2,3,3);
Points P = (Points)T;
object obj = (object)T;
IEnumerable ienum = (IEnumerable)T;
```

Приховування компонентів

C# дає змогу замінити члени базового класу в нащадках. Розглянемо такі два класи:

```
public class A
{
    public int x = 0;
}
public class B: A
{
    public int x = 1;
}
```

Клас B успадковує клас A, отже – і поле `x`. Однак у класі B оголошене нове поле з ідентичним іменем. У цьому випадку компілятор не є упевненим, що він розуміє логіку програміста, отож видасть повідомлення щодо своїх сумнівів. Проте код буде все ж скомпільовано.

Надання новим членам похідного класу імен, вже використаних у базовому, – потенційна небезпека помилок. Однак інколи така потреба виникає. У цьому випадку потрібно приховати компонент x класу A, оголосивши явно компонент x класу B новим за допомогою ключового слова new:

```
public class B: A
{
    public new int x = 1;
}
```

Для об'єкта класу B можемо отримати значення обох компонентів x:

```
B b = new B ();
int bx = b.x;      //bx набуде значення 1
int ax = ((A)b).x; //ax набуде значення 0
```

Приховування методів може стати необхідним у випадку конфлікту версій базового класу. Припустимо, що програміст *A* розробив базовий клас A, а програміст *B* на основі класу A – клас B, у який додав новий метод з назвою M. Через деякий час *A* дописує в класі A новий метод з назвою M та публікує нову версію. Після перекомпілювання програми результат виконання програми може бути не таким, як очікував *B*.

Оскільки компілятор C# відстежує такі ситуації, то він видасть відповідне повідомлення. Програміст *B* має два варіанти дій. Якщо він контролює усі класи, породжені від класу B, то краще перейменувати свій метод M. Якщо ж клас B опублікований для використання іншими користувачами, то до оголошення методу M необхідно додати модифікатор new.

Абстрактні методи

Нехай проектується деякий клас A. Вважають, що всі його дочірні класи повинні мати деякий метод M. Однак на рівні класу A недостатньо інформації для змістовного

означення цього методу. Якщо існує необхідність присутності методу M у класі A, то цей метод можна оголосити з модифікатором `abstract` без реалізації. У цьому випадку метод M називатиметься *абстрактним*. Якщо клас містить абстрактні методи, то він повинен також містити модифікатор `abstract`. Наприклад:

```
abstract public class A
{
    abstract public void M();
}
```

Зауважимо також:

– неможливо утворити об'єкт абстрактного класу;

– неможливо оголосити конструктор абстрактним методом;

– абстрактні класи використовуються для породження інших класів;

– дочірні класи (якщо вони не є також абстрактними) зобов'язані містити реалізацію усіх абстрактних методів, успадкованих від базового класу.

Віртуальні методи

Продовжимо розгляд класу `Points`. Об'єкт цього класу містить індексовану множину точок. Ці точки можна розглядати як вузли ламаної лінії. Тоді можна ввести поняття довжини та оголосити метод `GetLength`, який повертає цю довжину.

Похідний від `Points` клас `Triangle` успадкує цей метод `GetLength`. Однак для трикутника довжина – це периметр. А успадкований `GetLength` не враховує в довжині пряму, що з'єднує останню точку з першою.

Якщо може виникнути потреба у зміні реалізації деякого методу в дочірніх класах, його потрібно оголошувати *віртуальним*. З цією метою використовують модифікатор `virtual`.

Додамо метод GetLength до класу Points:

```
public class Points : IEnumerable
{
  public double GetDistance(Point p1, Point p2)
  {
    return Math.Sqrt((p1.x – p2.x) * (p1.x – p2.x) +
                     (p1.y – p2.y) * (p1.y – p2.y));
  }
  public virtual double GetLength()
  {
    double length = 0;
    for (int i = 0; i < count – 1; i++)
      length += GetDistance (Points[i],Points[i+1]);
    return length;
  }
}
```

Метод GetLength тут оголошено віртуальним, оскільки поняття довжини може змінюватися в класах-нащадках. А от відстань між точками навряд чи потребуватиме переозначення. Тому метод GetDistance не описано як віртуальний.

Перекривання віртуальних методів

Ми вже розглядали перекривання віртуальних методів у класі Point, де перекривається успадкований від object віртуальний метод ToString:

```
public override string ToString()
```

У свою чергу в класі Points з таким самим синтаксисом перекривається успадкований уже від Point віртуальний метод ToString. Щоб перекрити віртуальний метод, означений у базовому класі, необхідно у похідному класі повторити оголошення методу, але модифікатор virtual замінити на модифікатор override. Очевидно, що потрібно також написати нову реалізацію методу. Наприклад:

```
public class Triangle: Points
{
    public override double GetLength()
    {
       return          base.GetLength()          +
GetDistance(points[Count-1], points[0]);
    }
}
```

C# має модифікатор `sealed`, який використовують у парі з `override` і який дає вказівку заборонити перекривання методу в дочірніх класах – *запечатує.*

Поліморфізм

Механізм віртуальних функцій реалізує концепцію поліморфізму об'єктно-орієнтованого програмування.

Розглянемо такі два класи:

```
public class A
{
    public string Method()
    {
        return "A.Method";
    }
    public virtual string VirtualMethod()
    {
        return "A.VirtualMethod";
    }
}
public class B: A
{
    public new string Method()
    {
        return "B.Method";
    }
    public override string VirtualMethod()
    {
```

```
        return "B.VirtualMethod";
    }
}
```

Клас B приховує успадкований метод `Method`, оголосивши новий з ідентичним іменем. А віртуальний метод `VirtualMethod` клас А перекриває. Оголосимо змінні:

```
A a = new A ();
B b = new B ();
A x = b;
```

Змінні `a` та `b` містять адреси утворених екземплярів відповідно, класів А та В. Змінна `x` містить адресу того ж об'єкта, що й `b`, але має тип класу А. Наступний код демонструє особливості віртуальних функцій:

```
string s;
s = a.Method(); //"A.Method"
s = b.Method(); //"B.Method"
s = x.Method(); //"A.Method"
s = a.VirtualMethod(); //"A.VirtualMethod"
s = b.VirtualMethod(); //"B.VirtualMethod"
s = x.VirtualMethod(); //"B.VirtualMethod"
s = ((A)b).VirtualMethod(); //"B.VirtualMethod"
```

Якщо метод не є віртуальним, компілятор використовує той тип, який змінна мала при оголошенні. У нашому випадку `x` має тип А. Отож код `x.Method()` викличе метод класу А, хоча реально `x` є посиланням на об'єкт класу В. Якщо метод є віртуальним, компілятор згенерує код, який під час виконання перевірятиме, куди насправді вказує посилання, і використовуватиме методи відповідного класу. Хоча `x` має тип А, викликається метод `VirtualMethod` класу В. Окрім того, навіть явне приведення типу до А ситуацію не змінює.

Використаємо описану властивість поліморфізму для означення функції, яка повертає довжину об'єкта класу `Points` або `Triangle`.

```
public double PointsLength(Points points)
{
    return points.GetLength();
}
```

Оскільки метод `GetLength` є віртуальним у класах `Points` та `Triangle`, то функція `PointsLength` повертатиме коректні значення довжини для об'єктів різних типів:

```
Points Ps = new Points(3);
Ps[0] .Set (0, 0);
Ps[l] .Set (0, 3);
Ps[2] .Set (4, 0);
Triangle T = new Triangle(Ps[0], Ps[1], Ps [2]);
double pl = PointsLength(Ps);    // pl = 8
double tl = PointsLength(T);     // tl = 12
```

Приклади виконання завдань

Завдання №1

Розробити класи, які б реалізовували малювання геометричних фігур (точки, квадрата) у заданих координатах текстового режиму консолі за допомогою певного символу, наприклад «*». Побудувати ієрархію класів, а базовий клас при цьому об'явити абстрактним, реалізувати механізми наслідування та поліморфізму. У якості поліморфізму виконати перевантаження одного з методів базового класу. Побудувати UML-діаграму класів та діаграму послідовностей засобами Microsoft Visual Studio.

```
using System;
namespace Lab4_2
{
```

```
// клас містить властивості олівця колір
class Pen
{
  public ConsoleColor Color;
  public char Symbol;
  public Pen() // конструктор
  {
    Color = ConsoleColor.White; // білий колір
    Symbol = '*'; // за умовчанням малює зірочками
  }
}
// об’єкт цього класу створити не можна
abstract class Figure
{
  public int x;
  public int y;
  public Pen pen = new Pen();
  public abstract void Draw();
}
class Point : Figure // клас точки
{
  // перевантаження методу
  public override void Draw()
  {
    Console.ForegroundColor = pen.Color;
    Console.SetCursorPosition(x, y);
    Console.Write(pen.Symbol);
  }
}
class Square : Figure // клас квадрата
{
  public int size = 10; // розмір у символах
  // перевантаження методу
  public override void Draw()
  {
    int left = x - size / 2;
    int top = y - size / 2;
    for (int i = 0; i <= size; i++)
```

```
      {
        Console.ForegroundColor = pen.Color;
        Console.SetCursorPosition(left + i, top);
        Console.Write(pen.Symbol);
        Console.SetCursorPosition(left+i, top+size);
        Console.Write(pen.Symbol);
        Console.SetCursorPosition(left, top + i);
        Console.Write(pen.Symbol);
        Console.SetCursorPosition(left+size, top+i);
        Console.Write(pen.Symbol);
      }
    }
  }
  class Program
  {
    static void Main(string[] args)
    {
      Figure fig1 = new Point();
      Fig1.x = 10;
      fig1.y = 10;
      fig1.Draw();
      Figure fig2 = new Square();
      fig2.x = 20;
      fig2.y = 10;
      fig2.Draw(); // приклад поліморфізму
      Console.ReadKey();
    }
  }
}
```

Для створення діаграми класів спершу до поточного рішення потрібно додати новий проект, обравши пункт меню *Файл* | *Добавить* | *Создать проект…* (рис. 2.1).

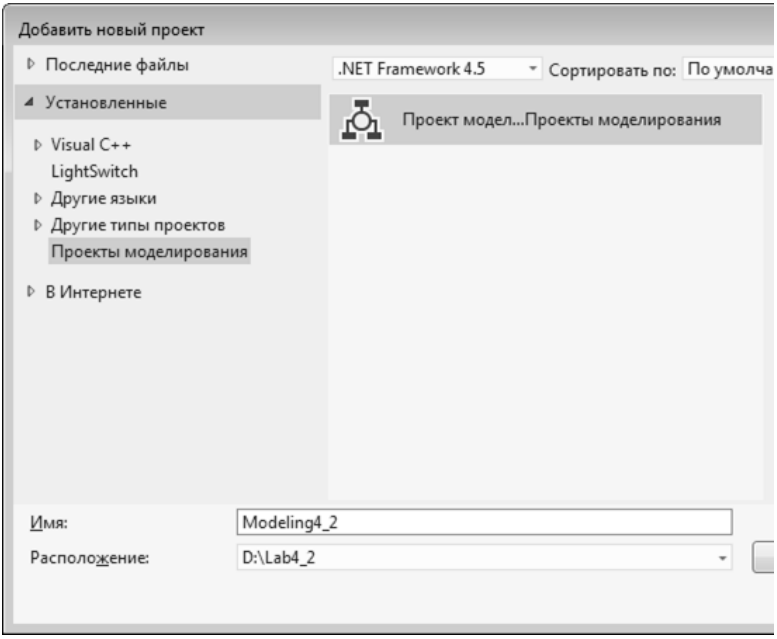


Рис. 2.1. Створення проекту моделювання

Наступним кроком потрібно додати діаграму класів UML до проекту, створеного раніше (рис. 2.2).

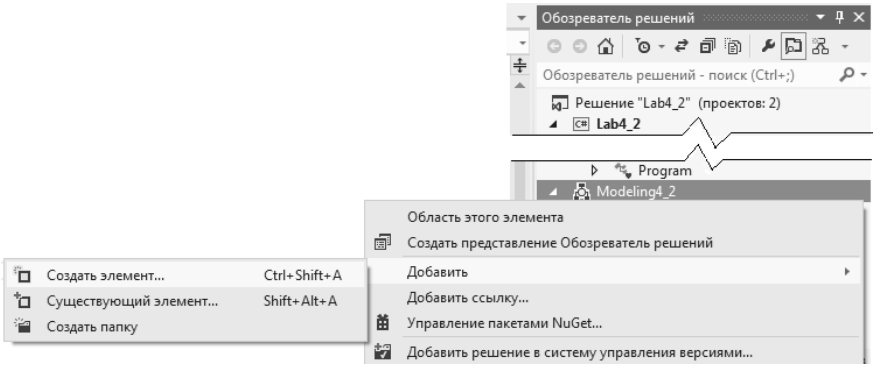


Рис. 2.2. Додавання нової діаграми до проекту моделювання

Після перенесення на робочу область всіх класів програмного коду одержимо діаграму, зображену на рисунку 2.3.

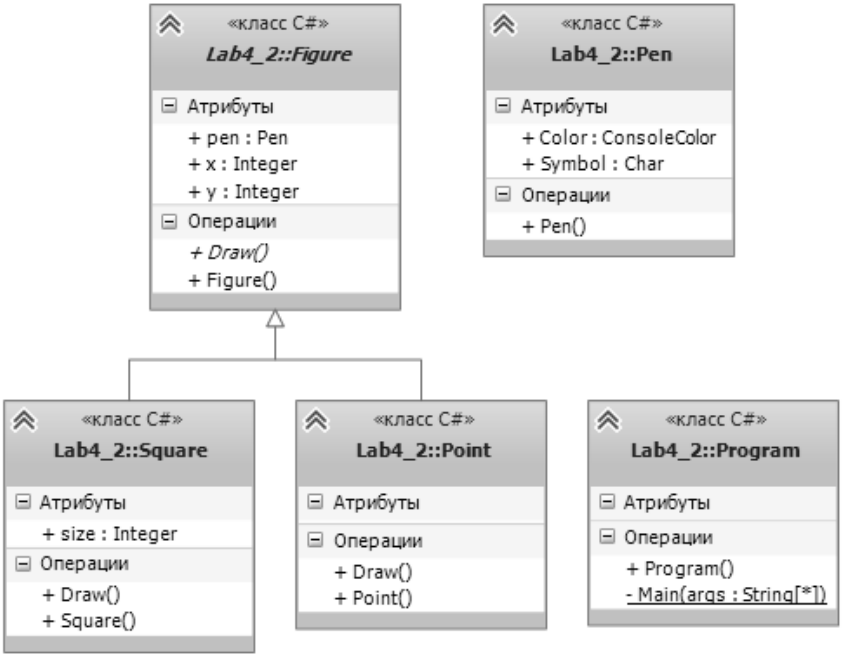


Рис. 2.3. Діаграма класів UML

Для відображення часу життя об’єктів та послідовності викликів методів використовується діаграма послідовностей UML [1, 5, 10, 22]. Для її побудови потрібно обрати фрагмент коду (наприклад, точку входу у програмний додаток) та скористатися контекстним меню редактора коду (рис. 2.4, 2.5).

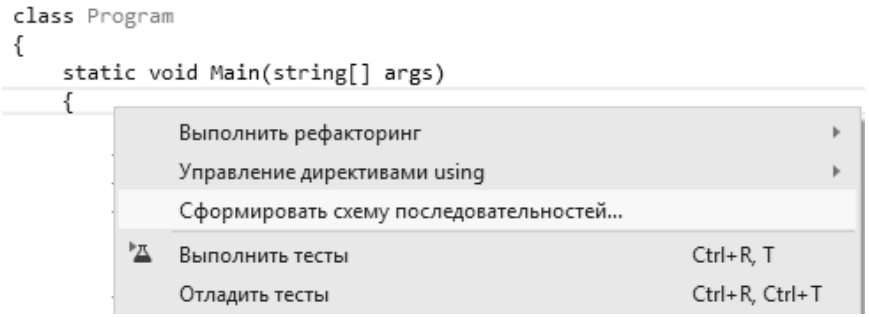


Рис. 2.4. Формування діаграми послідовностей UML

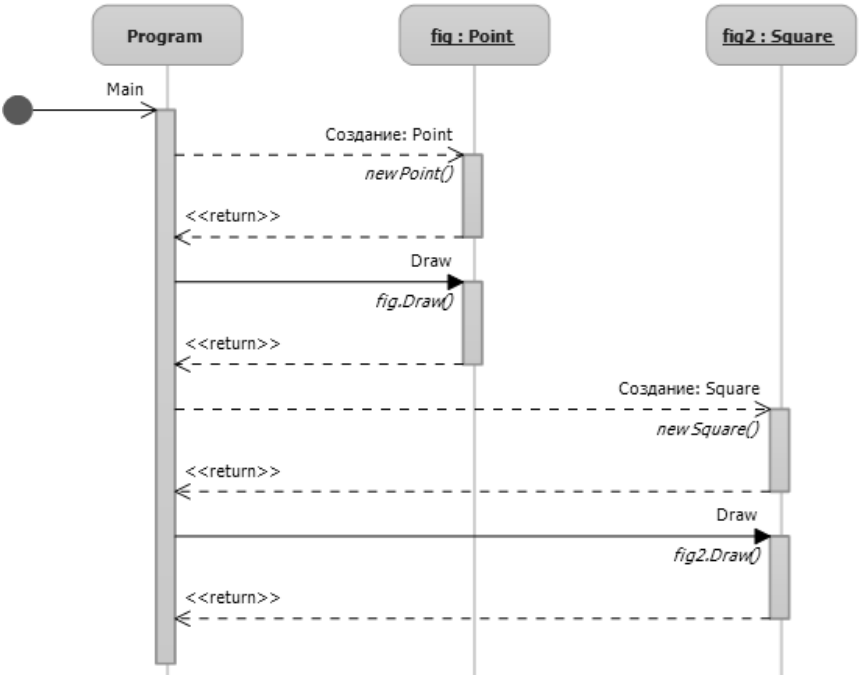


Рис. 2.5. Діаграми послідовностей UML для методу `Main`

У випадку багатопоточних додатків та складній програмній архітектурі діаграми послідовностей UML можуть бути дуже корисними при документуванні та презентації розроблених програмістом бібліотек [17].

Завдання №2

Реалізувати клас, який інкапсулює властивості матриці розміром 3×3 елемента цілого типу. Реалізувати методи випадкового заповнення, обнулення, транспонування та виведення на консоль. Створити нащадка від базового класу, додавши можливість отримання максимального та мінімального значення в матриці. Дослідити послідовність виклику конструкторів при створенні ієрархії класів.

```
using System;

namespace Lab4_3
{
  class Matrix3i // базовий клас матриці
  {
    public int[,] m = new int[3, 3];
    // формування випадкової матриці, кожний елемент
    // якої знаходиться у межах від min до max
    public void Random(int min, int max)
    {
      Random rnd = new Random();
      for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
          m[i, j] = rnd.Next(min, max);
     }
     public void Print() // друк на консоль
     {
       Console.WriteLine("\nm = ");
       Console.WriteLine("{0}\t{1}\t{2}",
                         m[0, 0], m[0, 1], m[0, 2]);
       Console.WriteLine("{0}\t{1}\t{2}",
                         m[1, 0], m[1, 1], m[1, 2]);
      Console.WriteLine("{0}\t{1}\t{2}",
                        m[2, 0], m[2, 1], m[2, 2]);
    }
    public void Transpose() // транспонування матриці
    {
      int tmp;
```

```
    tmp = m[0, 1]; m[0, 1] = m[1, 0]; m[1, 0] = tmp;
    tmp = m[0, 2]; m[0, 2] = m[2, 0]; m[2, 0] = tmp;
    tmp = m[1, 2]; m[1, 2] = m[2, 1]; m[2, 1] = tmp;
  }
  public void SetZero() // заповнення нулями
  {
    for (int i = 0; i < 3; i++)
      for (int j = 0; j < 3; j++)
        m[i, j] = 0;
  }
  public Matrix3i() // конструктор
  {
    Console.WriteLine("Create Matrix3i");
  }
}

class Matrix3iEx : Matrix3i // похідний клас
{
  // властивість тільки для читання
  public int MaxValue
  {
    get
    {
      int max = m[0, 0];
      for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
          if (m[i, j] > max)
            max = m[i, j];
      return max;
    }
  }

  // властивість тільки для читання
  public int MinValue
  {
    get
    {
      int min = m[0, 0];
```

```
        for (int i = 0; i < 3; i++)
          for (int j = 0; j < 3; j++)
            if (m[i, j] < min)
              min = m[i, j];
        return min;
       }
    }
    public Matrix3iEx() // контсруктор
    {
      Console.WriteLine("Create Matrix3iEx");
    }
  }
  class Program
  {
    static void Main(string[] args)
    {
      Matrix3iEx m = new Matrix3iEx();
      m.Print();
      m.Random(-10, 10);
      m.Print();
      m.Transpose();
      m.Print();
      Console.WriteLine();
      Console.WriteLine("Max = {0}", m.MaxValue);
      Console.WriteLine("Min = {0}", m.MinValue);
      Console.ReadKey();
    }
  }
}
```

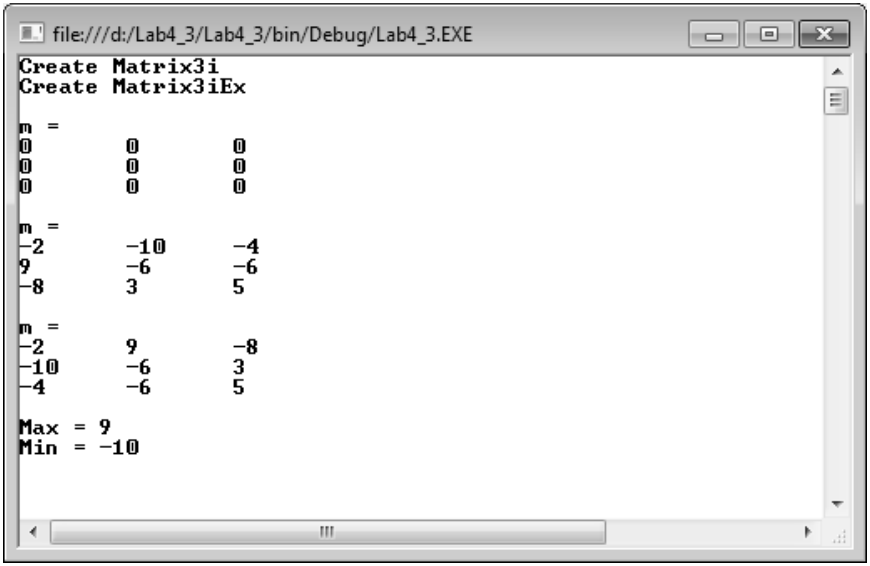


Рис. 2.6. Демонстрація роботи програми

Завдання для самостійного виконання

Відповідно до варіанту та обраного рівня складності виконати такі завдання.

Рівень завдань	Завдання
Початковий	Відкомпілювати та протестувати аудиторні завдання
Базовий	До кожного з класів, які наведено нижче, додати по одному полю та методу, побудувати ієрархію класів та реалізувати механізм наслідування
Високий	До кожного з класів, які наведено нижче, додати по три поля, два методи та по одному конструктору з використання модифікаторів доступу `private, public, protected`. Побудувати ієрархію класів, а базовий клас при цьому об’явити абстрактним, реалізувати

Рівень завдань	Завдання
	механізми наслідування та поліморфізму. У якості поліморфізму виконати перевантаження одного з методів базового класу. Побудувати UML-діаграму класів та діаграму послідовностей засобами Microsoft Visual Studio

№ варіанта	Завдання
1	Студент, викладач, персона, завідувач кафедри
2	Службовець, персона, робочий, інженер
3	Робітник, кадри, інженер, адміністрація
4	Деталь, механізм, виріб, вузол
5	Організація, страхова компанія, нафтогазова компанія, завод
6	Журнал, книга, друковане видання, підручник
7	Тест, екзамен, випускний екзамен, випробування
8	Місце, область, місто, мегаполіс
9	Іграшка, продукт, товар, молочний продукт
10	Квитанція, накладна, документ, рахунок
11	Речовина, мідь, соляна кислота, хлорид натрію, уран
12	Ртуть, вода, рідина, етиловий спирт
13	Резистор, світлодіод, елемент, трансформатор
14	Джойстик, мишка, пристрій введення, клавіатура
15	Лінія, точка, геометрична фігура, трикутник
16	Ноутбук, комп’ютер, планшет, нетбук
17	Гітара, рояль, сопілка, музичний інструмент
18	Ксерокс, принтер, периферійний пристрій, сканер
19	Маршрутизатор, комутатор, концентратор, мережевий пристрій
20	Автомобіль, поїзд, транспортний засіб, експрес

№ варіанта	Завдання
21	Двигун, двигун внутрішнього згоряння, дизель, реактивний двигун
22	Республіка, монархія, королівство, держава
23	Ссавець, парнокопитне, птиця, тварина
24	Товар, велосипед, гірський велосипед, самокат
25	Лев, дельфін, птиця, синиця, тварина
26	Тест, екзамен, випускний екзамен, випробування
27	Місце, область, місто, мегаполіс
28	Іграшка, продукт, товар, молочний продукт
29	Журнал, книга, друковане видання, підручник
30	Робітник, кадри, інженер, адміністрація

Контрольні питання

1. Що таке клас у контексті мови C#?
2. Які основні принципи об’єктно-орієнтованого програмування вам відомі?
3. У чому полягає сутність абстрактних методів?
4. Коротко охарактеризувати модифікатори доступу `private, public, protected.`
5. У чому сутність інкапсуляції та поліморфізму?
6. Чим відрізняються змінні за посиланням від змінних за значенням?
7. Які існують правила для конструкторів класів?
8. Яким чином програміст повинен видаляти об’єкти із пам’яті після їх використання?
9. Як відкомпілювати файл програми на мові C# у командному рядку?
10. Для чого призначена уніфікована мова моделювання UML?

ТЕМА 3. ПРОГРАМУВАННЯ МЕРЕЖНИХ ДОДАТКІВ З ВИКОРИСТАННЯМ ПРОТОКОЛІВ TCP ТА UDP

Мета: розробити клієнтську та серверну частину мережної інформаційної системи на базі об'єктно-орієнтованого підходу.

Мережні сокети та порти

Мережний сокет (Network Socket) – це кінцева точка двохсторонньої комунікаційної взаємодії комп'ютерів в мережі.

Інтерфейсний сокет (Socket API) – програмний інтерфейс, який надає операційна система та дозволяє програмним додаткам керувати та використовувати мережні сокети.

Адреса сокету (Socket Address) – поєднання IP-адреси та номера порту. Базуючись на цій адресі, інтернет-сокети доставляють відповідному програмному процесу чи потоку пакети даних, що надходять з мережі.

Слід розрізняти клієнтські і серверні сокети. Клієнтські сокети можна порівняти з кінцевими апаратами телефонної мережі, а серверні – з комутаторами. Клієнтський додаток, наприклад браузер, використовує тільки клієнтські сокети, а серверний, наприклад веб-сервер, якому браузер посилає запити, – як клієнтські, так і серверні сокети.

Інтерфейс сокетів вперше з'явився в BSD Unix. Програмний інтерфейс сокетів описаний в стандарті POSIX.1 і в тій чи іншій мірі підтримується всіма сучасними операційними системами. Кожен процес може створити сокет для прослуховування або серверний сокет і прив'язати його до якого-небудь порту операційної системи (рис. 3.1). Процес, що прослуховує канал, зазвичай знаходиться в циклі очікування, тобто прокидається при появі нового з'єднання. При цьому зберігається можливість перевірити наявність з'єднань на даний момент, встановити тайм-аут для операції.

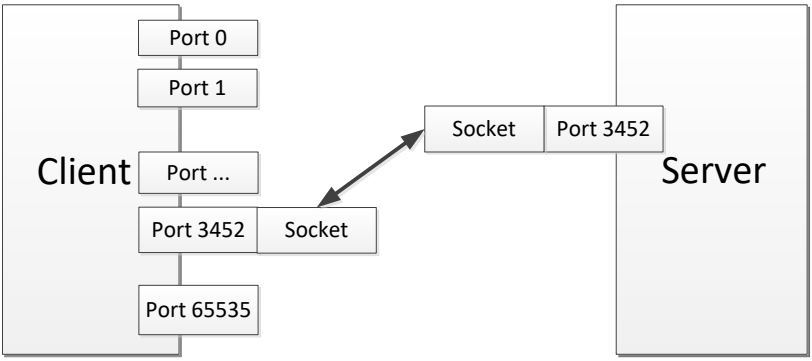


Рис. 3.1. Загальна схема роботи сокетів

Кожен сокет має свою адресу. Операційні системи сімейства UNIX можуть підтримувати багато типів адрес, але обов'язковими є INET-адреса і UNIX-адреса. Якщо прив'язати сокет за UNIX-адресою, то буде створений спеціальний файл або файл сокету за заданим шляхом, через який зможуть «спілкуватися» будь-які локальні процеси шляхом читання/запису з нього. Сокети типу INET доступні з мережі і вимагають виділення номера порту. Зазвичай клієнт явно під'єднується до слухача, після чого будь-яке читання або запис через його файловий дескриптор буде передаватися між ним і сервером. Загальновідомі сокети представляють собою зручний механізм апріорного прив'язування адреси сокету до якого завгодно стандартного сервісу. Наприклад, процес-сервер для програми Telnet жорстко зв'язаний з конкретним сокетом (рис. 3.2). Адреса сокету може бути зарезервована для доступу до процедури перегляду, яка могла б вказувати сокет, крізь який можна було б отримати новоутворені послуги.

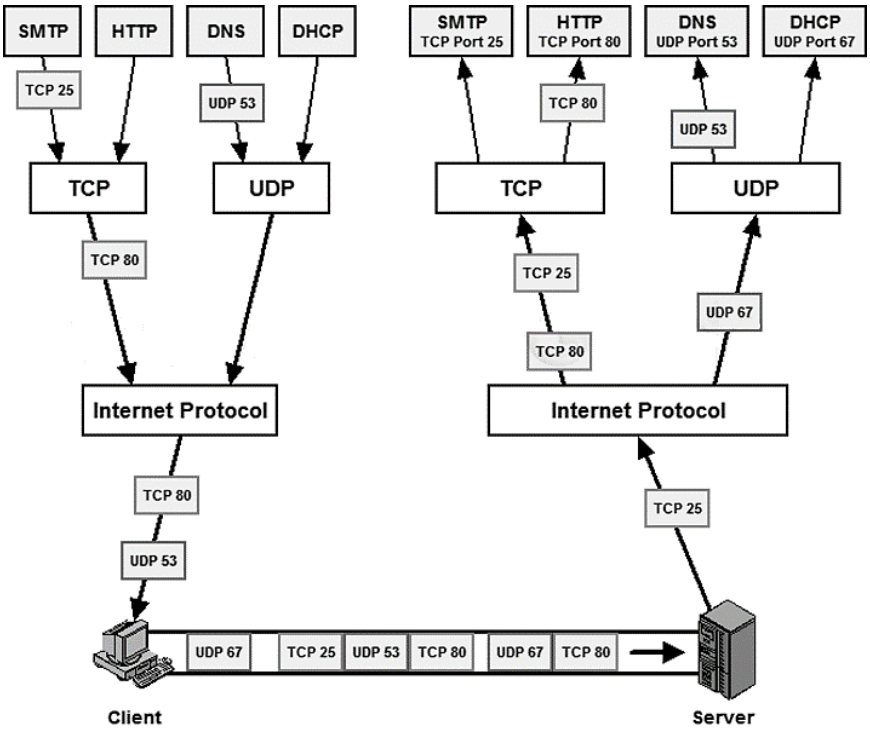


Рис. 3.2. Пояснення схеми роботи портів

У протоколах TCP/UDP порт – це системний ресурс, який має номер та виділяється програмі для зв'язку з додатками, що виконуються на інших мережних хостах (табл. 3.1). Порт може бути зайнятий тільки однією програмою і в цей час не може використовуватися іншою. Усі програми для зв'язку між собою за допомогою мережі використовують порти (до 65536 портів).

Таблиця 3.1

Деякі загальновідомі порти

Порт	TCP	UDP	Опис
0	√	√	Зарезервовано
20	√		Протокол FTP (дані)
21	√		Протокол FTP (команди)
22	√	√	Протокол SSH
23	√	√	Протокол Telnet

Порт	TCP	UDP	Опис
53		√	Протокол DNS
80	√	√	Протокол HTTP
443	√		Протокол HTTPS
6969	√		Клієнт BitTorrent
33434	√	√	Службова утиліта Traceroute

Стек протоколів TCP/IP. Протоколи TCP та UDP

TCP/IP – це абревіатура терміну Transmission Control Protocol / Internet Protocol (Протокол керування передачею / міжмережевий протокол) [19]. Фактично TCP/IP не один протокол, а декілька (табл. 3.2).

Таблиця 3.2

Співвідношення моделі OSI та стеку протоколів TCP/IP

OSI Model	TCP/IP Model	
Application	Application	Telnet, SMTP, POP3, FTP, HTTP, SNMP, DNS, SSH, …
Presentation		
Session		
Transport	Transport	TCP, UDP
Network	Internet	IP, ICMP, ARP, DHCP
Data Link	Network Access	Ethernet, ADSL, …
Physical		

Саме тому його часто називають набором, або комплектом (стеком) протоколів, серед яких TCP і IP – два основні. Фактично TCP/IP представляє цей базовий набір протоколів, відповідальний за розбивання вихідного повідомлення на пакети (TCP), доставку пакетів на вузол адресата (IP) і збирання (відновлення) вихідного повідомлення з пакетів (TCP).

Прикладний рівень

Протоколи прикладного рівня TCP/IP визначають процедури стосовно організації взаємодії прикладних процесів (програм) різних мережних комп’ютерів і форми подання

інформації за такої взаємодії. За ознаками взаємодії прикладних процесів виокремлюють два типи прикладного програмного забезпечення: програма-клієнт та програма-сервер. Протоколи прикладного рівня зорієнтовано на конкретні прикладні завдання. Серед традиційних послуг, котрі забезпечують протоколи прикладного рівня із сімейства TCP/IP, сьогодні найпопулярнішими є електронна пошта – протоколи SMTP та POP3, передавання файлів – FTP та TFTP, емуляції віддаленого терміналу – TELNET тощо.

В інтернеті активно використовуються послуги, які базуються на технології WWW, яка ґрунтується на протоколі передавання гіпертексту HTTP. Сьогодні популярні послуги пакетної IP-телефонії на базі стандартів IETF, які стосуються спеціальних протоколів прикладного, транспортного і мережного рівнів, наприклад сигналізації SIP, передавання в режимі реального часу RTP та RTCP, резервування ресурсів RSVP, рекомендацій ITU H.323 тощо.

Транспортний рівень

Протоколи транспортного рівня TCP/IP надають транспортні послуги прикладним процесам. Основними протоколами транспортного рівня TCP/IP є протокол керування передаванням TCP (Transmission Control Protocol) і протокол користувальницьких дейтаграм UDP (User Datagram Protocol). Транспортні послуги цих протоколів суттєво відрізняються. Протокол UDP доставляє дейтаграми без установлення з'єднання. При цьому він не гарантує їхнього доставляння. Протокол TCP забезпечує надійне доставляння байтових потоків (сегментів) із попереднім встановленням транспортного дуплексного з'єднання (віртуального каналу) між модулями TCP мережних комп'ютерів. Для розв'язання транспортних завдань протоколи TCP та UDP у перебігу передавання даних формують і додають до даних свої заголовки обсягом 20 байт та 8 байт відповідно.

Кожен прикладний процес взаємодіє з модулем транспортного рівня TCP або UDP через окремий порт, що

дозволяє при взаємодії систем однозначно ідентифікувати прикладні процеси. Ці порти нумеруються, починаючи з нуля. При передаванні запиту прикладної програми клієнта до прикладної програми сервера транспортний модуль, формуючи дейтаграму чи сегмент, вказує номери портів програмних модулів прикладних протоколів сервера й клієнта. З цією метою в заголовку пакета протоколу транспортного рівня виділено два поля – «порт одержувача» і «порт відправника», обсягом по 2 байти. Номери портів TCP та UDP до прикладних протоколів сервера стандартизовано IETF. Для цього надано номери в діапазоні від 1 до 1023. Наприклад, програмний модуль TCP сервера взаємодіє з модулем протоколу HTTP через порт з номером 80. Взаємодія модуля TCP чи UDP клієнта з будь-яким модулем прикладного протоколу відбувається через порт, якому надається вільний номер, за значенням більший ніж 1023.

Мережний рівень

Протоколи мережного рівня TCP/IP забезпечують взаємодію поміж мережами різної архітектури тощо. Основним протоколом мережного рівня технології TCP/IP є міжмережний протокол IP та його допоміжні протоколи: адресний протокол ARP; реверсний адресний протокол RARP (Reverse ARP); протокол діагностичних повідомлень ICMP (Internet Control Message Protocol), який надсилає повідомлення вузлам мережі про помилки на маршруті, які виникають при передаванні пакетів тощо.

Головне завдання міжмережного протоколу IP – це маршрутизація пакетів даних поміж різнотипними комп'ютерними мережами. Для розв'язання цього завдання протокол IP підтримує IP-адресацію мереж та вузлів, використовує таблицю маршрутизації пакетів, виконує, за необхідності, фрагментацію та дефрагментацію цих пакетів.

Функціонування мережного рівня також забезпечує низка протоколів динамічної маршрутизації RIP, OSPF, які динамічно формують маршрути таблиці маршрутизації за векторними

алгоритмами VDA (Vector Distance Algorithm) і алгоритмами стану зв'язку LSA (Link State Algorithm) відповідно, протоколів політики зовнішньої маршрутизації EGP (Exterior Gateway Protocol), BGP (Border Gateway Protocol) тощо.

Засоби мережевого рівня забезпечують доставку даних між пристроями в складових мережі, а саме комп'ютерами, маршрутизаторами і тощо. Однак не слід забувати, що на одному вузлі може функціонувати паралельно декілька програм, яким потрібен доступ до мережі. Отже, дані всередині комп'ютерної системи повинні розподілятися між програмами. Тому при передачі даних мережею недостатньо просто адресувати конкретний вузол. Необхідно також ідентифікувати програму-одержувача, що неможливо здійснити засобами мережевого рівня.

Іншою серйозною проблемою протоколів мережевого рівня є відсутність засобів, що дозволяють передавати великі масиви даних. Коли вихідні дані перевищують максимально допустимий розмір пакета мережного рівня, то ці дані повинні бути розбиті на порції, кожна з яких передається в мережу окремим пакетом. Проте кожен пакет мережевого рівня передається мережею як єдиний, незалежний від інших блок даних. Якщо будь-які пакети «загубилися», то модуль мережевого протоколу на приймаючій стороні не зможе виявити втрату, і, отже – виявити порушення цілісності загального масиву даних. Тому кошти транспортного рівня забезпечують відсутність втрат інформації. Такий режим передачі даних отримав назву гарантованої доставки.

Таким чином, засоби транспортного рівня представляють собою функціональну надбудову над мережним рівнем і вирішують дві основні задачі:

1) забезпечення доставки даних між програмами, що функціонують на різних вузлах мережі;

2) забезпечення гарантованої доставки масивів даних довільного розміру.

У даний час в Інтернет використовуються два транспортних протоколу – UDP, що забезпечує негарантовану

доставку даних між програмами, і TCP, що забезпечує гарантовану доставку з встановленням віртуального з'єднання.

Для ідентифікації програм протоколи транспортного рівня в мережі Інтернет (TCP і UDP) використовують унікальні числові значення, так звані порти. Номери портів призначаються програмами відповідно до її функціонального призначення на основі певних стандартів. Для кожного протоколу існують стандартні списки відповідності номерів портів і програм. Так, наприклад, програмне забезпечення WWW, що працює через транспортний протокол TCP, використовує TCP-порт 80, модулі протоколу FTP – TCP-порт 21, а служба DNS взаємодіє з транспортними протоколами TCP і UDP через TCP-порт 53 і UDP-порт 53 відповідно.

Таким чином, протокол мережевого рівня IP і транспортні протоколи TCP і UDP реалізують дворівневу схему адресації: номери TCP-і UDP-портів дозволяють однозначно ідентифікувати програму в рамках вузла, а сам вузол однозначно визначається IP-адресою. Отже, комбінація IP-адреси і номера порту дозволяє однозначно ідентифікувати програму в мережі Інтернет. Така комбінована адреса називається сокетом або socket.

Протокол UDP (User Datagram Protocol) – протокол транспортного рівня, який входить у стек протоколів TCP/IP, що забезпечує негарантовану доставку даних без встановлення віртуального з'єднання (рис. 3.3).

Оскільки на протокол не покладається завдань щодо забезпечення гарантованої доставки, а лише потрібно забезпечувати зв'язок між різними програмами, то структура заголовка дейтаграми UDP, таку назву має пакет протоколу, виглядає досить просто – вона включає в себе всього чотири поля. Перші два поля містять номери UDP-портів програми-відправника та програми-одержувача. Два інших поля в структурі заголовка дейтаграми призначені для управління обробкою – це загальна довжина дейтаграми і контрольна сума заголовка.

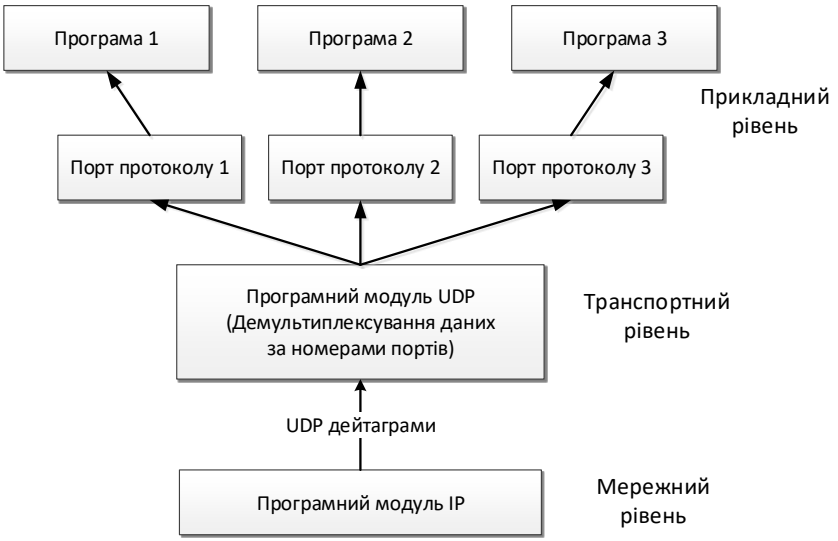


Рис. 3.3. Робота додатка за UDP протоколом

Протокол TCP (Transmission Control Protocol) є транспортним протоколом стека протоколів TCP/IP, що забезпечує гарантовану доставку даних з встановленням віртуального з'єднання (рис. 3.4).

Протокол надає програмам, що використовують його, можливість передачі безперервного потоку даних. Дані, що підлягають відправці в мережу, розбиваються на порції, кожна з яких забезпечується службовою інформацією, тобто формуються пакети даних. У термінології TCP пакет називається сегментом.

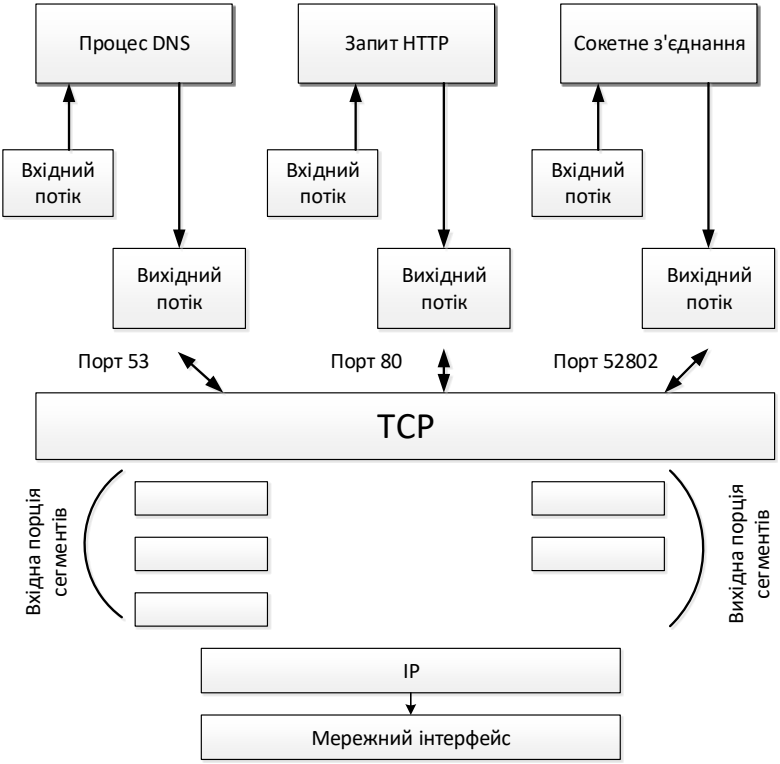


Рис. 3.4. Робота додатка за TCP протоколом

Відповідно до функціонального призначення протоколу структура TCP-сегмента передбачає наявність таких інформаційних полів:

1) номер порту-відправника і номер порту-одержувача – номери портів, що ідентифікують програми, між якими здійснюється взаємодія;

2) поля, призначені для забезпечення гарантованої доставки: розмір вікна, номер послідовності і номер підтвердження;

3) керуючі прапори – спеціальні бітові поля, що управляють протоколом.

Для забезпечення гарантованої доставки протокол TCP використовує механізм відправки підтвердження. З метою

зниження завантаження мережі протокол TCP допускає відправлення одного підтвердження відразу для декількох отриманих сегментів. Обсяг даних, які можуть бути передані в мережу відправником до отримання підтвердження, визначається спеціальним параметром протоколу TCP – розміром вікна. Розмір вікна узгоджується при встановленні з'єднання між відправником та одержувачем і може автоматично змінюватися програмними модулями протоколу TCP в залежності від стану каналу зв'язку. Якщо в процесі передачі даних втрати відбуваються досить часто, то розмір вікна зменшується, і навпаки – вікно може мати великий розмір, якщо висока надійність каналу даних.

Для того, щоб дані могли бути правильно зібрані одержувачем у потрібному порядку, в заголовку TCP-сегмента присутня інформація, яка визначає положення вкладених даних у загальному потоці. Відправляючи підтвердження, одержувач вказує положення даних, які він очікує отримати в наступному сегменті, тим самим побічно повідомляючи відправнику, який фрагмент загального потоку був успішно прийнятий. Відповідні поля заголовка TCP-сегмента отримали назву – номер послідовності і номер підтвердження.

Таблиця 3.3

Порівняння протоколів TCP та UDP

TCP	UDP
Надійний протокол	Ненадійний протокол
Встановлення віртуального з'єднання	Без встановлення віртуального з'єднання
Повторна передача сегментів та керування потоком	Без повторної передачі повідомлень
Використовується впорядкування сегментів	Випадковий порядок отримання сегментів
Для контролю слугують сегменти підтвердження	Підтвердження передачі відсутнє

Перед початком передачі потоку даних абоненти повинні узгодити параметри передачі: розмір вікна та початкові номери послідовностей, щодо яких буде відраховуватися положення переданих у сегментах даних усередині загального потоку. Очевидно, що таке узгодження передбачає обмін спеціальними сегментами і виділення ресурсів, зокрема, блоків пам'яті, необхідних для прийому та обробки даних і підтверджень. Відповідна послідовність дій називається встановленням віртуального з'єднання.

Розглянемо схему створення TCP-з'єднання (рис. 3.5).

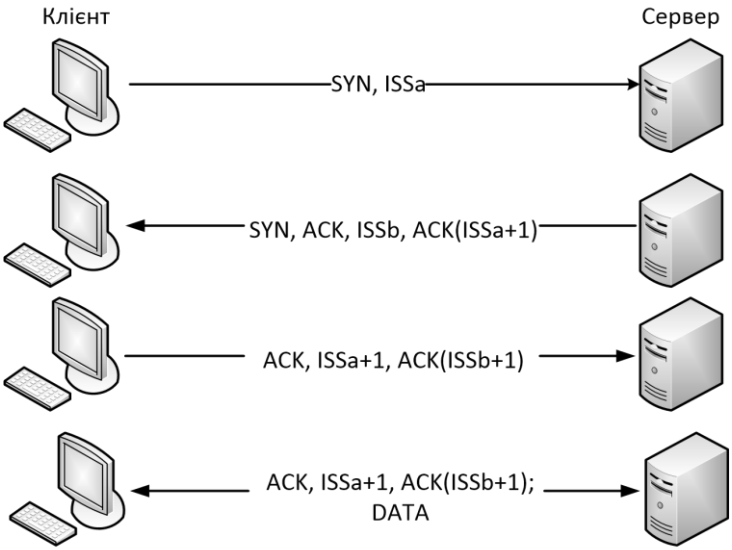


Рис. 3.5. Схема створення сокетного TCP з'єднання

Припустимо, що клієнту A необхідно створити TCP-з'єднання з сервером B. Тоді A посилає на B таке повідомлення: SYN, ISSa. Це означає, що в повідомленні, яке передає A, встановлений біт SYN (synchronize sequence number), а у поле Sequence Number встановлено початкове 32-бітне значення ISSa (Initial Sequence Number). Далі B відповідає: SYN, ACK, ISSb, ACK(ISSa+1).

У відповідь на отриманий від А запит В відповідає повідомленням, в якому встановлені біти SYN та ACK; у поле Sequence Number сервером В встановлюється своє початкове значення лічильника – ISSb. Поле Acknowledgment Number містить значення ISSa, отримане у першому пакеті від клієнта А та збільшене на одиницю. А, завершуючи рукостискання (handshake), надсилає: ACK, ISSa+1, ACK(ISSb+1).

У цьому пакеті встановлений біт ACK. Поле Sequence Number містить ISSa + 1. Поле Acknowledgment Number містить значення ISSb + 1. Відсиланням цього пакета на В закінчується трьохступеневий handshake та TCP-з'єднання між А та В вважається встановленим. Тепер клієнт може посилати пакети з даними на В щойно створеним віртуальним TCP-каналом: ACK, ISSa+1, ACK(ISSb+1); DATA.

Щоб ідентифікувати окремі потоки даних, які підтримує протокол TCP, останній визначає ідентифікатори портів. Оскільки ідентифікатори портів обираються кожною програмою протоколу TCP незалежно, то вони не будуть унікальними. Щоб забезпечити унікальність адрес для кожної програми протоколу TCP, треба об'єднати ідентифікуючу цю програму Internet-адресу та ідентифікатор порту. У результаті отримуємо сокет, який буде унікальний у всіх локальних мережах, об'єднаних у єдине ціле.

З'єднання повністю визначається парою сокетів на своїх кінцях. Локальний сокет може брати участь у багатьох з'єднаннях з різноманітними чужими сокетами. З'єднання можна використовувати для передачі даних у обох напрямках, іншими словами, воно є повністю дуплексним.

Протокол TCP може довільним чином зв'язувати порти з процесами. Проте при будь-якій реалізації протоколу необхідно дотримуватися деяких основних концепцій. Мають бути присутні загальновідомі сокети, які протокол TCP асоціює виключно з відповідними їм процесами.

З'єднання задається командою OPEN, виконаною з локального порту, та має аргументом чужий сокет. У відповідь на такий запит програма протоколу TCP надає ім'я локального

з'єднання. За цим ім'ям користувач адресується до даного з'єднання при наступних викликах. Існує певна структура даних, що має назву «блок керування передачею» або Transmission Control Block (TCB), призначена для збереження описаної вище інформації. Можна реалізувати протокол таким чином, щоб локальне ім'я для з'єднання було б вказівником на структуру TCB останнього. Запит OPEN вказує також, чи здійснюється з'єднання активним способом, чи проходить пасивне очікування з'єднання ззовні.

Запит на пасивне відкриття з'єднання означає, що процес чекає отримання ззовні запитів на з'єднання замість того, щоб намагатися самому встановити його. Часто процес, що зробив запит на пасивне відкриття, буде приймати запити на з'єднання від будь-якого іншого процесу. У цьому випадку чужий сокет вказується як такий, що складається повністю з нулів, що означає невизначеність. Невизначені чужі сокети можуть бути присутні лише в командах пасивного відкриття. Сервісний процес, що бажає обслужити інші, невідомі йому процеси, міг би здійснити запит на пасивне відкриття з вказанням невизначеного сокету. У цьому випадку з'єднання може бути встановлене з будь-яким процесом, що запросив з'єднання з цим локальним сокетом. Така процедура буде корисною, якщо відомо, що обраний локальний сокет асоційований з певним сервісом.

Мережні засоби платформи .NET Framework

У таблиці 3.4 коротко подано перелік класів бібліотеки .NET, з використанням яких можна побудувати мережний додаток.

Таблиця 3.4

Основні класи для роботи з сокетами в .NET

Клас .NET	Опис
`Socket`	Забезпечує базову функціональність сокету для додатка
`TcpClient`	Побудований на класі Socket, щоб забезпечити TCP обслуговування на більш високому рівні. TcpClient надає

Клас .NET	Опис
	декілька методів для відправки та отримання даних мережею
`TcpListener`	Побудований на низькорівневому класі Socket. Його основне призначення – серверні додатки. Він очікує вхідні запити на з'єднання від клієнтів та повідомляє додатки про будь-які з'єднання
`UdpClient`	Призначений для реалізації UDP обслуговування
`SocketException`	Цей виняток генерується, коли у сокеті виникає помилка

Базовим класом при побудові мережних додатків є клас System.Net.Sockets.Socket, деякі властивості та методи якого представлено в таблицях 3.5, 3.6.

Таблиця 3.5

Опис властивостей класу `System.Net.Sockets.Socket`

Властивості	Короткий опис та призначення
`AddressFamily`	Сімейство адрес сокету (значення із перерахунку Socket.AddressFamily)
`Available`	Об'єм даних, які можна зчитати
`Blocking`	Чи знаходиться сокет в блокуючому режимі
`Connected`	Чи з'єднаний сокет з віддаленим хостом
`LocalEndPoint`	Локальна кінцева точка сокету
`ProtocolType`	Тип протоколу сокету
`RemoteEndPoint`	Віддалена кінцева точка сокету
`SocketType`	Тип сокету

Таблиця 3.6

Опис методів класу `System.Net.Sockets.Socket`

Методи	Короткий опис та призначення
`Accept()`	Створює новий сокет для обробки вхідного запиту на з'єднання
`Bind()`	Зв'язує сокет з локальною кінцевою точкою для очікування вхідних запитів на з'єднання
`Close()`	Закриває сокет
`Connect()`	Встановлює з'єднання з віддаленим хостом
`Listen()`	Переводить сокет у режим прослуховування
`Receive()`	Отримує дані від приєднаного сокету
`Poll()`	Визначає статус сокету
`Send()`	Відправляє дані з'єднаному сокету
`ShutDown()`	Забороняє операції відправки та отримання даних на сокеті

Приклади виконання завдань

Завдання №1

Побудувати клієнтські та серверні додатки з використання протоколу TCP та класу `Socket, TcpListener, TcpClient.` Виконати передачу повідомлення між створеними додатками.

Розв'язання

TCP-сервер на базі класу `Socket`

```
static void Main(string[] args)
{
  //Sockets.NET Server

  IPEndPoint ipEndPoint =
  new IPEndPoint(IPAddress.Parse("127.0.0.1"), 8000);
  Socket serverSock =
    new Socket(AddressFamily.InterNetwork,
               SocketType.Stream, ProtocolType.Tcp);
```

```
  serverSock.Bind(ipEndPoint);
  Console.WriteLine("Ожидание входящего
                     TCP подключения...");
  serverSock.Listen(10);

  Socket clientSock = serverSock.Accept();
  IPEndPoint remote =
    (IPEndPoint)clientSock.RemoteEndPoint;
  Console.WriteLine("Соединение установлено
                     по адресу {0}.
                     Входящее сообщение:",
                     remote.Address.ToString());
  byte[] bufRecieve = new byte[8];
  clientSock.Receive(bufRecieve);
  Console.WriteLine(Encoding.UTF8.GetString(
    bufRecieve));
  Console.WriteLine("Для завершения нажмите
                     любую клавишу...");
  Console.ReadKey();
  serverSock.Close();
  clientSock.Close();
}
```

TCP-клієнт на базі класу Socket

```
static void Main(string[] args)
{
  //NET.Socket Client
  Socket cliSock =
     new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
  IPAddress ipServ = IPAddress.Parse("127.0.0.1");
  IPEndPoint ipEndP = new IPEndPoint(ipServ, 8000);
  try
  {
    cliSock.Connect(ipEndP);
  }
  catch (SocketException ex)
  {
```

```
    Console.WriteLine(ex);
  }
  byte[] pack = Encoding.UTF8.GetBytes("Hello");
  cliSock.Send(pack);
  Console.WriteLine("Передача завершена.
    Для продолжения нажмите любую клавишу...");
  Console.ReadKey();
  cliSock.Close();
  }
```

TCP-сервер на базі класу `TcpListener`

```
static void Main(string[] args)
{
  // TCPListener .NET
  TcpListener tcpServer =
    new TcpListener(IPAddress.Parse("127.0.0.1"),
                    8000);
  tcpServer.Start();
  Console.WriteLine("Ожидание подключения...");
  TcpClient tcpClient = tcpServer.AcceptTcpClient();
  Console.WriteLine("Подключение установлено,
                     входное сообщение:");
  byte[] buffer = new byte[8];
  NetworkStream streamTcp = tcpClient.GetStream();
  streamTcp.Read(buffer, 0, buffer.Length);
  Console.WriteLine(Encoding.UTF8.GetString(buffer));
  streamTcp.Close();
  tcpClient.Close();
  tcpServer.Stop();
  Console.WriteLine("Нажмите любую клавишу
                     для завершения...");
  Console.ReadKey();
}
```

TCP-клієнт на базі класу `TcpClient`

```
static void Main(string[] args)
{
  // TCPClient .NET
```

```
  TcpClient clientTcp = new TcpClient();
  clientTcp.Connect(IPAddress.Parse("127.0.0.1"),
                    8000);
  NetworkStream streamTcp = clientTcp.GetStream();
  byte[] buffer = new byte[8];
  buffer = Encoding.UTF8.GetBytes("Hello");
  streamTcp.Write(buffer, 0, buffer.Length);
  streamTcp.Close();
  clientTcp.Close();
  Console.WriteLine("Передача выполнена. Нажмите
    любую клавишу для завершения...");
  Console.ReadKey();
}
```

Завдання №2

Побудувати клієнтський та серверний додаток з використання протоколу UDP.

Розв'язання

UDP-сервер

```
static void Main(string[] args)
{
  byte[] buffer = new byte[8];
  IPEndPoint ipEndpoint =
    new IPEndPoint(IPAddress.Any, 8000);
  Socket serverSock =
    new Socket(AddressFamily.InterNetwork,
               SocketType.Dgram, ProtocolType.Udp);
  serverSock.Bind(ipEndpoint);
  Console.WriteLine("Ожидание подключения
                     по UDP-протоколу...");
  serverSock.Receive(buffer);
  Console.WriteLine("Передача данных UDP,
                     сообщение:");
  Console.WriteLine(Encoding.UTF8.GetString(buffer));
  serverSock.Close();
  Console.WriteLine("Для завершения нажмите
                     любую клавишу...");
```

```
  Console.ReadKey();
}
```

UDP-клієнт

```
static void Main(string[] args)
{
  UdpClient clientUdp = new UdpClient();
  clientUdp.Connect(IPAddress.Parse("127.0.0.1"),
                    8000);
  byte[] buffer = new byte[8];
  buffer = Encoding.UTF8.GetBytes("Hello");
  clientUdp.Send(buffer, buffer.Length);
  clientUdp.Close();
  Console.WriteLine("Сообщение отправлено.
    Для завершения нажмите любую клавишу");
  Console.ReadKey();
}
```

Завдання для самостійного виконання

Відповідно до варіанта та обраного рівня складності виконати такі завдання.

№	Рівень завдань		
	Початковий	Базовий	Високий
1	Відкомпілювати та протестувати аудиторні завдання	Завдання №1	Завдання №2
2	–	Завдання №2	Завдання №3

Завдання №1

Розробити клієнтський та серверний додаток шляхом внесення незначних змін в програми аудиторного завдання.

№ варіанта	Завдання
1	Ввести своє ім’я та вік з консолі, відправити їх з клієнта на сервер
2	Відправити рядок текста з сервера на клієнт, на клієнті вивести довжину рядка в символах
3	Відправити масив із 10 цілих чисел з клієнта на сервер
4	Відправити масив із 5 чисел типу `double` з сервера на клієнт
5	Відправити два числа з клієнта на сервер та знайти їх добуток
6	Відправити три числа з сервера на клієнт та знайти їх середнє арифметичне значення
7	Відправити з сервера на клієнт повідомлення та вивести на клієнті його у зворотному порядку
8	Відправити своє прізвище, ім’я, по батькові від клієнта до сервера та вивести на консоль
9	Знайти максимальний елемент масиву, відправити масив та це значення з сервера на клієнт
10	Відправити свою IP-адресу на сервер та знайти суму октетів на ньому
11	Ввести своє ім’я та вік з консолі, відправити їх з сервера на клієнт
12	Відправити з сервера на клієнт повідомлення та вивести на клієнті його прописними літерами
13	Відправити масив із 8 беззнакових цілих чисел з клієнта на сервер
14	Відправити масив із 6 чисел типу `byte` з клієнта на сервер
15	Ввести своє прізвище та групу з консолі, відправити їх з клієнта на сервер
16	Відправити рядок тексту з клієнта на сервер, на сервері вивести довжину рядка в символах
17	Відправити масив із 5 символів з клієнта на сервер

№ варіанта	Завдання
18	Відправити масив із 10 чисел типу `double` з клієнта на сервер
19	Відправити два числа з клієнта на сервер та знайти остачу від їх частки
20	Відправити три числа з сервера на клієнт та знайти їх середнє геометричне значення
21	Відправити з сервера на клієнт повідомлення та вивести на клієнті його без пробілів
22	Відправити своє прізвище ім’я, по батькові від клієнта до сервера та вивести на консоль прізвище та ініціали
23	Відправити масив з сервера на клієнт та знайти мінімальний елемент масиву
24	Відправити свою IP-адресу на сервер та знайти на ньому побітову операцію АБО перших двох октетів
25	Ввести своє прізвище та факультет з консолі, відправити їх з сервера на клієнт
26	Відправити три числа з клієнта на сервер та знайти суму тільки додатних
27	Відправити два рядки тексту з сервера на клієнт, на клієнті вивести довжину рядків у символах
28	Відправити масив із 10 символів з клієнта на сервер, на сервері змінити їх регістр на протилежний
29	Відправити масив із 5 чисел типу `float` з клієнта на сервер
30	Відправити два числа з клієнта на сервер та знайти їх різницю

Завдання №2

Розробити клієнтський та серверний додаток, обґрунтувавши обраний протокол передачі даних TCP або UDP.

№ варіанта	Завдання
1	Відправити з клієнта на сервер своє ім'я, на сервері додати до імені своє прізвище та відправити результат на клієнт
2	Відправити на сервер два числа, знайти їх суму та повернути результат клієнту
3	Відправити на клієнт два числа, знайти їх добуток та повернути результат серверу
4	Відправити на сервер масив, знайти його максимальний елемент та повернути результат клієнту
5	Відправити на сервер число, сервер перевіряє, чи є воно більшим за введене на ньому значення, якщо так, то повертає на клієнт своє число, інакше – повертає клієнту його ж число
6	Відправити на сервер число, сервер перевіряє, чи є це число паліндромом. Якщо так, то відправляє на клієнт «Yes», якщо ні, то повертає «No»
7	Відправити з сервера на клієнт масив чисел, відсортувати їх за зростанням та повернути на сервер
8	Відправити на сервер повідомлення, знайти у ньому кількість літер «а» та відправити на клієнт результат
9	Відправити на клієнт повідомлення, знайти його довжину і відправити відповідну кількість нулів на сервер
10	Відправити повідомлення на сервер, перевірити, чи є в ньому цифри, якщо так, то відправити повідомлення «Yes» на клієнт, інакше – «No»

№ варіанта	Завдання
11	Відправити з клієнта на сервер своє прізвище, на сервері додати до прізвища своє ім’я та відправити результат на клієнт
12	Відправити на сервер три числа, знайти їх суму та повернути результат клієнту
13	Відправити на клієнт три числа, знайти добуток від’ємних чисел та повернути результат серверу
14	Відправити на сервер масив, знайти його мінімальний та максимальний елементи та повернути результати клієнту
15	Клієнт відправляє чотири числа на сервер, сервер додає до чисел їх середнє значення та повертає клієнту
16	Відправити на сервер число, сервер перевіряє, чи є це число простим. Якщо так, то відправляє на клієнт «Yes», якщо ні, то повертає «No»
17	Відправити з сервера на клієнт масив цілих чисел, який вводить користувач, відсортувати їх за спаданням та повернути на сервер
18	Відправити на сервер повідомлення, знайти у ньому кількість пробілів та відправити на клієнт результат
19	Відправити на клієнт повідомлення, знайти його довжину і відправити відповідну кількість символів «X» на сервер
20	Відправити повідомлення на сервер, перевірити, чи є в ньому пробіли, якщо так, то відправити повідомлення «Yes» на клієнт, інакше – «No»
21	Відправити на сервер число, розкласти його на прості множники, результат повернути клієнту у вигляді масиву
22	Відправити на сервер число, сервер перевіряє, чи є це число квадратом іншого цілого числа. Якщо

№ варіанта	Завдання
	так, то відправляє на клієнт «Yes», якщо ні, то повертає «No»
23	Відправити з сервера на клієнт масив чисел, замінити від'ємні значення нулем та повернути на сервер
24	Відправити на сервер повідомлення, знайти у ньому кількість голосних літер та відправити на клієнт результат
25	Відправити на клієнт повідомлення, знайти його довжину і відправити відповідну кількість випадкових чисел на сервер
26	Відправити повідомлення на сервер, перевірити, чи містить воно знаки пунктуації, якщо так, то відправити повідомлення «Yes» на клієнт, інакше – «No»
27	Відправити з клієнта на сервер рядок тексту, на сервері вилучити всі прописні літери, результат повернути клієнту
28	Відправити на сервер три числа, знайти їх середнє геометричне та повернути результат клієнту
29	Відправити на клієнт два повідомлення, якщо вони однакові, то серверу надіслати рядок тексту «Failed»
30	Відправити на сервер масив, знайти добуток його елементів та повернути результати клієнту

Завдання №3

Розробити клієнтський та серверний додаток на основі протоколу TCP. З'єднання між клієнтом та сервером підтримувати до того часу, поки сервер не отримає від клієнта повідомлення «Exit». Передбачити обробку помилок введення даних користувачем. Функції клієнта та сервера подано нижче.

№ варіанта	Завдання
1	Відправляти повідомлення на сервер, видаляти у ньому всі голосні літери та повертати отримане повідомлення на клієнт
2	Розробити просту програму-тестування. Сервер має список питань, на які можна дати відповіді «Yes» чи «No». Після підключення він по черзі задає питання та підраховує кількість правильних відповідей. Після закінчення опиту пересилає на клієнт кількість вірних відповідей
3	Створити програму віддаленого віртуального піаніно. На клієнті пишеться мелодія у вигляді нот-символів, потім вона відправляється на сервер, де мелодія програється. Коли сервер став вільним, він повідомляє клієнт повідомленням «Playing done». Для генерації звуків використати метод `Console.Beep()`
4	Створити програму-гру «Вгадай вік», на сервері задається число, клієнт відправляє варіант на сервер, сервер обробляє результат та відсилає «Equal», «More», «Less»
5	Організувати віддалений стек за принципом Last Input First Output (LIFO). Відправляти повідомлення з клієнта на сервер у вигляді «push *value*», де *value* – дійсне число. Команда «pop» вилучає з сервера останнє передане число. Команди «add» та «sub» додають та віднімають відповідно два числа, що знаходяться на вершині стеку на сервері, і потім сервер повертає результат клієнту
6	Розробити клієнт, який намагається виконати DDoS-атаку на сервер, посилаючи неперервний потік повідомлень «gettime». У випадку, коли навантаження на сервер не перевищує три

№ варіанта	Завдання
	повідомлення за секунду, у відповідь на запит «gettime» сервер повертає клієнту поточний локальний час, у зворотному випадку – сервер не відповідає. Розробити клієнт так, щоб можна було промоделювати різну частоту генерування повідомлень
7	Створити програму-чат між клієнтом та сервером
8	Створити клієнт для перегляду файлової структури сервера. На клієнті реалізувати такі операції: «ls» – показати перелік файлів поточного каталогу; «pwd» – показати поточний каталог; «cd..» – піднятися в дереві каталогів у батьківську директорію
9	Відправляти з клієнта на сервер команди у форматі «getrnd *n*», де *n* – кількість випадкових чисел, які сервер повертає клієнту у вигляді масиву
10	Відправляти числа з клієнта на сервер, сервер додає їх до масиву. Якщо надсилається повідомлення «show», то на клієнт посилається масив чисел, який виводиться на консоль
11	На клієнті реалізувати можливість отримання даних від сервера про поточну дату та час. У консолі клієнта запит здійснюється за допомогою повідомлень: «showdate, showtime»
12	Створити аналог команди ping. Луна-сервер повинен повторювати клієнту відправлені ним повідомлення, розмір повідомлення та час відповіді сервера
13	Відправляти на сервер повідомлення з 5 чисел та команду, що з ними робити. «SORTUP» – сортувати за зростанням, «SORTDOWN» –

№ варіанта	Завдання
	сортувати за спаданням. Результат повернути клієнту
14	Відправляти на сервер повідомлення з двома числами та команду, що з ними робити. «SUM» – знайти суму, «DEVIDE» – поділити, «MUL» – помножити. Результат повернути клієнту
15	Відправляти масив чисел з клієнта на сервер, сервер сортує їх за зростанням та відправляє результат на клієнт
16	Відправляти масив чисел з клієнта на сервер, сервер сортує їх за спаданням та відправляє результат на клієнт
17	Відправляти на сервер повідомлення, сервер випадковим чином переставляє в повідомленні букви та відправляє результат клієнту
18	Відправляти на сервер два восьмирозрядних двійкових числа, записаних з використанням символів «0» та «1». Сервер виконує побітові операції AND, OR, XOR між прийнятими числами та відправляє результат клієнту
19	Відправляти на сервер повідомлення, у відповіді від сервера вказати, чи є в повідомленні парні цифри, якщо так, то передати їх загальну кількість, якщо ні, то відповідне повідомлення
20	Відправляти декілька рядків тексту на сервер, сервер випадковим чином змінює порядок рядків та повертає клієнту
21	Створити програму-чат між клієнтом та сервером, заборонивши при цьому передавати числа та прописні літери
22	Відправляти двомірну матрицю на сервер, знаходити суму її елементів та відправляти

№ варіанта	Завдання
	результат на клієнт. Розмірність матриці та елементи повинні задаватися користувачем
23	Відправляти повідомлення з клієнта на сервер, сервер обробляє їх та повертає відповідь у вигляді довжини повідомлення
24	Відправляти на сервер повідомлення, у відповіді від сервера вказати, чи є в ньому цифри, якщо так, то передати їх загальну кількість, якщо ні, то відповідне повідомлення
25	Створити аналог команди ping. Луна-сервер повинен повторювати клієнту відправлені ним повідомлення у зворотному порядку
26	Відправляти на сервер повідомлення з 8 чисел та команду, що з ними робити. «VAR» – розрахувати дисперсію, «MEAN» – розрахувати математичне очікування. Результат повернути клієнту
27	Створити елемент скриптової мови. Клієнт відправляє команди серверу: «*var = value*» – задати змінній *var* значення *value* (наприклад, «a = 5»); «*var*» – вивести на консоль значення змінної *var*. Усі змінні повинні зберігатися на сервері
28	Відправляти масив чисел з клієнта на сервер, сервер сортує тільки від'ємні значення за зростанням та відправляє результат на клієнт
29	Розробити просту програму-тестування. Сервер має список питань, на які можна дати відповіді «Yes» чи «No». Після підключення він по черзі задає питання та підраховує кількість помилок. Після закінчення опиту пересилає на клієнт результати тестування

№ варіанта	Завдання
30	Створити програму-чат між клієнтом та сервером, перевіряючи при підключенні пароль, встановлений на сервері

Контрольні питання

1. Що таке мережний сокет?
2. Які основні принципи мережної взаємодії додатків із застосуванням портів?
3. Коротко охарактеризуйте стек протоколів TCP/IP.
4. Перерахуйте номери портів таких протоколів та служб: HTTP, DNS, FTP, Telnet, ICQ, Skype.
5. Як співвідноситься модель відкритої взаємодії OSI та стек протоколів TCP/IP?
6. Чим відрізняються протоколи транспортного рівня TCP та UDP?
7. Коротко опишіть схему створення сокетного TCP з'єднання.
8. Які засоби платформи .NET Framework призначені для передачі даних між вузлами комп'ютерної мережі?
9. Запропонуйте підхід до створення багатоклієнтського сервера на основі протоколу TCP.
10. Яким чином можна реалізувати луна-сервер та визначити час передачі даних між вузлами?

ТЕМА 4. СТВОРЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ ІЗ ЗАСТОСУВАННЯМ ТЕХНОЛОГІЇ ДОСТУПУ ДО БАЗ ДАНИХ ADO.NET

Мета: розробити клієнтський додаток типу WinForms із застосуванням технології ADO.NET для оброблення інформації із бази даних під управлінням Microsoft SQL Server Compact Edition, розробити запити на мові SQL.

Основи технологій доступу до баз даних

ADO.NET (ActiveX Data Objects .NET) – це набір бібліотек, що поставляється з Microsoft .NET Framework і призначений для взаємодії з різними сховищами даних з .NET-додатків. Бібліотеки ADO.NET включають класи для приєднання до джерела даних, виконання запитів і обробки їхніх результатів. Крім того, ADO.NET можна використовувати в якості надійного, ієрархічно організованого, відокремленого кешу даних для автономної роботи з даними. ADO.NET була розроблена компанією Microsoft для вирішення проблем, які виникали при роботі з ADO та попередніми технологіями, такими як: Data Access Objects (DAO), Remote Data Objects (RDO) [15, 36].

ADO.NET має багато переваг порівняно з іншими технологіями доступу до даних, а саме: підтримка XML, простота модифікації та програмування, висока продуктивність.

ADO також підтримує XML, але не буде так само ефективно обробляти XML-дані, як це робить ADO.NET, оскільки ADO.NET створювався з урахуванням XML.

Протягом терміну служби системи в неї можна вносити незначні зміни, однак спроби провести архітектурні зміни трапляються рідко, через виняткову складність завдання. На жаль, при природному розвитку подій такі зміни іноді виявляються необхідними.

Компоненти даних ADO.NET в Visual Studio інкапсулюють функціональні можливості доступу до даних різними способами, що допомагає розробляти програмні продукти значно швидше і з меншою кількістю помилок.

Для непідключених додатків набори даних ADO.NET дають виграш у продуктивності у порівнянні з непідключеними наборами записів ADO. Передача непідключеного набору записів між рівнями за допомогою COM-упаковки може призвести до великої витрати обчислювальних ресурсів, тому що значення в наборі записів перетворюються до типів даних, відомих COM. У ADO.NET таке перетворення типів даних непотрібно.

Моделі ADO.NET

Бібліотеки ADO.NET можна застосовувати трьома концептуально різними способами: у підключеному режимі, в автономному режимі та за допомогою технології Entity Framework. При використанні підключеного рівня (Connected Layer) кодова база безпосередньо підключається до відповідного сховища даних та відключається від нього. При такому способі використання ADO.NET взаємодія зі сховищем даних звичайно проводиться за допомогою об'єктів підключення, об'єктів команд та об'єктів читання даних.

Автономний рівень (Disconnected Layer) дозволяє працювати з набором об'єктів DataTable, які містяться в DataSet, котрий на стороні клієнта надає копію зовнішніх даних. При отриманні DataSet із допомогою відповідного об'єкта адаптера даних підключення відкривається та закривається автоматично. Такий підхід дозволяє швидко звільняти підключення для інших викликів та підвищує масштабованість системи.

Отримавши об'єкт `DataSet`, викликаючий код може розглядати та обробляти дані без витрат на мережевий трафік. А якщо потрібно занести зміни до сховища даних, то адаптер даних (разом з набором операторів SQL) використовується для оновлення даних; при цьому підключення відкривається заново

для проведення оновлення в базі даних, а потім одразу ж закривається [2, 16, 20, 31, 38].

Після випуску .NET 3.5 SP1 в ADO.NET з'явилася підтримка нового API, який називається Entity Framework (EF). Технологія EF показує, що багато низькорівневих деталей роботи з базами даних (наприклад, складні SQL-запити) приховані від програміста та оброблюються за нього при генерації відповідного LINQ-запита (LINQ Entities).

Постачальник даних ADO.NET

Постачальник даних .NET – це набір класів, призначених для взаємодії зі сховищем даних певного типу. .NET Framework включає два постачальники – SQL Client.NET Data Provider і OLE DB.NET Data Provider. Постачальник OLE DB .NET Data Provider дозволяє взаємодіяти з різними сховищами даних за допомогою постачальника OLE DB. Постачальник SQL Client.NET Data Provider розрахований виключно на взаємодію з БД SQL Server. Кожен постачальник даних. NET реалізує однакові базові класи – Connection, Command, DataProvider, Parameter і Transaction, конкретне ім'я яких залежить від постачальника (табл. 4.1). У постачальника SQL Client .NET Data Provider є об'єкт SqlConnection, а у постачальника OLE DB .NET Data Provider – об'єкт OleDbConnection.

Таблиця 4.1

Основні об'єкти постачальників даних ADO.NET

№ з/п	Постачальник даних	Простір імен	Бібліотека
1	`OLE DB`	`System. Data.OleDb`	`System.Data.dll`
2	`Microsoft SQL Server`	`System.Data. SqlClient`	`System.Data.dll`
3	`Microsoft SQL Server Mobile`	`System.Data. SqlServerCe`	`System.Data. SqlServerCe.dll`

№ з/п	Постачальник даних	Простір імен	Бібліотека
4	ODBC	System. Data.Odbc	System. Data.dll
5	Oracle	System. Data.OracleClient	System.Data. OracleClient.dll

У таблиці 4.2 наведено деякі загальні основні об’єкти, їх базові класи (визначені у просторі імен System.Data.Common) і основні інтерфейси (визначені у просторі імен System.Data), які вони реалізують.

Таблиця 4.2

Основні об’єкти постачальників даних ADO.NET

№ з/п	Тип об’єкта	Базовий клас	Відповідні інтерфейси
1	Connection	DbConnection	IDbConnection
	Дозволяє підключатися до сховища даних та відключатися від нього. Крім того, об’єкти підключення забезпечують доступ до відповідних об’єктів транзакцій		
2	Command	DbCommand	IDbCommand
	Представляє SQL-запит чи збережену процедуру. Крім того, об’єкти команд надають доступ до об’єктів читання даних конкретного постачальника даних		
3	DataReader	DbDataReader	IDataReader, IDataRecord
	Надає доступ до даних тільки для читання у прямому напрямку за допомогою курсора на стороні сервера		
4	DataAdapter	DbDataAdapter	IDataAdapter, IDbDataAdapter
	Пересилає набори даних із сховища даних до процесу і в зворотному напрямі. Адаптери даних містять підключення та набір із чотирьох внутрішніх об’єктів команд для вибірки, вставки, зміни та видалення інформації з бази даних		

№ з/п	Тип об'єкта	Базовий клас	Відповідні інтерфейси
5	`Parameter`	`DbParameter`	`IDataParameter, IDbDataParameter`
	Представляє іменований параметр у параметризованому запиті		
6	`Transaction`	`DbTransaction`	`IDbTransaction`
	Інкапсулює транзакцію у базі даних		

Простори імен ADO.NET

Крім просторів імен, які визначають типи конкретних постачальників даних, у бібліотеках базових класів .NET містяться додаткові простори імен, орієнтовані на ADO.NET. Деякі з них перелічені в таблиці 4.3.

Таблиця 4.3

Додаткові простори імен для роботи з ADO.NET

№ з/п	Простір імен	Призначення
1	`Microsoft. SqlServer. Server`	Містить типи для роботи служби інтеграції CLR та SQL Server
2	`System.Data`	Визначає основні типи ADO.NET, які використовуються всіма постачальниками даних, у тому числі загальні інтерфейси та різні типи, що забезпечують автономний рівень (наприклад, `DataSet, DataTable` та ін.)
3	`System.Data. Common`	Містить типи для загального використання всіма постачальниками ADO.NET, у тому числі і загальні абстрактні базові класи
4	`System.Data. Sql`	Містить типи, які дозволяють визначити екземпляри Microsoft SQL Server, які встановлені у локальній мережі
5	`System.Data. Sql.Types`	Містить типи даних, які використовує Microsoft SQL Server

Приклад виконання завдання

Створити власну базу даних (БД) типу Microsoft SQL Server Compact Edition. Додати до бази даних одну таблицю «Ноутбуки». Заповнити таблицю за допомогою інструментів Visual Studio, додавши до неї декілька записів. Програмно вивести зміст таблиці на головне вікно додатка типу Windows Forms за допомогою компонента DataGridView. Додатково створити конструктор запитів для одного з полів таблиці для вибірки записів за цим критерієм (наприклад, відібрати всі ноутбуки з обсягом оперативної пам'яті не менше 4 ГБ.)

Розв'язання

Спочатку потрібно створити проект типу Windows Form [39], який дозволяє проектувати графічний інтерфейс користувача з використанням елементів керування Windows (рис. 4.1).

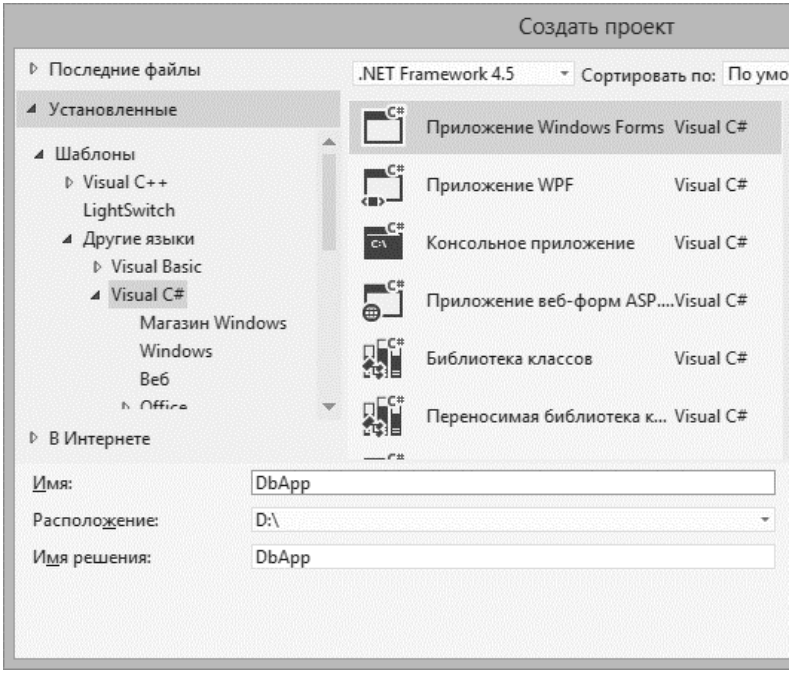


Рис. 4.1. Діалог створення нового проекту WinForms

До проекту необхідно додати новий елемент (рис. 4.2).

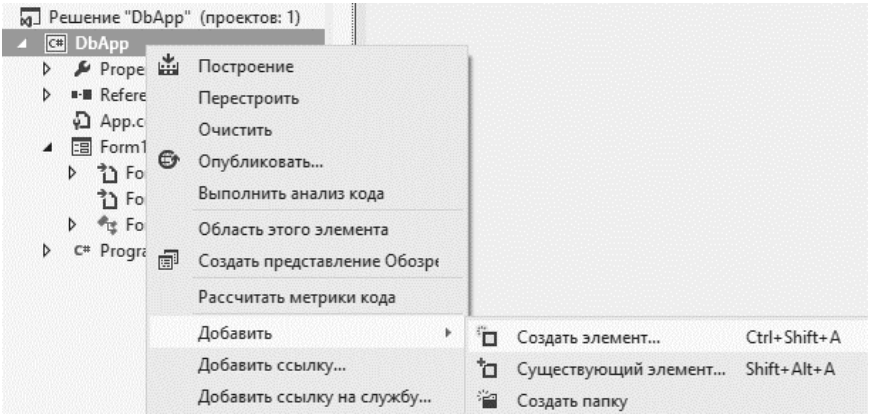


Рис. 4.2. Додавання до проекту нового елемента

У якості сховища даних потрібно обрати локальну базу даних типу SQL Server Compact Edition (рис. 4.3).

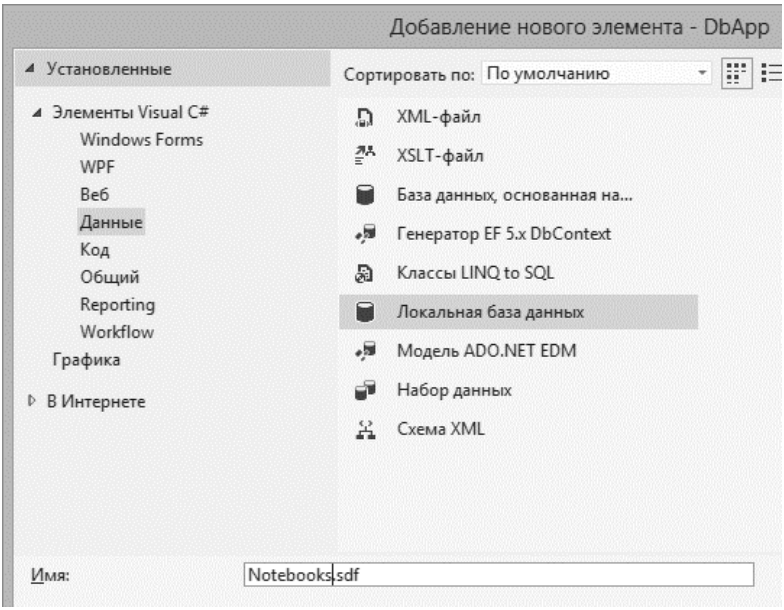


Рис. 4.3. Створення локальної бази даних Notebooks

Для роботи зі створеною базою даних необхідно відкрити переглядач серверів за допомогою відповідного пункту меню «Вид» (рис. 4.4).

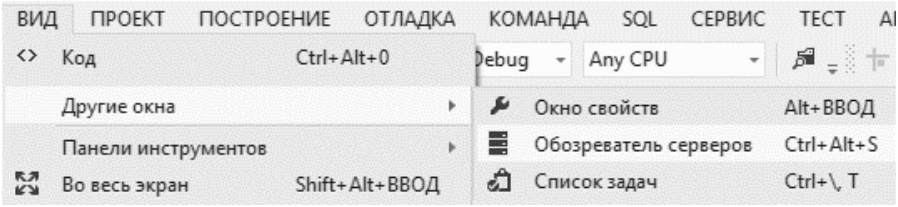


Рис. 4.4. Відображення вікна «Обозреватель серверов»

Після підключення до бази даних потрібно додати таблицю за допомогою контекстного меню вкладки «Таблиці» (рис. 4.5).

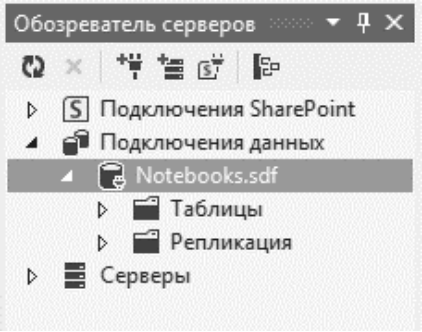


Рис. 4.5. Перегляд структури бази даних Notebooks

Для опису ноутбуків застосовано первинний ключ ID. Таблиця бази даних містить поля щодо моделі ноутбука, встановленого процесора, обсягу оперативної пам'яті та жорсткого диска, а також ціну на дану модель (рис. 4.6).

Після створення таблиці її потрібно наповнити даними (рис. 4.7, 4.8).

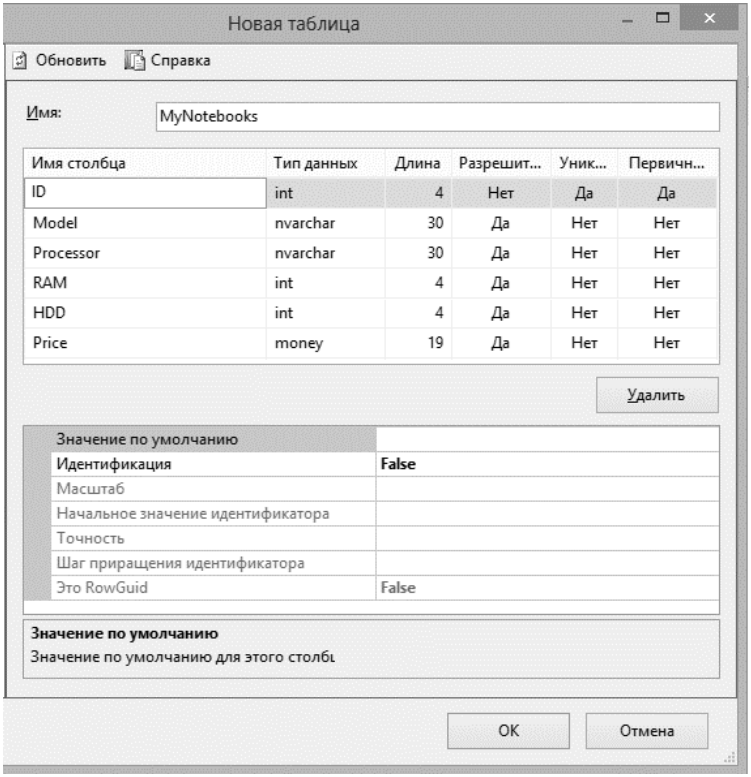


Рис. 4.6. Створення структури таблиці, що містить інформацію про ноутбуки

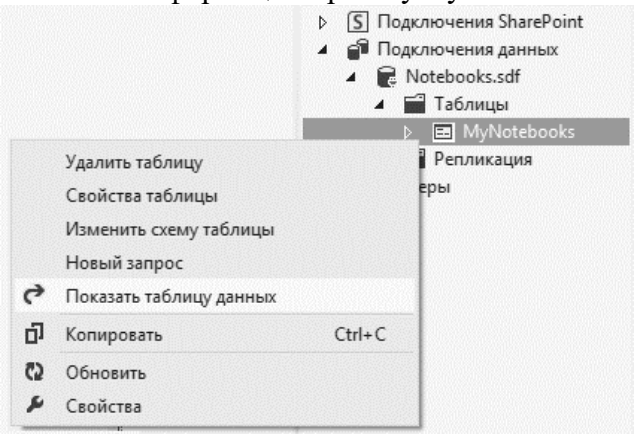


Рис. 4.7. Відображення таблиці для введення даних

MyNotebooks: Зап...pp\Notebooks.sdf) | Form1.cs [Конструктор]*

	ID	Model	Processor	RAM	HDD	Price
	1	HP 655 (B6N22EA)	AMD Dual-Core E2-...	4	500	3176
	2	Lenovo IdeaPad B570e	Intel Pentium B950	4	500	3323
	3	Lenovo IdeaPad G580GH	Intel Core i3-2328M	4	1000	4117
	4	Asus X301A (X301A-RX157D)	Intel Core i3-3110M	4	500	4348
▶	5	Samsung 350V5 (NP350V5C...	Intel Core i5-3210M	6	750	7343
*	NULL	NULL	NULL	NULL	NULL	NULL

Рис. 4.8. Додавання даних до таблиці

Для роботи з SQL Server Compact Edition необхідно додати посилання на збірку C:\Program Files\Microsoft SQL Server Compact Edition\v4.0\Desktop\System.Data.SqlServerCe.dll (рис. 4.9, 4.10).

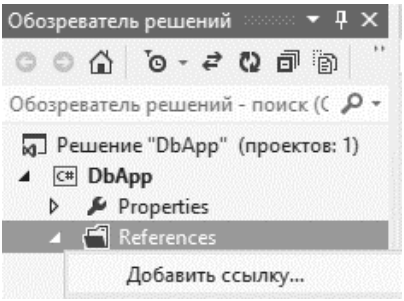


Рис. 4.9. Додавання нової збірки SQL Server CE

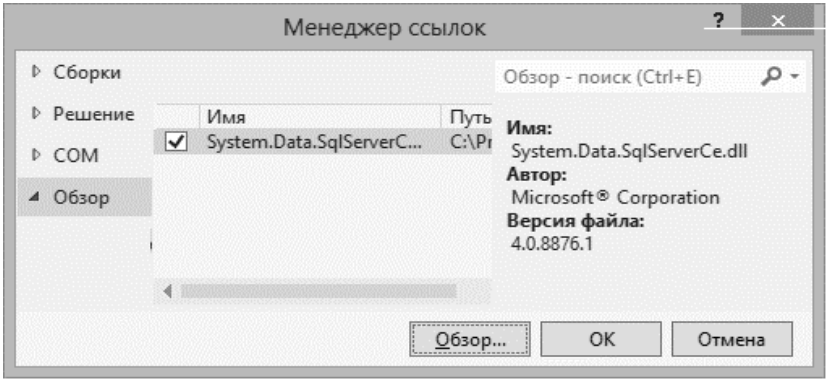


Рис. 4.10. Підключення збірки System.Data.SqlServerCe.dll до проекту

Після цього в програмному коді потрібно додати посилання на простір імен `using System.Data.SqlServerCe`.

Для створення графічного інтерфейсу користувача застосовані об'єкти: DatagridView, Label, MaskedTextBox, Button (рис. 4.11).

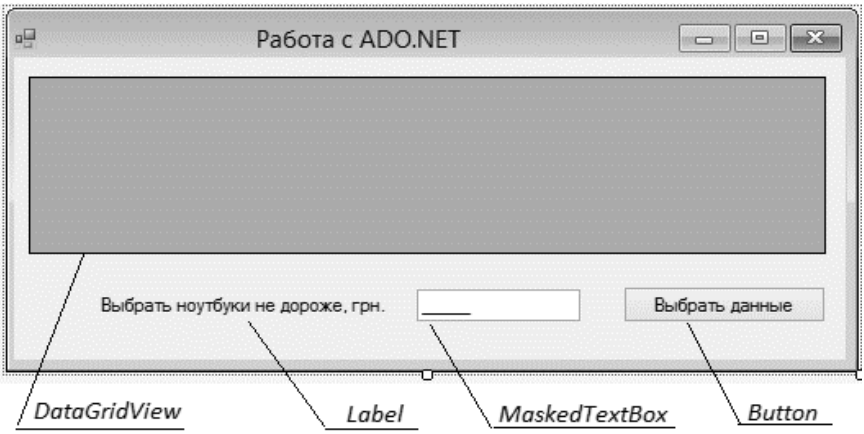


Рис. 4.11. Додавання елементів у конструкторі форм

Після коригування імен візуальних компонентів за допомогою вікна «Властивості» та налаштування компонента MaskedTextBox програмний код може мати такий вигляд:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlServerCe; // додати
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
```

```
namespace DbApp
{
  public partial class Form1 : Form
  {
      public Form1()
      {
         InitializeComponent();
      }

  private void buttonSelect_Click(object sender,
                                  EventArgs e)
  {
    SqlCeConnection con =
      new SqlCeConnection("Data Source=
                  D:\\DbApp\\DbApp\\Notebooks.sdf");
    // SQL-запит
    string sqlStr = "Select * from MyNotebooks
                     where Price <= " +
                     maskedTextBox.Text;
    SqlCeDataAdapter dataAdapter =
      new SqlCeDataAdapter(sqlStr, con);
    DataSet queryResult = new DataSet();
    dataAdapter.Fill(queryResult);
    dataGridView.DataSource = queryResult.Tables[0];
    con.Close();
  }
}
```

Можна впевнитися, що програма працює з локальною базою даних, виконуючи відповідний SQL-запит для вибору ноутбуків, які не дорожчі за вказану ціну (рис. 4.12).

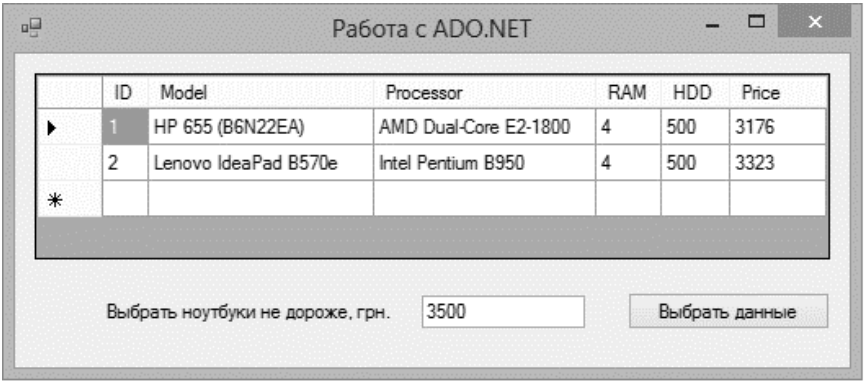


Рис. 4.12. Тестування додатка

Зауваження: з метою спрощення пояснення роботи з ADO.NET шлях до бази даних задавався жорстким посиланням у коді. Звичайно, у ситуації переміщення бази даних програма перестане працювати. Найбільш доцільно у цьому випадку або використати поточний локальний каталог, з якого запускається додаток, наприклад, за допомогою `System.IO.Directory.CurrentDirectory()`, або застосувати діалогове вікно відкриття файлу бази даних `OpenFileDialog`.

Завдання для самостійного виконання

Відповідно до варіанта та обраного рівня складності виконати такі завдання.

№	Рівень завдань		
	Початковий	Базовий	Високий
1	Відкомпілювати та протестувати аудиторні завдання	Завдання №1	Завдання №2
2	–	Завдання №2	Завдання №3

Завдання №1

Створити власну базу даних типу Microsoft SQL Server Compact Edition. Додати до бази даних одну таблицю, яка містила б не менше чотирьох інформаційних полів. Заповнити

таблицю за допомогою інструментів Visual Studio, додавши не менше п’яти записів. Програмно вивести зміст таблиці на головне вікно додатка типу Windows Forms за допомогою компонента DataGridView. Таблицю індивідуальних варіантів подано нижче.

Завдання №2

Виконати завдання 1. Додатково створити два запити типу Select, результати яких повинні виводитись до DataGridView. Таблицю індивідуальних варіантів подано нижче.

Завдання №3

Виконати завдання 1. Додатково створити конструктор запитів для одного з полів таблиці для вибірки записів за цим критерієм (наприклад, відібрати всіх студентів, у яких поле «вік» > 21, або «бал» < 4 тощо). Таблицю індивідуальних варіантів подано нижче.

№ варіанта	Завдання
1	Таблиця «Студентська група»
2	Таблиця «Результати екзамену»
3	Таблиця «Склад кафедри»
4	Таблиця «Факультети університету»
5	Таблиця «Предмети курсу»
6	Таблиця «Міста України»
7	Таблиця «Розклад занять»
8	Таблиця «Математична довідка»
9	Таблиця «Погода на тиждень»
10	Таблиця «Моє місто – основні дані»
11	Таблиця «Комп’ютери»
12	Таблиця «Кадри»
13	Таблиця «Товари»
14	Таблиця «Лікарські препарати»
15	Таблиця «Автомобілі»
16	Таблиця «Пацієнти»

№ варіанта	Завдання
17	Таблиця «Абітурієнти»
18	Таблиця «Закони»
19	Таблиця «Фільми»
20	Таблиця «Музика»
21	Таблиця «Мікропроцесори»
22	Таблиця «Монітори»
23	Таблиця «Планшети»
24	Таблиця «Мобільні телефони»
25	Таблиця «Мережеві кабелі»
26	Таблиця «Журнали»
27	Таблиця «Книги»
28	Таблиця «Розклад»
29	Таблиця «Операційні системи»
30	Таблиця «Мови програмування»

Контрольні питання

1. Коротко охарактеризуйте технологію ADO.NET.
2. Які основні моделі роботи зі сховищами даних передбачено у ADO.NET?
3. Для чого використовуються провайдери даних?
4. Призначення та функціональність класів SqlConnection, SqlDataAdapter, DataSet.
5. Які візуальні елементи керування WinForms потрібні для побудови додатка для роботи з базами даних?
6. Синтаксис SQL-оператора Select.
7. Як вибрати дані з декількох таблиць бази даних?
8. Як видалити рядки з таблиці, які відповідають певній умові?
9. Дайте коротку характеристику системі керування базами даних SQL Server CE.
10. Що таке нормалізація баз даних?

ТЕМА 5. РОЗРОБКА WEB-САЙТУ НА БАЗІ ТЕХНОЛОГІЙ ASP.NET, HTML, CSS, JAVA SCRIPT ТА FLASH

Мета: розробити Web-сайт у складі корпоративної інформаційної системи підприємства.

Огляд технології ASP.NET

ASP.NET – технологія створення веб-додатків і веб-сервісів від компанії Microsoft. Вона є складовою частиною платформи Microsoft .NET і розвитком старішої технології Microsoft ASP. ASP.NET зовні багато в чому зберігає схожість із старішою технологією ASP, що дозволяє розробникам відносно легко перейти на ASP.NET. У той же час внутрішня архітектура ASP.NET істотно відрізняється від ASP, оскільки вона заснована на платформі .NET і, отже, використовує всі нові можливості, що надаються цією платформою.

Хоча ASP.NET бере свою назву від старої технології Microsoft ASP, вона значною мірою від неї відрізняється. Microsoft повністю перебудувала ASP.NET, ґрунтуючись на Common Language Runtime (CLR), який є основою всіх додатків Microsoft. NET. Розробники можуть писати код для ASP.NET, використовуючи практично будь-які мови програмування, що входять у комплект. NET Framework (C#, Visual Basic .NET, і JScript. NET). ASP.NET має перевагу у швидкості в порівнянні зі скриптовими технологіями, тому що при першому зверненні код компілюється і розміщується у спеціальний кеш, і згодом тільки виконується, не вимагаючи витрат часу на парсинг, оптимізацію та ін.

Якщо зазвичай Rich Media Application створюють за допомогою Flash, то тепер ASP.NET передбачає схожі можливості за допомогою модуля Silverlight. Корпорація Microsoft випустила декілька розширень для ASP.NET: ASP.NET AJAX, ASP.NET MVC Framework.

Роль протоколу HTTP

Веб-додатки значною мірою відрізняються від настільних графічних додатків. Очевидна суттєва відмінність полягає в тому, що професійний веб-додаток завжди вимагає існування щонайменше двох машин, об'єднаних у мережу: на одній знаходиться веб-сайт, а на іншій переглядаються дані за допомогою веб-браузера [15, 23, 32]. Зрозуміло, що під час розробки, роль клієнта і веб-сервера виконуватиме один комп'ютер. Це можливо завдяки тому, що до складу Visual Studio входить портативна версія веб-сервера Microsoft Internet Information Services (IIS). Мережний протокол, який використовується для роботи веб-сайтів, називається протоколом передачі гіпертексту (HTTP) [19].

Коли на клієнтській машині запускається веб-браузер (наприклад, такий як Google Chrome, Opera, Mozilla Firefox, Apple Safari, Microsoft Internet Explorer), виконується HTTP-запит для доступу до певного ресурсу (наприклад, до веб-сторінки типу *www.facebook.com*). Веб-сервер приймає запит та виконує динамічне генерування сторінки HTML (рис. 5.1).

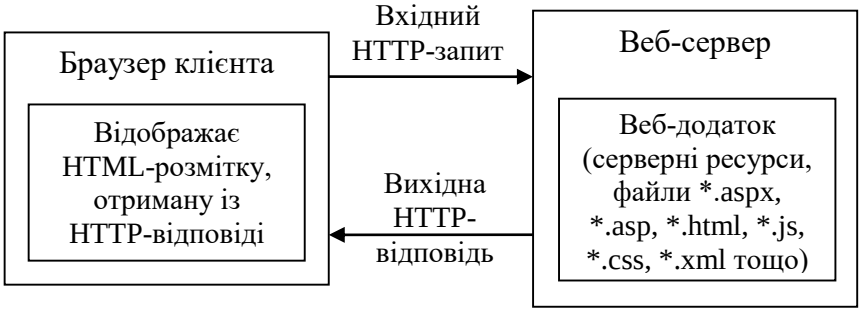


Рис. 5.1. Цикл запиту/відповіді HTTP

Інший аспект веб-розробки, який суттєво відрізняє її від програмування традиційних настільних додатків, полягає в тому, що HTTP – це мережний протокол без збереження стану. Після відправки з веб-сервера відповіді клієнтському браузеру весь хід їх попередньої взаємодії забувається. Технологія

ASP.NET пропонує багато різних способів підтримки стану: змінні сеансу, cookie-набори, кеш додатка, API-інтерфейс керування профілями ASP.NET.

Інструменти візуального конструювання Web-форм

У панелі інструментів (ToolBox) середовища розробки Visual Studio міститься велика кількість компонентів для розміщення їх на веб-формах (рис. 5.2).

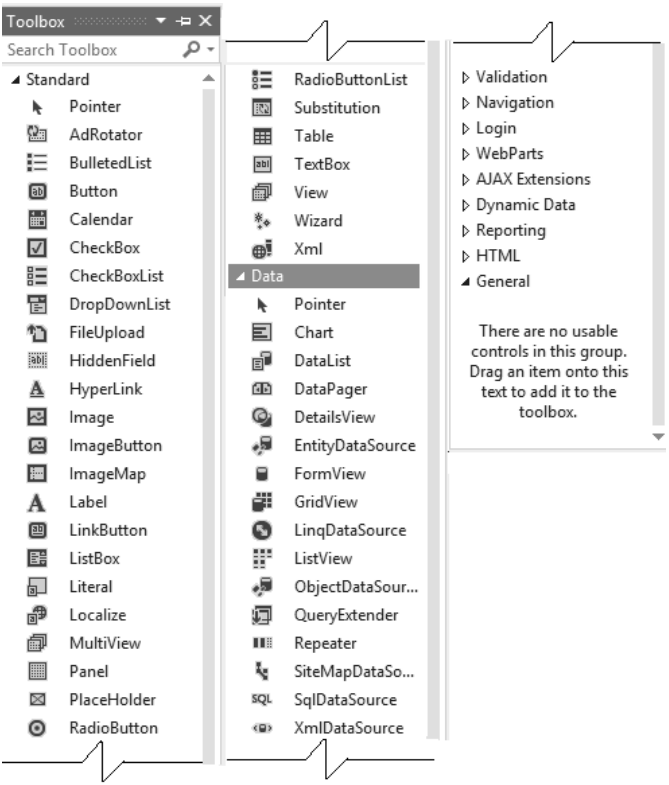


Рис. 5.2. Набір компонентів ASP.NET WebForms

Проектування веб-додатків з використанням фреймворку ASP.NET дуже схоже на розробку настільних додатків типу Windows Froms або Windows Presentation Foundation (WPF).

Вікно візуального конструктора дозволяє проектувати веб-сторінки шляхом перетягування неохідних компонентів на поле форми, або безпосередньо в область програмного коду (рис. 5.3)

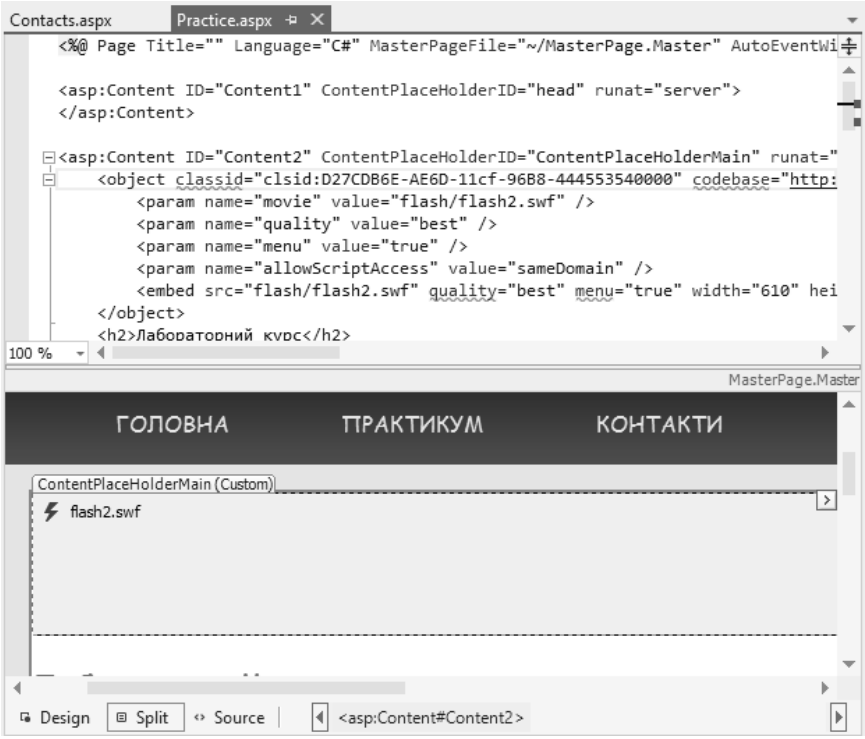


Рис. 5.3. Візуальний конструктор HTML у Visual Studio

Директиви сторінок ASP.NET

Типовий файл скрипту *.aspx, як правило, розпочинається набором директив. Директиви ASP.NET завжди відмічаються маркерами `<%@ … %>` і можуть містити різні атрибути.

Кожен файл *.aspx повинен містити щонайменше директиву `<%@ Page %>`, яка слугує для визначення керованої мови. Перелік найбільш уживаних атрибутів вищезазначеної директиви подано у таблиці 5.1.

Таблиця 5.1

Основні атрибути директиви <%@ Page %>

Атрибут	Призначення
CodePage	Вказує назву пов'язаного cs-файлу
EnableTheming	Вказує, чи підтримує елемент керування на .aspx-сторінці теми ASP.NET
EnableViewState	Вказує, чи підтримується стан представлення між запитами сторінки
Inherits	Визначає клас у файлі відокремленого коду, від якого успадковується сторінка; може бути будь-яким класом, похідним від System.Web.UI.Page
MasterPageFile	Встановлює майстер-сторінку, яка використовуватиметься з поточною
Trace	Дозволяє виконати трасування

Більш детально процес створення простого веб-сайту з компонентами ASP.NET WebForms розглянемо на практичній задачі. Вважається, що користувач має базові знання для роботи з HTML-розміткою та таблицями стилів CSS.

Приклад виконання завдання

Завдання №1

Створити Web-сайт дисципліни «Мережні інформаційні технології». Сайт повинен містити три сторінки ASP.NET: головну, перелік практичних робіт та контакти. Необхідно реалізувати всі основні компоненти сайту: заголовок (header), меню (nav), основний контент (article) та нижній колонтитул (footer). Додатково створити декілька flash-банерів та розмістити їх на сайті. З використанням мови Java Script додатково створити рекламний банер з випадковим контентом.

Розв'язання

Спочатку потрібно створити Web-сайт за допомогою меню Visual Studio: *File* | *New* | *Web-site* (рис. 5.4).

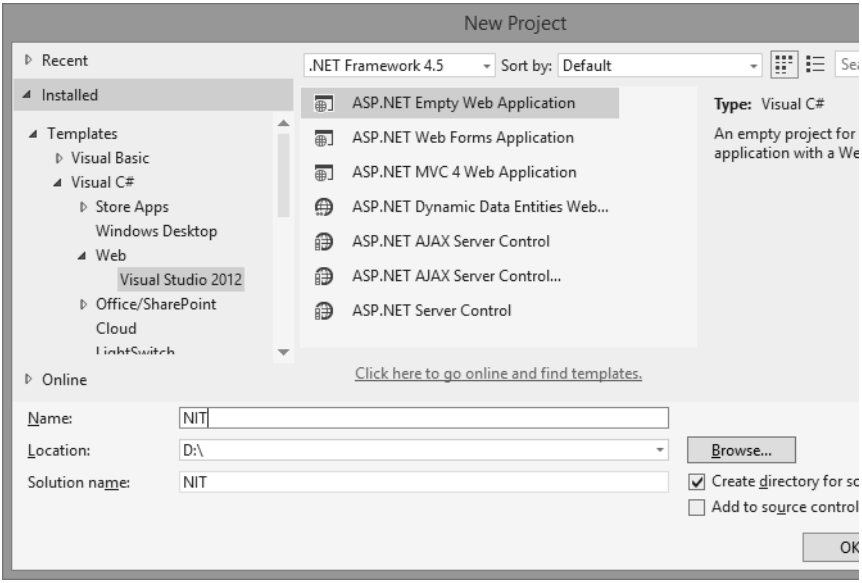


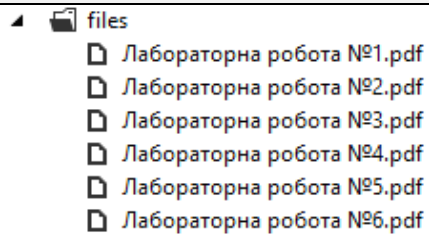
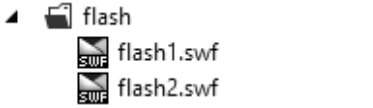
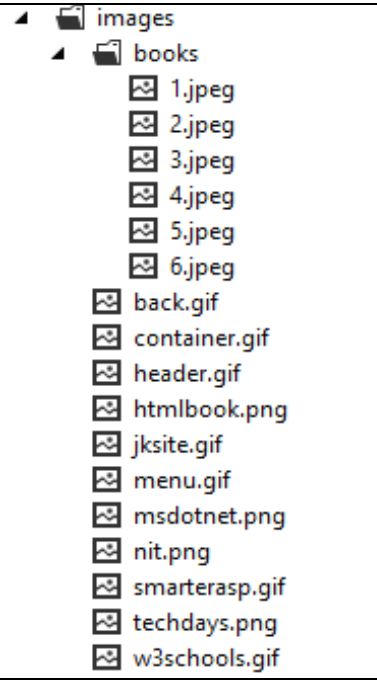
Рис. 5.4. Створення нового пустого сайту ASP.NET

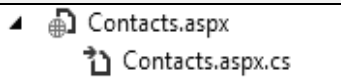
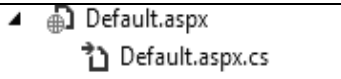
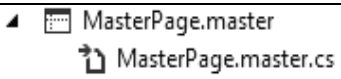
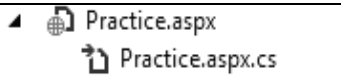
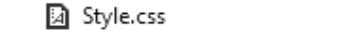
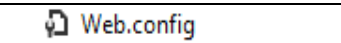
Для створення сайту скористуємось шаблоном із сайту *www.freecsstemplates.org*. За допомогою контекстного меню Visual Studio до проекту додаються відповідні каталоги, Web-форми та CSS-файл. Із використанням файлового менеджера до каталогів проекту копіюються всі потрібні файли даних (*pdf, jpeg, png*). Після оновлення вмісту каталогів у переглядачі рішень за допомогою контекстного меню структура проекту буде мати такий вигляд (табл. 5.2).

Таблиця 5.2

Опис елементів проекту

Елемент проекту	Короткий опис
Обозреватель решений - поиск (Ctrl+;) Решение "NIT" (проектов: 1) NIT	Перелік проектів, які містяться в рішенні NIT

Елемент проекту	Короткий опис
	Лабораторні роботи, які можна завантажити зі сторінки Practice.aspx
	Директорія містить згенеровані flash-файли, наприклад, за допомогою сайту *www.flashvortex.com*
	Файл коду Java Script, який містить функцію випадкового генерування рекламного банера
	Перелік рисунків, які використовуються для представлення (верстки) сайту та переліку книг

Елемент проекту	Короткий опис
	Сторінка контактів
	Домашня (головна) сторінка сайту
	Сторінка, яка використовується у якості шаблонної для генерування інших сторінок сайту
	Сторінка, на якій розміщено перелік лабораторних робіт з можливістю їх завантаження
	Файл таблиць каскадних стилів
	Конфігураційний файл, який створюється Visual Studio за умовчанням

ASP.NET сторінка MasterPage дозволяє створювати шаблон (заготовку) для декількох сторінок сайту. Одна головна сторінка (master page) визначає вигляд і стандартну поведінку, яку Ви хочете одержати від усіх сторінок (групи сторінок) сайту. Потім можна створити окремі сторінки, які будуть містити індивідуальний контент. Коли користувач запитує сторінки сайту, вони поєднуються з головною сторінкою для видачі кінцевого HTML-файлу. У цьому випадку місцезамінником слугує елемент *ContentPlaceHolder* (рис. 5.5).

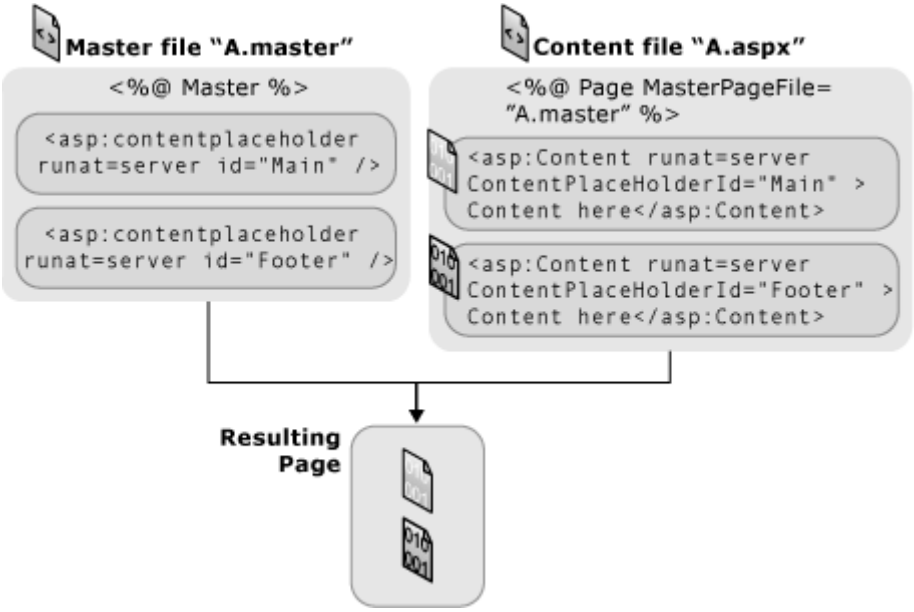


Рис. 5.5. Генерування кінцевої сторінки
з двох файлів *A.master* та *A.aspx*

Нижче наводиться програмний код усіх сторінок сайту.

Лістинг файлу MasterPage.master

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="MasterPage.master.cs" Inherits="MasterPage"
%>

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <meta http-equiv="Content-Type"
    content="text/html; charset=utf-8" />
  <title>Мережні інформаційні технології</title>
  <meta name="keywords" content="C#, .NET, ADO.NET,
    ASP.NET, IT, Socket" />
  <meta name="description" content="Курс дисципліни
    мережні інформаційні технології" />
```

```
  <meta name="author" content="Кафедра комп'ютерних
    систем та мереж КНУ" />
  <link href="Style.css" rel="stylesheet" />
  <script type="text/javascript"
    src="js/main.js"></script>
</head>
<body>
  <form id="formMain" runat="server">
    <header>
      <h1>Мережні інформаційні технології</h1>
    </header>
    <nav>
      <asp:Menu ID="MenuMain" runat="server"
        Orientation="Horizontal">
        <Items>
          <asp:MenuItem
            NavigateUrl="Default.aspx"
            Text="Головна"
            Value="Головна">
          </asp:MenuItem>
          <asp:MenuItem
            NavigateUrl="Practice.aspx"
            Text="Практикум"
            Value="Практикум">
          </asp:MenuItem>
          <asp:MenuItem
            NavigateUrl="Contacts.aspx"
            Text="Контакти"
            Value="Контакти">
          </asp:MenuItem>
        </Items>
        <StaticHoverStyle BackColor="#025586" />
        <StaticMenuItemStyle CssClass="menuitem"
          HorizontalPadding="40px"
        VerticalPadding="17px" />
      </asp:Menu>
    </nav>
    <article>
```

```
      <div id="content">
        <asp:ContentPlaceHolder
          ID="ContentPlaceHolderMain" runat="server">
        </asp:ContentPlaceHolder>
      </div>
      <div id="sidebar">
        <asp:ContentPlaceHolder
          ID="ContentPlaceHolderBar" runat="server">
        </asp:ContentPlaceHolder>
      </div>
    </article>
    <footer>
      <p>Copyright (c) 2012 КСМ.
         Всі права захищено</p>
     </footer>
  </form>
</body>
</html>
```

Лістинг файлу Default.aspx

```
<%@ Page Title="" Language="C#"
MasterPageFile="MasterPage.master"
AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<asp:Content ID="Content1"
  ContentPlaceHolderID="ContentPlaceHolderMain"
  runat="Server">
  <object classid="clsid:D27CDB6E-AE6D-11cf-
    96B8-444553540000"
    codebase="http://download.macromedia.com/pub/
    shockwave/cabs/flash/swflash.cab#version=9,0,0,0"
    width="634" height="74">
      <param name="movie" value="flash/flash1.swf" />
      <param name="quality" value="best" />
      <param name="menu" value="true" />
      <param name="allowScriptAccess"
        value="sameDomain" />
```

```
    <embed src="flash/flash1.swf" quality="best"
      menu="true" width="634" height="74"
      type="application/x-shockwave-flash"
      pluginspage="http://www.macromedia.com/go/
      getflashplayer"
    allowscriptaccess="sameDomain" />
</object>
<asp:Image ID="Image1" runat="server"
  ImageUrl="images/nit.png"
  AlternateText="Network IT"
CssClass="images" />
<p>
Програма вивчення нормативної дисципліни
<strong>"Мережні інформаційні технології"</strong>
складена відповідно до місця та значення дисципліни
за структурно-логічною схемою, передбаченою
освітньо-професійною програмою спеціаліста зі
спеціальності 7.05010201 "Комп'ютерні системи та
мережі", і охоплює всі змістові модулі, визначені
анотацією для мінімальної кількості годин,
передбачених стандартом.
</p>
<p>
Предметом вивчення дисципліни <strong>"Мережні
інформаційні технології"</strong> є технології
розробки мережного програмного забезпечення,
вивчення принципів побудови інформаційних систем на
базі розподілених баз даних та Web-додатків.
</p>
<p>
Міждисциплінарні зв'язки: дисципліна
<strong>"Мережні інформаційні технології"</strong>
вимагає знань про структуру протоколів стека TCP/IP,
що вивчаються дисципліною "Основи комп'ютерних
мереж". Якісне вивчення курсу <strong>"Мережні
інформаційні технології"</strong> ґрунтується на
знаннях у галузі програмування на мові C/C++,
об'єктно-орієнтованого програмування та знань,
```

```
    одержаних протягом вивчення "Основи Web-технологій"
    та "Організації баз даних".
  </p>
</asp:Content>

<asp:Content ID="Content2"
  ContentPlaceHolderID="ContentPlaceHolderBar"
  runat="Server">
  <h2>Корисне</h2>
  <p>MSDN C# туторіали</p>
  <a href="http://msdn.microsoft.com/en-
    us/library/618ayhy6(v=vs.71).aspx">
    <img src="images/msdotnet.png" border="0"
      alt="C# Programmer's Reference"></a>

  <p>Довідковий ресурс HTML, CSS, JavaScript,
     ASP.NET</p>
  <a href="http://www.w3schools.com">
    <img src="images/w3schools.gif" border="0"
    alt="Visit W3Schools">
  </a>

  <p>Windows Server 2012 хостинг з підтримкою IIS8,
    ASP.NET 4.5, MS SQL 2012</p>
  <a href="http://www.smarterasp.net/index?r=vanopro">
    img src="images/smarterasp.gif" border="0"></a>

  <p>Корисні ресурси</p>
  <script type="text/javascript">
    RndBanner();
  </script>
</asp:Content>
```

Лістинг файлу Practice.aspx

```
<%@ Page Title="" Language="C#"
  MasterPageFile="~/MasterPage.Master"
  AutoEventWireup="true"
  CodeBehind="Practice.aspx.cs"
```

```
  Inherits="NIT.Practice"
%>

<asp:Content ID="Content1"
  ContentPlaceHolderID="head" runat="server">
</asp:Content>

<asp:Content ID="Content2"
  ContentPlaceHolderID="ContentPlaceHolderMain"
  runat="Server">
  <object classid="clsid:D27CDB6E-AE6D-11cf-
    96B8-444553540000"
    codebase="http://download.macromedia.com/
    pub/shockwave/cabs/flash/swflash.cab#
    version=9,0,0,0" width="610" height="103">
    <param name="movie" value="flash/flash2.swf" />
    <param name="quality" value="best" />
    <param name="menu" value="true" />
    <param name="allowScriptAccess"
  value="sameDomain" />
  <embed src="flash/flash2.swf" quality="best"
  menu="true" width="610" height="103"
  type="application/x-shockwave-flash"
  pluginspage="http://www.macromedia.com/go/
  getflashplayer" allowscriptaccess="sameDomain" />
  </object>
  <h2>Лабораторний курс</h2>
  <p>
    <asp:HyperLink ID="HyperLink1" runat="server"
      NavigateUrl="files/Лабораторна робота №1.pdf"
      CssClass="hyperlink">Лабораторна робота №1
    </asp:HyperLink>
    "Аналіз інформаційних потоків підприємства та
    оцінка інформаційного навантаження фахівців".
  </p>
  <p>
    <asp:HyperLink ID="HyperLink2" runat="server"
      NavigateUrl="files/Лабораторна робота №2.pdf"
```

```
    CssClass="hyperlink">Лабораторна робота №2
  </asp:HyperLink>
  "Моделі оптимального розміщення файлів
  розподіленої бази даних по вузлах обчислювальної
  мережі".
</p>
<p>
  <asp:HyperLink ID="HyperLink3" runat="server"
    NavigateUrl="files/Лабораторна робота №3.pdf"
    CssClass="hyperlink">Лабораторна робота №3
  </asp:HyperLink>
  "Основи розробки мережних інформаційних систем
  засобами мови C# на платформі .NET Framework".
</p>
<p>
  <asp:HyperLink ID="HyperLink4" runat="server"
    NavigateUrl="files/Лабораторна робота №4.pdf"
    CssClass="hyperlink">Лабораторна робота №4
  </asp:HyperLink>
  "Об’єктно-орієнтований підхід до створення
  мережних інформаційних систем засобами мови C# на
  платформі Microsoft .NET Framework".
</p>
<p>
  <asp:HyperLink ID="HyperLink5" runat="server"
    NavigateUrl="files/Лабораторна робота №5.pdf"
    CssClass="hyperlink">Лабораторна робота №5
  </asp:HyperLink>
  "Програмування мережних додатків на мові C#
  з використанням протоколів TCP та UDP".
</p>
<p>
  <asp:HyperLink ID="HyperLink6" runat="server"
    NavigateUrl="files/Лабораторна робота №6.pdf"
    CssClass="hyperlink">Лабораторна робота №6
  </asp:HyperLink>
  "Технологія доступу до баз даних засобами ADO.NET".
</p>
```

```
  <p>
    <asp:HyperLink ID="HyperLink7" runat="server"
      NavigateUrl="files/Лабораторна робота №7.pdf"
      CssClass="hyperlink">Лабораторна робота №7
    </asp:HyperLink>
    "Розробка Web-клієнта мережної інформаційної
    системи на базі технології ASP.NET".
  </p>
</asp:Content>

<asp:Content ID="Content3"
ContentPlaceHolderID="ContentPlaceHolderBar"
runat="Server">
  <h2>Література</h2>
  <asp:HyperLink ID="HyperLink8" runat="server"
  NavigateUrl="http://www.ozon.ru/context/detail/id/5
  602592/" ImageUrl="images/books/1.jpeg"
  ImageWidth="100px"
  CssClass="books"></asp:HyperLink>
  <asp:HyperLink ID="HyperLink9" runat="server"
  NavigateUrl="http://www.ozon.ru/context/detail/id/6
  279214/" ImageUrl="images/books/2.jpeg"
  ImageWidth="100px"
  CssClass="books"></asp:HyperLink>
  <asp:HyperLink ID="HyperLink10" runat="server"
  NavigateUrl="http://www.amazon.com/Illustrated-C-
  2012-Daniel-Solis/dp/1430242787"
  ImageUrl="images/books/3.jpeg" ImageWidth="100px"
  CssClass="books"></asp:HyperLink>
  <asp:HyperLink ID="HyperLink11" runat="server"
  NavigateUrl="http://www.ozon.ru/context/detail/id/5
  538877/" ImageUrl="images/books/4.jpeg"
  ImageWidth="100px"
  CssClass="books"></asp:HyperLink>
  <asp:HyperLink ID="HyperLink12" runat="server"
  NavigateUrl="http://www.ozon.ru/context/detail/id/6
  287517/" ImageUrl="images/books/5.jpeg"
```

```
   ImageWidth="100px"
   CssClass="books"></asp:HyperLink>
   <asp:HyperLink ID="HyperLink13" runat="server"
   NavigateUrl="http://www.amazon.com/Beginning-
   Microsoft-Server-2012-Programming/dp/1118102282"
   ImageUrl="images/books/6.jpeg" ImageWidth="100px"
   CssClass="books"></asp:HyperLink>
   </asp:Content>
```

Лістинг файлу Contacts.aspx

```
<%@ Page Title="" Language="C#"
  MasterPageFile="~/MasterPage.Master"
  AutoEventWireup="true"
  CodeBehind="Contacts.aspx.cs"
  Inherits="NIT.Contacts"
%>

<asp:Content ID="Content1"
  ContentPlaceHolderID="head" runat="server">
</asp:Content>

<asp:Content ID="Content2"
  ContentPlaceHolderID="ContentPlaceHolderMain"
  runat="Server">
  <h2>Контакти</h2>
  <p>ДВНЗ "Криворізький національний університет"</p>
  <p>
    <asp:HyperLink ID="HyperLink1" runat="server"
      NavigateUrl="http://www.knu.edu.ua">
      http://www.knu.edu.ua
    </asp:HyperLink>
  </p>
  <p>Кафедра комп'ютерних систем та мереж</p>
  <p>
    <asp:HyperLink ID="HyperLink2" runat="server"
      NavigateUrl="http://www.ksm.knu.edu.ua">
      http://www.ksm.knu.edu.ua</asp:HyperLink>
  </p>
```

```
  <p>Адреса: 50027 м. Кривий Ріг,
     вул. ХХ Партз'їзду, 11</p>
  <p>Телефон (056) 409-17-20</p>
  <p>Email: <a href="mailto:musicvano@gmail.com">
     musicvano@gmail.com</a></p>
</asp:Content>

<asp:Content ID="Content3"
  ContentPlaceHolderID="ContentPlaceHolderBar"
  runat="server">
</asp:Content>
```

Лістинг файлу Style.css

```
body {
    margin: 0px;
    background-image: url(images/back.gif);
    background-repeat: repeat-x;
    font-family: 'Comic Sans MS';
}
#formMain {
    width: 1180px;
    margin: 0px auto;
}
header {
    height: 167px;
    background-image: url(images/header.gif);
}
    header h1 {
        margin: 0px;
        padding: 60px;
        text-align: center;
        color: white;
        letter-spacing: 5px;
        text-shadow: black 0.1em 0.1em 0.2em;
    }
nav {
    height: 58px;
    background-image: url(images/menu.gif);
```

```
    padding-left: 150px;
}
.menuitem {
    padding: 17px 40px;
    color: white;
    text-transform: uppercase;
}
article {
    height: 640px;
    background-image: url(images/container.gif);
    margin: 0px auto;
    padding: 20px 130px;
}
#content {
    float: left;
    width: 610px;
    text-align: justify;
}
    #content h2 {
        color: #1188B5;
    }
#sidebar {
    float: right;
    width: 230px;
    font-size: 14px;
}
footer p {
    color: lightgray;
    text-align: center;
}
.hyperlink {
    text-decoration: none;
}
a:hover {
    text-decoration: underline;
}
.images {
    float: left;
```

```
    margin: 20px;
    margin-right: 40px;
}
.books {
    padding-right: 10px;
}
a img {
    border:none;
}
```

Лістинг файлу Main.js

```
function RndBanner()
{
  var images = [];
  images[0] =
  "<a href='http://www.htmlbook.ru'>
   <img src='images/htmlbook.png' border='0'
   alt='HTML Book'></a>";
  images[1] =
    "<a href='http://www.javascriptkit.com'>
     <img src='images/jksite.gif' border='0'
     alt='Java Script'></a>";
  images[2] =
    "<a href='http://www.techdays.ru/videos/ASP.NET'>
     <img src='images/techdays.png' border='0'
     alt='Techdays'></a>";
  var index =
    Math.floor(Math.random() * images.length);
  document.write(images[index]);
}
```

Тестування роботи сайту можна проводити завдяки вбудованому у Visual Studio серверу IIS (рис. 5.6).

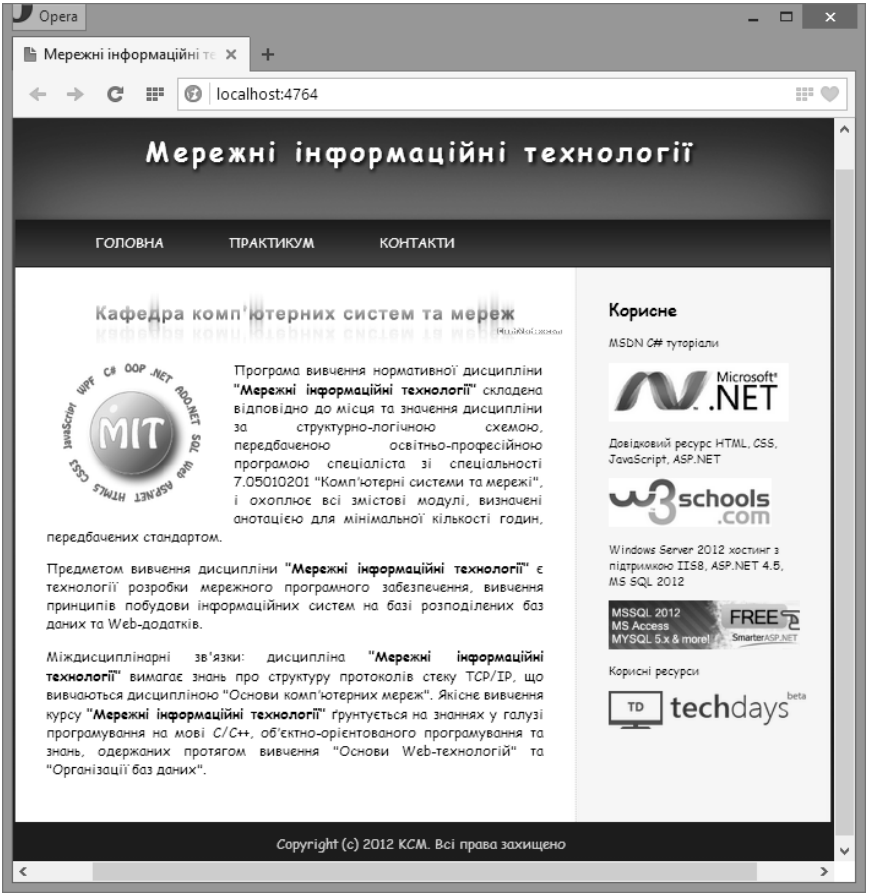


Рис. 5.6. Тестування сайту у браузері

Завдання для самостійного виконання

Відповідно до варіанта та обраного рівня складності виконати такі завдання.

№	Рівень завдань		
	Початковий	Базовий	Високий
1	Відкомпілювати та протестувати аудиторні завдання	Завдання №1	Завдання №1

№	Рівень завдань		
	Початковий	Базовий	Високий
2	–	Завдання №2	Завдання №2
3	–	–	Завдання №3

Завдання №1

Створити Web-сайт на основі готового вільного шаблону з мережі Internet. Сайт повинен містити не менше трьох сторінок ASP.NET, наприклад: головну, детальну інформацію з даної предметної області та контакти. Необхідно реалізувати всі основні компоненти сайту: заголовок (header), меню (nav), основний контент (article) та нижній колонтитул (footer). Таблицю індивідуальних варіантів подано нижче.

Завдання №2

Додатково створити декілька flash-банерів чи використати готові з Web-сайтів мережі Internet. Розмістити дану інформацію на своєму сайті.

Завдання №3

З використанням мови Java Script додатково створити або анімоване меню, або рекламний банер з випадковим контентом.

№ варіанта	Завдання
1	Web-сайт дисципліни «Бази даних»
2	Web-сайт, присвячений спеціальності «Комп'ютерні системи та мережі»
3	Web-сайт, присвячений аналізу сучасного мережного обладнання
4	Web-сайт дисципліни «Програмування на мові C#»
5	Web-сайт дисципліни «Web-програмування»
6	Web-сайт «Мобільні операційні системи»

№ варіанта	Завдання
7	Web-сайт «Технології розробки програмного забезпечення»
8	Web-сайт «Перспективи IT-ринку праці»
9	Web-сайт «Дистанційна освіта та технології створення віртуальних навчальних закладів»
10	Web-сайт «Приховані загрози соціальних мереж Vkontakte, Odnoklassniki, Twitter»
11	Web-сайт, присвячений сучасним технологіям супутникової навігації
12	Web-сайт дисципліни «Паралельні та розподілені обчислення»
13	Web-сайт розробника програмного забезпечення
14	Web-сайт «Сучасні багатопроцесорні системи»
15	Web-сайт факультету інформаційних технологій
16	Web-сайт, присвячений адмініструванню операційної системи Ubuntu (Linux, CentOS, Debian тощо)
17	Web-сайт «Технології та засоби проектування комп'ютерних систем та мереж»
18	Web-сайт «Аналіз IT-ринку мобільних пристроїв, програмного забезпечення та комп'ютерних ігор»
19	Web-сайт, присвячений опису P2P та Torrent-протоколів
20	Web-сайт, присвячений адмініструванню операційної системи Windows Server
21	Web-сайт «Сучасні пристрої зберігання електронної інформації»
22	Web-сайт дисципліни «Комп'ютерні мережі»
23	Web-сайт, присвячений аналізу засобів шифрування та збереження конфіденційності інформації
24	Web-сайт дисципліни «Комп'ютерні системи»
25	Web-сайт «Оптоволоконні технологій передачі даних»

№ варіанта	Завдання
26	Web-сайт, присвячений огляду технологій мобільного зв’язку
27	Web-сайт «Високопродуктивна технологія паралельних обчислень CUDA»
28	Web-сайт дисципліни «Комп’ютерні системи штучного інтелекту»
29	Web-сайт дисципліни «Комп’ютерний дизайн»
30	Web-сайт «Техніка безпеки у галузі інформаційних технологій»

Контрольні питання

1. Коротко охарактеризуйте технологію ASP.NET.
2. Які основні теги мови HTML присутні майже на кожному сайті?
3. У чому полягає сутність підходу Web 2.0 до створення Internet-ресурсів?
4. Призначення та можливості каскадних таблиць стилів CSS.
5. Коротко охарактеризуйте ASP.NET MVC.
6. Навіщо потрібна сторінка MasterPage?
7. Як прив’язати стиль оформлення CSS до компонента ASP.NET?
8. Як додати обробник події на мові C# до компонента ASP.NET?
9. Коротко опишіть процедуру налаштування хостингу на основі Microsoft Windows Server.
10. Переваги та недоліки ASP.NET WebForms у порівнянні з шаблоном MVC.

ТЕМА 6. ОСНОВИ ПОБУДОВИ РОЗПОДІЛЕНИХ ІНФОРМАЦІЙНИХ СИСТЕМ ІЗ ЗАСТОСУВАННЯМ ТЕХНОЛОГІЇ WINDOWS COMMUNICATION FOUNDATION

Мета: розробити службу WCF, яка надає певні методи обчислень для віддалених клієнтських додатків.

Поняття технології WCF

Windows Communication Foundation (WCF) – це платформа для побудови сервісноорієнтованих додатків. За допомогою WCF можна відправляти дані у вигляді асинхронних повідомлень від однієї кінцевої точки служби до іншої. Кінцева точка служби може входити до постійно доступної служби, яка розміщується в IIS, або представляти службу, яка розміщується у додатку. Кінцева точка може бути клієнтом служби, яка виконує запит даних від кінцевої точки служби. Повідомлення можуть представляти один символ чи одне слово, яке відправляється у форматі XML, або мати вигляд складного потоку двійкових даних. WCF-системи можуть бути впровадженні у декількох найбільш поширених сценаріях [17, 24, 36].

1. Захищена служба для обробки бізнес-транзакцій.
2. Служба, що передає іншим об’єктам поточні дані, такі як звіт про трафік або інша служба спостереження.
3. Служба бесід, яка дозволяє двом користувачам спілкуватися і обмінюватися даними у реальному часі.
4. Додаток панелі моніторингу, який опитує одну або кілька служб і дає логічне представлення отриманих даних.
5. Надання доступу до робочого процесу, реалізованому за допомогою Windows Workflow Foundation у вигляді служби WCF.
6. Додаток Silverlight для запиту даних з використанням служб.

Такі програми можна було створювати і до появи WCF, однак WCF істотно спрощує розробку кінцевих точок. Таким чином, платформа WCF реалізує керований підхід до створення веб-служб і клієнтів веб-служб.

Головні особливості WCF

У WCF входить такий набір можливостей.

Сервіс-орієнтованість. Застосування стандартів WS в WCF дозволяє створювати сервісноорієнтовані додатки. Сервісноорієнтована архітектура (SOA) передбачає застосування веб-служб для відправки та отримання даних. Загальною перевагою служб є слабка зв'язаність замість жорсткої запрограмованості для різних додатків. Слабкий зв'язок означає, що будь-який клієнт, створений на будь-якій платформі, може підключатися до довільної служби за умови, що виконуються необхідні контракти.

Взаємодія. WCF реалізує сучасні галузеві стандарти для сумісності з веб-службою.

Кілька шаблонів повідомлень. Обмін повідомленнями виконується по одному з декількох шаблонів. Найчастіше використовується шаблон «запит-відповідь», коли одна кінцева точка запитує дані від іншої кінцевої точки. Друга кінцева точка відповідає. Існують й інші шаблони, наприклад одностороннє повідомлення, коли одна кінцева точка відправляє повідомлення, не чекаючи відповіді. Більш складним є шаблон дуплексного обміну, коли дві кінцеві точки встановлюють з'єднання і відправляють дані в прямому і зворотному напрямках, подібно до програми обміну миттєвими повідомленнями.

Метадані служби. WCF підтримує публікацію метаданих служби з використанням форматів, зазначених у галузевих стандартах, таких як WSDL, схемах XML і WS-Policy. За допомогою таких метаданих можна автоматично створювати і настроювати клієнти для доступу до служб WCF. Метадані можуть публікуватися через HTTP і HTTPS або з використанням стандарту обміну метаданими веб-служб.

Контракти даних. Оскільки платформа WCF побудована на основі. NET Framework, до неї входить набір зручних методів передачі контрактів. Одним з універсальних типів контрактів є контракт даних. Якщо код служби створюється на мові Visual C# або Visual Basic, то найпростішим способом обробки даних фактично є створення класів, які представляють сутність даних з властивостями, що належать сутності даних. WCF включає складну систему для роботи з даними цим зручним способом. Після створення класів, що представляють дані, служба автоматично створює метадані, які дозволяють клієнтам забезпечувати відповідність заданим типам даних.

Безпека. Повідомлення можна шифрувати для захисту конфіденційності та вимагати від користувачів проходити перевірку автентичності перед прийомом повідомлень. Можна реалізувати широко відомі стандарти безпеки, такі як SSL і WS-SecureConversation.

Кілька транспортних протоколів і методів кодувань. Повідомлення можуть відправлятися будь-яким з декількох вбудованих транспортних протоколів з різними методами кодування. Найпоширенішим варіантом є передача повідомлень SOAP у текстовому вигляді з використанням протоколу HTTP. Крім того, WCF дозволяє відправляти повідомлення за допомогою протоколу TCP через іменовані канали або MSMQ. Повідомлення можна кодувати у вигляді тексту або використовувати оптимізований двійковий формат. Двійкові дані можна ефективно відправляти з використанням стандарту MTOM. Якщо жоден з наданих транспортів і кодувань не підходить, ви можете створити власний користувальницький транспорт або метод кодування.

Транзакції. WCF також підтримує транзакції відповідно до однієї з трьох моделей: WS-Atomic Ttransactions, API-інтерфейси простору імен System.Transactions і координатор розподілених транзакцій (Майкрософт).

Підтримка AJAX і REST. REST – це приклад розвитку технології Web 2.0. WCF можна налаштувати для обробки «звичайних» XML-даних, які не є упакованими у конверт

протоколу SOAP. WCF також можна розширити для підтримки певних форматів XML, таких як ATOM (поширений стандарт RSS), і навіть форматів, відмінних від XML, таких як нотація об'єктів JavaScript (JSON).

Розширюваність. Архітектура WCF передбачає ряд точок для розширення. Якщо потрібні додаткові можливості, даною технологією підтримуються точки входу, за допомогою яких можна налаштувати поведінку служби.

Типова архітектура розподілених WCF-систем

Для побудови розподілених систем WCF зазвичай створюються три такі взаємопов'язані збірки:

– *збірка служби WCF*. Ця бібліотека *.dll містить класи та інтерфейси, які забезпечують загальну функціональність, котрою користуються клієнти;

– *хост служби WCF*. Цей програмний модуль містить у собі збірку служби WCF і виконує роль сервера у розподіленій системі;

– *клієнт WCF*. Це додаток, який звертається до функціональності служби через проміжний проксі.

На рисунку 6.1 показано зв'язок між цими збірками. «За лаштунками» використовується багато низькорівневих деталей, внутрішні механізми передачі даних, налаштування транспортного рівня (фабрики, канали, слухачі).

Слід відзначити, що застосування файлу *.config серверною та клієнтською стороною не є обов'язковим. За бажанням можна жорстко закодувати хост та клієнт, вказавши необхідні деталі (кінцеві точки, прив'язки, адреси). Очевидна проблема такого підходу полягає в тому, що, якщо знадобиться змінити деталі налаштування, то потрібно вносити зміни в код, проводити перекомпіляцію і наново розміщувати збірки. Використання файлів *.config робить кодову базу набагато гнучкішою.

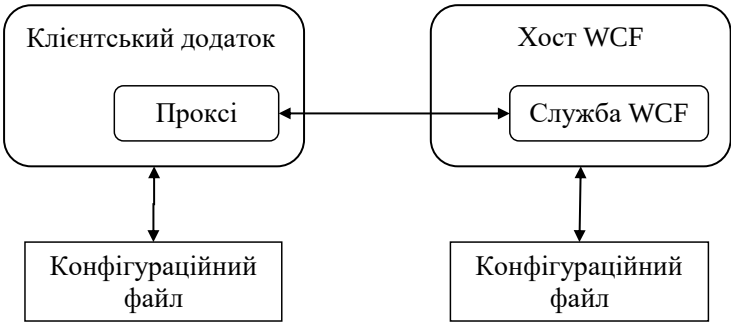


Рис. 6.1. Високорівневе представлення типового WCF-додатка

Хости і клієнти взаємодіють один з одним з використанням адрес, прив'язок та контрактів (`address`, `binding`, `contract`):

– *адреса* описує місцезнаходження служби, в коді представлена типом `System.Uri`, проте її значення зазвичай зберігається у файлах *.config;

– *прив'язка*. Інфраструктура WCF надає багато різних прив'язок, які зазначають мережні протоколи, механізми кодування та транспортний рівень;

– *контракт* надає опис кожного методу, який відкритий зі служби WCF.

Приклад виконання завдання

Завдання №1

Побудувати розподілену систему WCF – гру «Магічний шар». Декілька користувачів підключаються до серверного додатка, задають питання та отримують на них одну випадкову відповідь, наприклад, «так», «ні», «можливо» тощо. Клієнтський додаток розробити засобами Windows Forms.

Розв’язання
Побудова служби WCF

Створимо збірку MagicBallService, яка буде надавати певний функціонал через WCF.

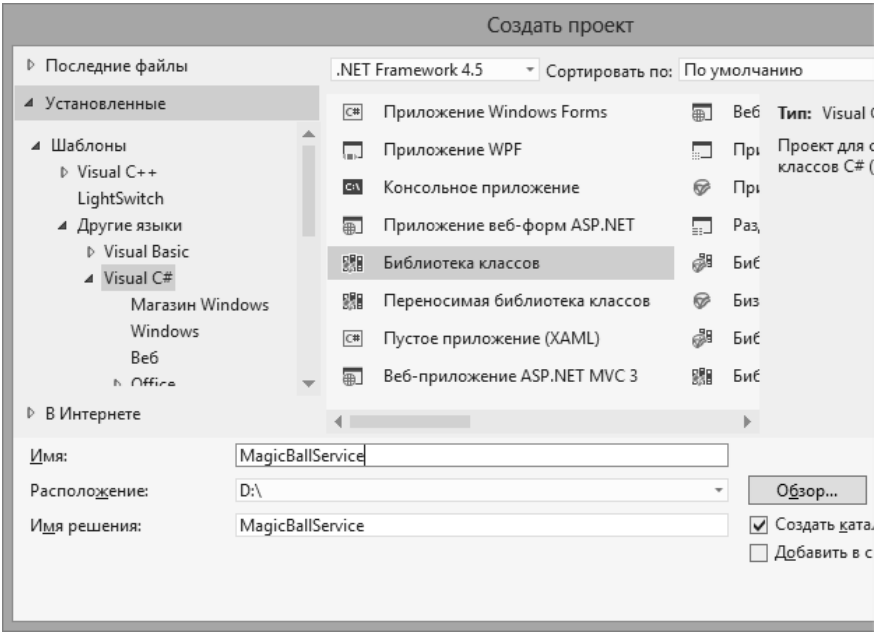


Рис. 6.2. Створення DLL-збірки

У створеному проекті слід видалити файл *Class1.cs* та створити свій власний. З метою більшої гнучкості доцільно функціонал WCF-служби подати у вигляді інтерфейсу `IMagicBall` та класу `MagicBall`, який реалізує даний інтерфейс.

Для відкриття методів інтерфейсу через WCF-службу необхідно додати посилання на збірку `System.ServiceModel` (рис. 6.3, 6.4) та позначити методи атрибутами `[ServiceContract]`, `[OperationContract]`.

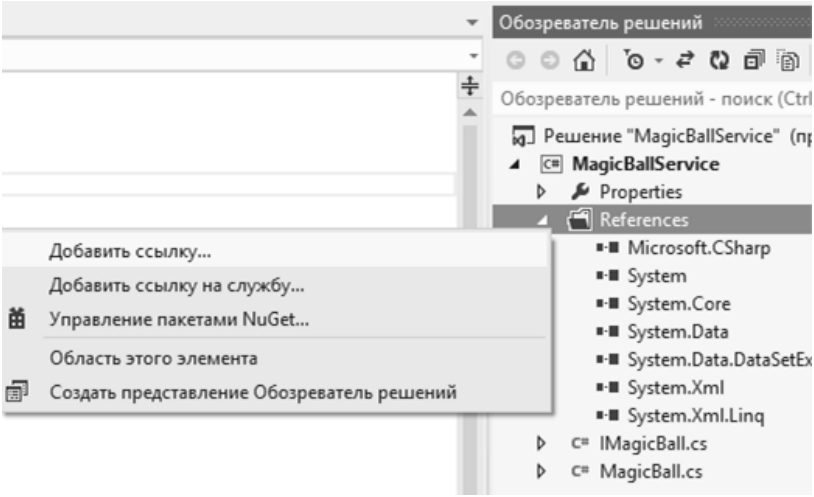


Рис. 6.3. Додавання посилання на збірку

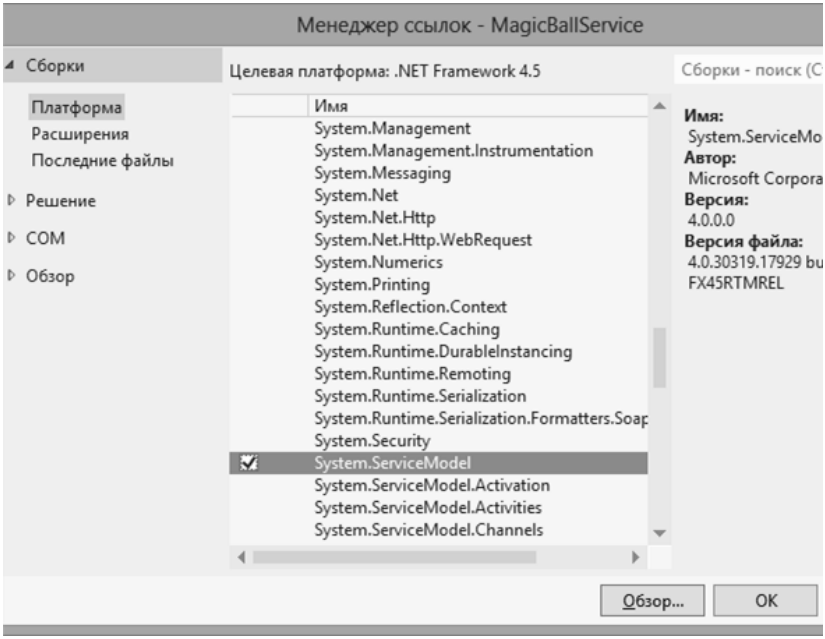


Рис. 6.4. Вибір бібліотеки System.ServiceModel.dll

Вміст створеного класу та інтерфейсу подано нижче.

Код інтерфейсу IMagicBall:

```
using System.ServiceModel;
namespace MagicBallService
{
    [ServiceContract]
    interface IMagicBall
    {
        [OperationContract]
        string GetAnswer(string userQuestion);
    }
}
```

Код класу MagicBall:

```
using System;
namespace MagicBallService
{
  public class MagicBall: IMagicBall
  {
    // для отображения на хосте
    public MagicBall()
    {
      Console.WriteLine("MagicBall ожидает вопросы
        от пользователей...");
    }
    // реализация интерфейсного метода
    public string GetAnswer(string userQuestion)
    {
      string[] answers = { "Будущее неопределенно",
        "Да", "Нет", "Возможно",
        "Задайте вопрос позже", "Определенно" };
      // Вернуть случайный ответ
      Random rnd = new Random();
      return answers[rnd.Next(answers.Length)];
    }
  }
}
```

Після компіляції збірки служби переходимо до створення серверного хосту, де дана збірка буде розміщена.

Хостинг служби WCF

Тепер все готово для визначення хосту. Зазвичай служба промислового рівня повинна розміщуватися в службі Windows або у віртуальному каталозі IIS, проте для спрощення реалізації виконаємо хостинг служби WCF у звичайному консольному додатку. Для цього до відкритого рішення, яке було створене на попередньому кроці, додаємо новий проект `MagicBallHost`. Обов'язково слід не забути додати посилання на збірки System.ServiceModel.dll та MagicBallService.dll (рис. 6.5).

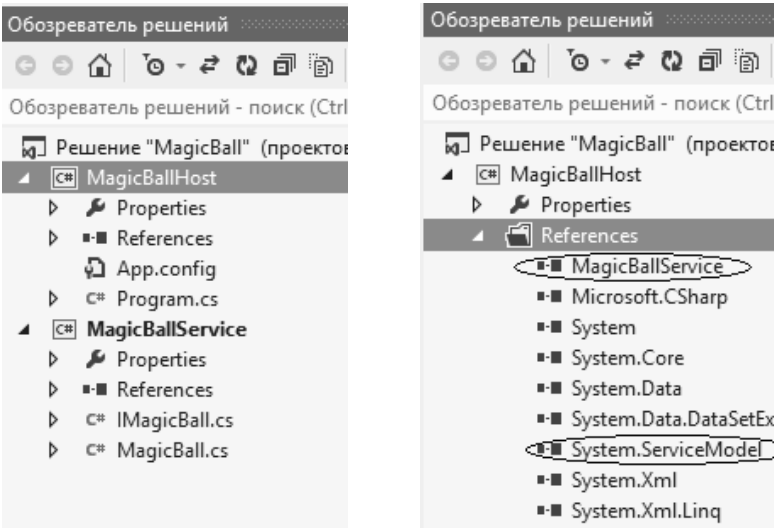


Рис. 6.5. Додавання до рішення нового проекту MagicBallHost та посилань на збірки

Програмний код MagicBallHost:

```
using System;
using System.ServiceModel;
using MagicBallService;
namespace MagicBallHost
```

```
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Console Based WCF Host");
      using (ServiceHost serviceHost =
             new ServiceHost(typeof(MagicBall)))
      {
        // Открыть хост на прослушивание
        // входных сообщений
        serviceHost.Open();
        // Оставить службу функционируюшей до
        // тех пор, пока не будет нажата
        // клавиша Enter
        Console.WriteLine("The service is ready");
        Console.WriteLine("Press Enter to terminate
                           service");
        Console.ReadLine();
      }
    }
  }
}
```

У файлі App.config слід додати налаштування служби WCF, задавши адресу, протокол, контракт, поведінку при HTTP-запитах, дозволити передачу метаданих. У лістингу, поданому нижче, курсивом виділено фрагмент, який потрібно додати до конфігураційного файлу.

Лістинг файлу App.config:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0"
     sku=".NETFramework,Version=v4.5" />
  </startup>
  <system.serviceModel>
```

```
    <services>
      <service name="MagicBallService.MagicBall"
        behaviorConfiguration=
       "MagicBallServiceMexBehavior">
        <endpoint address="http://localhost:8000/
          MagicBallService"
          binding="basicHttpBinding"
          contract="MagicBallService.IMagicBall">
        </endpoint>
        <endpoint address="mex"
           binding="mexHttpBinding"
           contract="IMetadataExchange">
        </endpoint>
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8000/
            MagicBallService"/>
          </baseAddresses>
        </host>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="MagicBallServiceMexBehavior">
          <serviceMetadata httpGetEnabled="true"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Для того, щоб запустити на виконання консольний проект, його потрібно призначити головним (рис. 6.6). Крім того, середовище Visual Studio необхідно запустити з правами адміністратора.

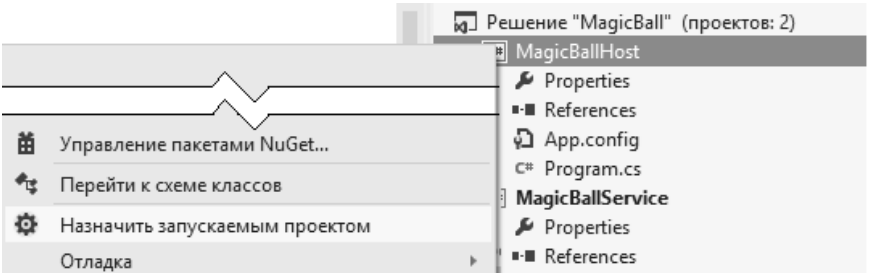


Рис. 6.6. Призначення проекту на виконання MagicBallHost

Після запуску хост-додатка можна перевірити працездатність служби WCF за допомогою браузера (рис. 6.7).

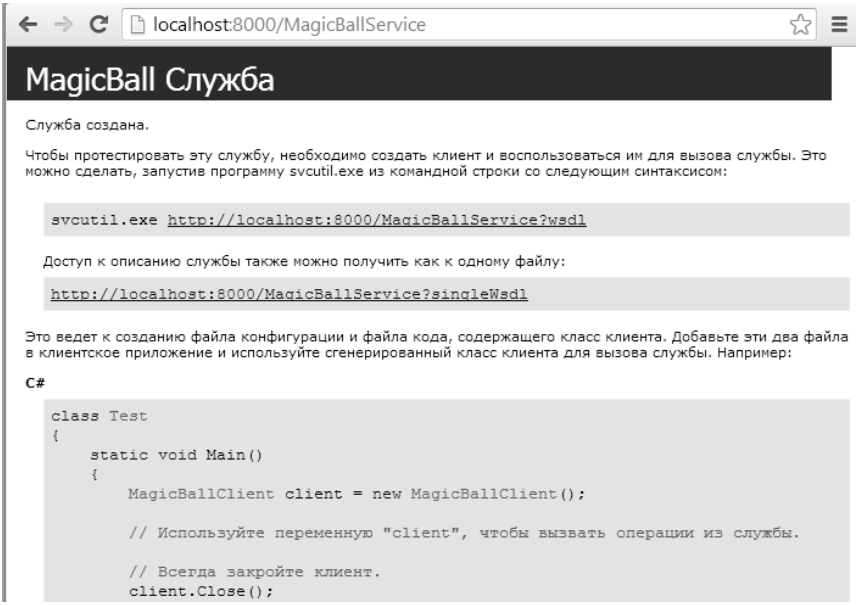


Рис. 6.7. Перегляд метаданих служби
за допомогою браузера

Побудова клієнтського додатка

Клієнтом служби WCF може бути будь-який додаток (консольний, WinForms, WPF, веб-додаток тощо). У даному прикладі створимо Windows Forms клієнт з графічним інтерфейсом. До рішення слід додати новий проект MagicBallClient та призначити його головним для виконання.

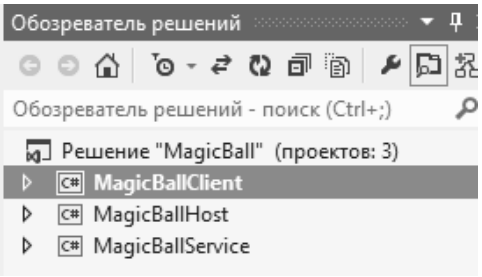


Рис. 6.8. Структура рішення MagicBall зі службою, хостом та клієнтом

Для використання можливостей WCF-служби у клієнтському додатку потрібно спершу запустити хост служби (файл MagicBallHost.exe), потім згенерувати у поточний проект код проксі за допомогою пункту меню Visual Studio «Проект / Добавить ссылку на службу…» (рис. 6.9).

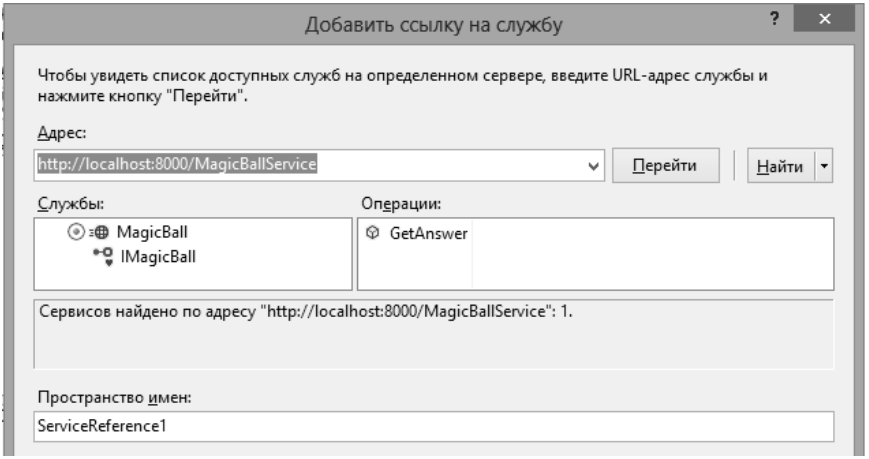


Рис. 6.9. Генерування проксі засобами Visual Studio

Проектування форми клієнтського додатку проводиться в конструкторі (рис. 6.10).

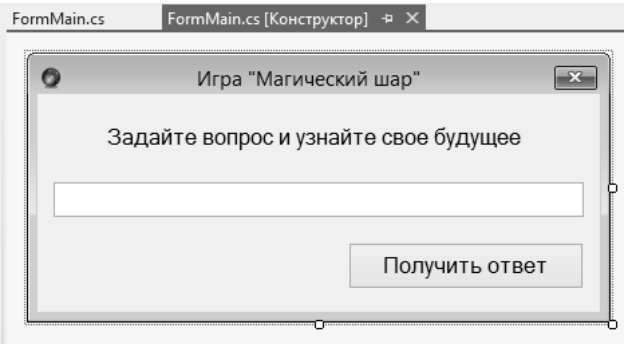


Рис 6.10. Проектування форми клієнтського додатку

Програмний код клієнта представлено нижче.

```
using System;
using System.Windows.Forms;
namespace MagicBallClient
{
  public partial class FormMain : Form
  {
    public FormMain()
    {
      InitializeComponent();
    }
    private void buttonAsk_Click(object sender,
                                 EventArgs e)
    {
      using (ServiceReference1.MagicBallClient ball =
             new ServiceReference1.MagicBallClient())
      {
        string answer =
          ball.GetAnswer(textBoxQuestion.Text);
        MessageBox.Show(answer, "Ответ");
        textBoxQuestion.Clear();
}}}}
```

Завдання для самостійного виконання

Відповідно до варіанта та обраного рівня складності виконати такі завдання.

Рівень завдань		
Початковий	Базовий	Високий
Відкомпілювати та протестувати аудиторні завдання	Завдання №1	Завдання №2

Завдання №1

Розробити WCF-службу у вигляді DLL-бібліотеки, яка виконує певну задачу згідно з індивідуальним варіантом. Розмістити WCF-службу у серверному консольному додатку. Створити клієнтський консольний додаток, який використовує функціонал WCF-служби та передає для обчислення введений користувачем масив.

№ варіанта	Завдання
1	Обчислити суму всіх елементів масиву
2	Визначити кількість парних елементів масиву
3	Обчислити добуток усіх елементів масиву
4	Визначити середнє геометричне значення елементів масиву
5	Визначити кількість елементів масиву, які менші 10
6	Визначити кількість елементів масиву, які більші 12
7	Визначити кількість від’ємних елементів масиву
8	Визначити середнє арифметичне значення елементів масиву
9	Визначити кількість елементів, які менші середнього арифметичного значення цього масиву
10	Визначити перший елемент в масиві, який ділиться націло на 5 або на 3

№ варіанта	Завдання
11	Визначити кількість елементів масиву, які більші 25 і менші 50
12	Визначити кількість додатних елементів масиву
13	Визначити кількість додатних непарних чисел в масиві
14	Обчислити суму непарних елементів масиву
15	Визначити кількість парних елементів масиву, які менші 15
16	Обчислити добуток усіх елементів масиву, які націло діляться на 3
17	Визначити середнє геометричне значення елементів масиву
18	Визначити кількість елементів масиву, які менші 53
19	Визначити кількість елементів масиву, які більші 28
20	Визначити кількість від'ємних елементів масиву, які менші 30
21	Визначити середнє арифметичне значення додатних елементів масиву
22	Визначити кількість елементів, які більші середнього арифметичного значення цього масиву
23	Визначити перший елемент в масиві, який ділиться націло на 6 або на 7
24	Визначити кількість елементів масиву, які більші 5 і менші 20
25	Визначити кількість від'ємних елементів масиву, які більші 70
26	Визначити кількість додатних непарних чисел в масиві, які менші 50
27	Визначити кількість парних елементів масиву, які менші 15

№ варіанта	Завдання
28	Обчислити добуток усіх елементів масиву, які націло діляться на 17
29	Визначити середнє геометричне значення елементів масиву
30	Визначити кількість елементів масиву, які менші 14

Завдання №2

Розробити WCF-службу у вигляді DLL-бібліотеки, яка виконує певну задачу згідно з індивідуальним варіантом. Розмістити WCF-службу у серверному консольному додатку. Створити клієнтський додаток типу Windows Forms, який використовує функціонал WCF-служби.

№ варіанта	Завдання
1	Виконати коригування тексту, видаляючи всі символи #, {}, $. Проте, якщо є запис вартості у вигляді 100$, то такі відомості залишити без змін
2	Виконати коригування тексту, видаляючи зайві знаки пробілів як всередині речення, так і на початку та в кінці
3	Виконати коригування тексту, вставляючи символи пробілу для правильного розділення слів та знаків пунктуації у реченнях. Якщо пробіл вже є, то коригувати непотрібно.
4	Виконати коригування тексту, видаляючи з нього всі знаки дефіс та нулі у числах, які не є значимими, наприклад 15,620
5	Виконати коригування тексту, додаючи до назв валют RUR, UAH, USD їх розшифрування в дужках

№ варіанта	Завдання
6	Визначити кількість слів, які повторюються в тексті більше трьох раз і хоча б два з них знаходяться разом
7	Виконати коригування тексту таким чином, щоб усі числа були виділені знаками «//»
8	Виконати коригування тексту, видаляючи з нього всі зайві коми
9	Виконати коригування тексту таким чином, щоб усі великі літери були замінені на їх номер в алфавіті
10	Виконати коригування тексту, видаляючи з нього всі зайві крапки
11	Виконати коригування тексту таким чином, щоб всі слова починалися з великої літери, а всі інші літери були тільки малими
12	Виконати коригування тексту, видаляючи всі числа, які є номерами телефонів
13	Виконати коригування тексту таким чином, щоб усі цифри в тексті були виділені пробілами
14	Виконати коригування тексту, видаляючи з нього всі зайві крапки, проте крапки і трикрапки в кінці речень видаляти непотрібно.
15	Виконати коригування тексту, видаляючи з нього всю інформацію, записану у дужках, дужки видалити також
16	Створити програму-гру «Вгадай вік», на сервері задається число, клієнт відправляє варіант на сервер, сервер обробляє результат та відсилає «Equal», «More», «Less»
17	Організувати віддалений стек за принципом Last Input First Output (LIFO). Відправляти повідомлення з клієнта на сервер у вигляді «push *value*», де *value* – дійсне число. Команда

№ варіанта	Завдання
	«pop» вилучає з сервера останнє передане число. Команди «add» та «sub» додають та віднімають відповідно два числа, що знаходяться на вершині стеку на сервері, і потім сервер повертає результат клієнту
18	Розробити клієнт, який намагається виконати DDoS-атаку на сервер, посилаючи неперервний потік повідомлень «gettime». У випадку, коли навантаження на сервер не перевищує три повідомлення за секунду, у відповідь на запит «gettime» сервер повертає клієнту поточний локальний час, у зворотному випадку – сервер не відповідає. Розробити клієнт так, щоб можна було промоделювати різну частоту генерування повідомлень
19	Створити програму-чат між клієнтом та сервером
20	Створити клієнт для перегляду файлової структури сервера. На клієнті реалізувати такі операції: «ls» – показати перелік файлів поточного каталогу; «pwd» – показати поточний каталог; «cd..» – піднятися в дереві каталогів у батьківську директорію

Контрольні питання

1. Що таке Windows Communication Foundation?
2. Які можливі прив’язки для транспортного рівня WCF?
3. Які атрибути потрібно використовувати для надання доступу до методів інтерфейсу засобами WCF?
4. Наведіть можливі сценарії застосування технології WCF.
5. Які засоби необхідні для побудови кросплатформних інформаційних систем?

ТЕМА 7. АНАЛІЗ ІНФОРМАЦІЙНИХ ПОТОКІВ ПІДПРИЄМСТВА ТА ОЦІНКА ІНФОРМАЦІЙНОГО НАВАНТАЖЕННЯ ФАХІВЦІВ

Мета: виконати аналіз внутрішніх та зовнішніх інформаційних потоків між підрозділами підприємства, оцінити ступінь інформаційного навантаження фахівців.

Інформаційна система в умовах виробництва

Ефективне управління підприємством будь-якого рівня неможливе без своєчасного аналізу різноманітної інформації (економічної, технологічної, організаційної тощо). Своєчасність надходження такої інформації впливає на формування управлінських рішень і відповідно на результати виробництва. Слід зазначити, що побудова ефективної інформаційної системи (ІС) на підприємстві (підрозділі) – це першочергове завдання керівництва [25].

Побудова інформаційної системи можлива у «класичному» вигляді на основі адміністративно-номенклатурної схеми і паперового документообігу, де інформація існує «на папері» у вигляді наказів, звітів, зведень тощо. Але більш ефективними на сьогодні виявляються ІС, що побудовані на основі використання сучасних інформаційних технологій. Їх безумовними перевагами є значно вища швидкість та надійність передачі інформації, яка досягається за рахунок використання електронних носіїв інформації та корпоративної комп'ютерної мережі.

Розробка сучасної ІС починається з аналізу існуючих інформаційних потоків (документообігу) об'єкта інформатизації (підприємства або окремого підрозділу, цеху, відділу тощо) та містить такі етапи:

- кількісний аналіз організаційно-управляючої інформації;
- оцінка інформаційного навантаження фахівців;
- побудова моделі інформаційних потоків;

– прийняття рішення про необхідність розробки нової (або модернізації існуючої) інформаційної системи на основі сучасних інформаційних технологій.

Кількісний аналіз організаційно-управляючої інформації

У якості узагальнених показників обсягу організаційно-управляючої інформації можуть використовуватися: цифра (наприклад, у звіті) чи слово (у тексті).

Кількість інформації, що несе один узагальнений показник оцінюється за допомогою формули Р. Харлі [29] для незалежних повідомлень однакової ймовірності

$$I = k \log_2 m_1, \qquad (7.1)$$

де I – кількість інформації в повідомленні, біт; k – кількість прийнятих символів у повідомленні; m_1 – число якісних ознак (символів) первинного алфавіту.

На підставі цього розраховується обсяг інформації, що обробляється в системі за певний термін часу. Наприклад, річний обсяг визначається таким виразом

$$H_P = I \cdot Q_P, \qquad (7.2)$$

де H_P – загальний обсяг інформації, що обробляється фахівцями, млн біт/рік; Q_P – кількість узагальнених інформаційних показників, млн показників/рік.

Оцінка інформаційного навантаження фахівців

Оцінка інформаційного навантаження фахівців необхідна насамперед для обґрунтування необхідності створення (модифікації) ІС на основі сучасних інформаційних технологій.

Розрахунок чисельності керуючого персоналу службовців $K_{пс}^{\min}$, мінімально необхідної для нормальної обробки річного обсягу інформації Q_P. З огляду на той факт, що при щоденній восьмигодинній роботі без вихідних і відпусток робочий час складе приблизно $t_{P.P.} = 10^7$ с, значення $K_{пс}^{\min}$ можна визначити, використовуючи таку залежність

$$K_{nc}^{\min} = \frac{H_P}{I_0 \cdot t_{P.P.}}, \qquad (7.3)$$

де I_0 – величина порога сприйняття інформації людиною.

Для досвідченого працівника, фахівця у своїй області I_0 = 2–3 біт/с [13]. Хоча в кожному індивідуальному випадку ця величина піддається різким коливанням, які залежать від інтелекту, кваліфікації, форми представлення інформації, а також від таких факторів, як зацікавленість, складність, утома, стан здоров'я тощо. Для подальших розрахунків можна прийняти максимальне значення I_0 = 3 біт/с.

Коефіцієнт оперативного навантаження (перевантаження) розраховується так

$$K_n = \frac{K_{nc}^{\min}}{K_{nc}^{факт.}}, \qquad (4.4)$$

де $K_{nc}^{факт.}$ – фактична чисельність персоналу службовців в аналізованих відділах підрозділу.

Якщо значення коефіцієнта оперативного навантаження перевищує значення 1, то кажуть про наявність явища «інформаційного бар'єра» (інформаційного перевантаження). Працюючи в режимі інформаційного перевантаження, оператор (фахівець) починає «втрачати» інформацію, що призводить до різкого зростання імовірності ухвалення неправильного рішення. Таким чином, ефективність роботи в таких умовах значно знижується.

Відомо, що найбільш ефективним шляхом вирішення цієї проблеми є створення ІС на основі сучасних інформаційних технологій.

Побудова моделі інформаційних потоків

На підставі даних кількісного аналізу будується модель інформаційних потоків у системі. Графічно дана модель може бути представлена у вигляді зваженого графа, приклад якого наведено на рисунку 7.1.

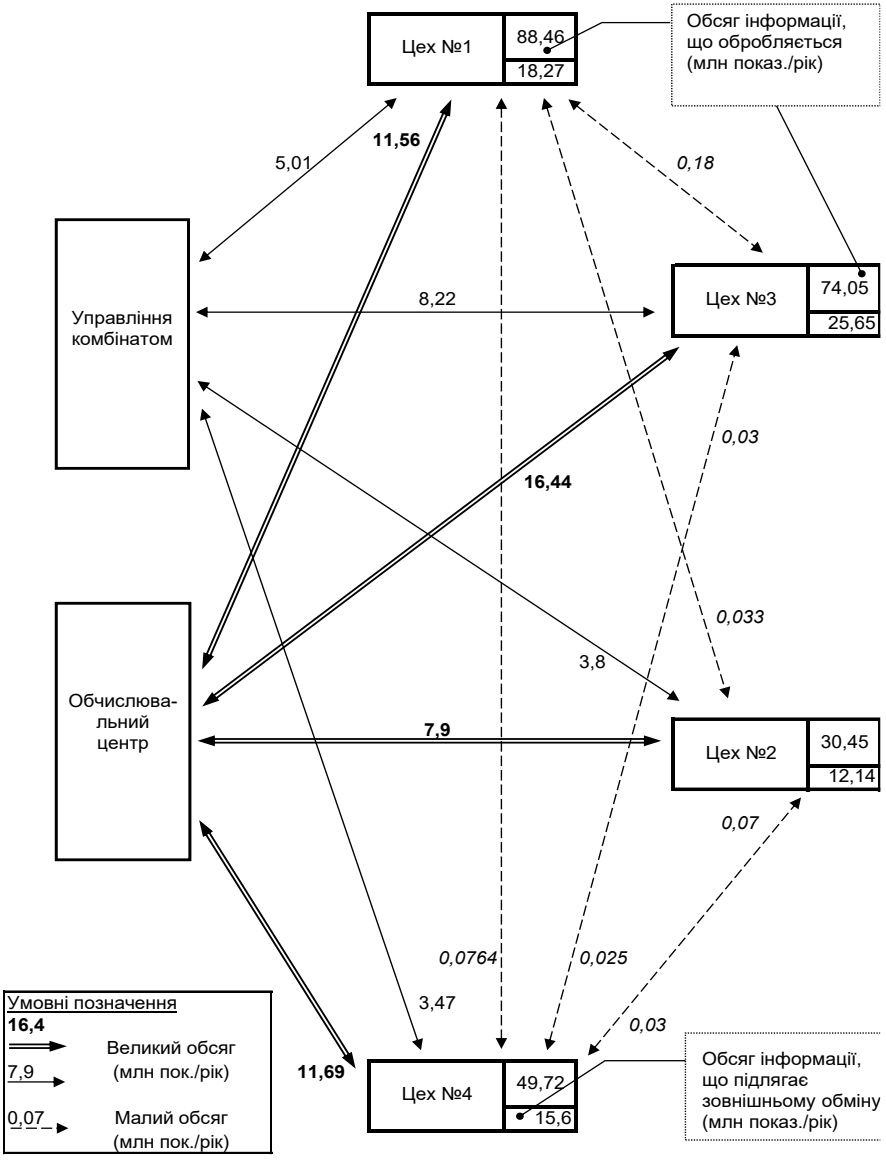


Рис. 7.1. Модель інформаційних потоків підприємства

Вузли графа вказують обсяги внутрішньої і зовнішньої інформації (в узагальнених показниках), а дуги вказують напрямок і обсяг переданої інформації.

Дана модель дозволяє:

– визначити сумарні обсяги інформації, що оброблюються підрозділами підприємства;

– простежити напрямки передачі інформації і виявити найбільш завантажені вузли;

– визначити інтенсивність зовнішніх інформаційних потоків.

Прийняття рішення щодо оптимізації інформаційного навантаження

Виходячи з умов наявності інформаційного перевантаження та реальних обсягів інформації, згідно з моделлю приймається рішення про необхідність розробки нової (або модернізації існуючої) інформаційної системи на основі сучасних інформаційних технологій.

Приклад виконання завдання

Завдання №1

Системотехніками підприємства проведено дослідження зовнішніх і внутрішніх інформаційних потоків окремих підрозділів (цехів) комбінату. Розрахунок обсягів інформації проводився в узагальнених показниках. У таблиці 7.1 зведені результати цих досліджень для цехів №1, 2. На момент збору статистики кількість працюючих фахівців цеху №1 не перевищувала 60 осіб, а цеху №2 – 21 особу.

Необхідно виконати аналіз інформаційних потоків між цехом №1 і цехом №2 підприємства (підсумкові рядки таблиці 7.1).

Таблиця 7.1

Обсяги інформації, яка обробляється та передається між відділами цехів №1, 2

№ з/п	Відділи (підрозділи) комбінату	Кількість показників, що фіксується		
		всього, $\times 10^6$ показ./рік	з них, що підлягають обміну	
			внутрішні, $\times 10^6$ показ./рік	зовнішні, $\times 10^6$ показ./рік
1.	Цех №1			
1.1	Геологічний	3	0,88	1,9
1.2	Праці і ЗП	12	6	2,8
1.3	Головного енергетика	2,25	0,4	0,3
1.4	Головного механіка	2,47	0,09	0,08
1.5	Бухгалтерія	3	0,72	1,7
1.6	Табельна	6,9	3,4	1,6
1.7	Адміністрація	4,1	3,2	0,84
1.8	Технічного контролю	5	3,2	1,5
1.9	Виробничий	8	6,5	6
1.10	Планово-економічний	15	12	1,1
1.11	Технічний	6,74	0,3	0,02
1.12	Маркшейдерський	20	0,64	0,43
	Всього	88,46	37,33	18,27
2.	Цех №2			
2.1	Планово-економічний	4,2	3,6	3,3
2.2	Табельна	4,35	3,5	0,87
2.3	Головного енергетика	1,42	0,28	0,17
2.4	Головного механіка	1,18	0,24	0,06
2.5	Бухгалтерія	1,9	1,52	1,14
2.6	Склади	6,6	5,3	0,8
2.7	Адміністрація	3,4	3,4	0,68
2.8	Диспетчери	7,4	7,4	5,12
	Всього	30,45	25,24	12,14

Розв'язання

1. Кількісний аналіз потоків організаційно-керуючої інформації. З даних таблиці 7.1 відомо, що протягом року фахівцями цеху №1 обробляється обсяг інформації, еквівалентний 88,46 млн узагальнених показників. Як узагальнені показники бралися: цифра у звіті чи слово у тексті. При цьому середньостатистична кількість символів у цифровому показнику вважалася рівною 6 знакам і середній довжині символьного показника (слова) – 5,5 знаків.

Для оцінки нижньої межі для кількості інформації в одному узагальненому показнику за величину k приймаємо середньостатистичну кількість знакомісць для символьного показника (k = 5,5 символів).

Для кодування всіх показників використовується первинний алфавіт довжиною не менш 64 символів. Тобто $m_1 = 64$ символи.

Тоді, відповідно до залежності (7.1), кількість інформації в одному узагальненому показнику складе

$$I = 5{,}5 \log_2 64 = 33 \text{ (біта).}$$

Загальний обсяг інформації, що обробляється фахівцями цеху №1 з розрахунку $Q_P = 2919{,}18$ млн узагальнених показників у рік (таблиця 7.1), складе

$$H_P = I \cdot Q_P = 33 \cdot 88{,}46 = 2919{,}18 \text{ (млн біт/рік).}$$

Аналогічні розрахунки для цеху №2 дають такі результати

$$H_P = I \cdot Q_P = 33 \cdot 30{,}45 = 1004{,}85 \text{ (млн біт/рік).}$$

2. Оцінка інформаційного навантаження фахівців. На підставі отриманої моделі інформаційних потоків може бути зроблений якісний аналіз, оцінене інформаційне навантаження на провідних спеціалістів основних підрозділів комбінату.

Відповідно до розрахунків для цеху №1 $H_P = 2919{,}18$ млн біт/рік $\approx 2{,}92 \cdot 10^9$ біт/рік. Тоді, на підставі залежності (7.3) одержуємо:

$$K_{пс}^{\min} = \frac{2{,}92 \cdot 10^9}{3 \cdot 10^7} \approx 97 \text{ (чол.)}$$

Таким чином, для обробки отриманого обсягу інформації штат службовців 12-ти провідних відділів цеху (табл. 7.1) повинен містити не менш 97 фахівців досить високої кваліфікації. Реально в даних відділах на момент збору статистики кількість працюючих фахівців не перевищувала 60 осіб.

У цьому випадку коефіцієнт оперативного перевантаження складе

$$K_n = \frac{K_{пс}^{\min}}{K_{пс}^{факт.}} = \frac{97}{60} \approx 1{,}62.$$

Аналогічні розрахунки для цеху №2 (фактична кількість працюючих – 21 фахівець) дають такі результати:

H_P = 1004,85 млн біт/рік ≈ 1,004*10^9 біт /рік;

$$K_{пс}^{\min} = \frac{H_P}{I_0 \cdot t_{P.P.}} = \frac{1{,}004 \cdot 10^9}{3 \cdot 10^7} \approx 33;$$

$$K_n = \frac{K_{пс}^{\min}}{K_{пс}^{факт.}} = \frac{33}{21} = 1{,}57.$$

3. Модель інформаційних потоків у системі керування технологічним процесом. На підставі даних таблиці 7.1 і наведених вище розрахунків може бути отримана модель інформаційних потоків у системі. Дана модель представляється у вигляді графа (рис. 7.2).

Таким чином, результати проведених вище розрахунків свідчать, що обсяг інформації, оброблюваний провідними спеціалістами цехів №1 і №2, перевищує припустиме граничне значення відповідно у 1,62 та 1,57 рази. Це говорить про те, що провідні спеціалісти зазначених підрозділів постійно працюють у режимі інформаційного перевантаження. Працюючи в режимі інформаційного перевантаження, оператор починає «втрачати» інформацію, що приводить до різкого зростання імовірності ухвалення неправильного

рішення. Отже, ефективність керування виробництвом при наявності інформаційного бар'єра значно знижується. Для зняття інформаційного бар'єра і підвищення ефективності керування рекомендується розробка інформаційної системи керування між цехами №1 та №2.

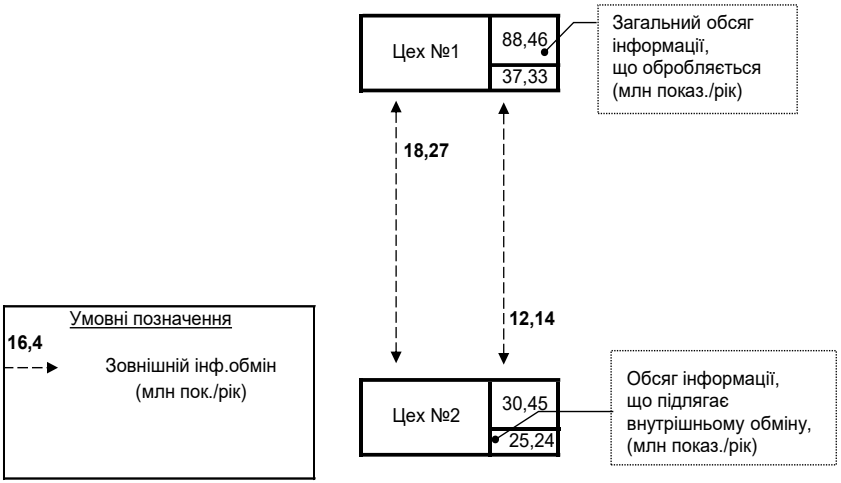


Рис. 7.2. Модель інформаційних потоків підприємства (цехи №1, 2)

Завдання для самостійного виконання

Згідно з варіантом обрати три відділи для відповідного цеху. Провести аналіз інформаційних потоків у даних відділах. Розрахувати інформаційне навантаження фахівців при умові, що у кожному відділі працюють не менш ніж 3 спеціалісти. Побудувати інформаційну модель у графічному вигляді для трьох відділів на підставі прикладу (рис. 7.2). Зробити висновки щодо наявності (відсутності) інформаційного перевантаження та запропонувати засоби його усунення.

№ варіанта	Завдання	
	Цех	Відділи
1	1	1, 2, 3
2	2	1, 2, 3
3	1	4, 5, 6
4	2	4, 5, 6
5	1	7, 8, 9
6	2	1, 7, 8
7	1	10, 11, 12
8	2	2, 4, 6
9	1	2, 4, 6
10	2	1, 3, 7
11	1	8, 10, 12
12	2	3, 5, 8
13	1	1, 3, 7
14	2	3, 5, 7
15	1	3, 5, 8

№ варіанта	Завдання	
	Цех	Відділи
16	1	4, 8, 12
17	2	5, 7, 8
18	1	3, 7, 11
19	2	4, 6, 7
20	1	2, 6, 10
21	2	1, 5, 8
22	1	1, 4, 8
23	2	3, 6, 7
24	1	5, 7, 11
25	2	2, 4, 7
26	1	7, 10, 12
27	2	1, 4, 7
28	1	6, 8, 11
29	2	2, 3, 5
30	1	8, 9, 11

Контрольні питання

1. Що таке інформація та інформаційна система?
2. Які етапи розробки інформаційних систем?
3. Як оцінюється кількість інформації?
4. Що таке узагальнений показник?
5. Як оцінити інформаційне навантаження фахівців?
6. Як розраховується обсяг інформації в системі?
7. Що таке модель інформаційних потоків?
8. Що дозволяє визначити модель інформаційних потоків?
9. На підставі чого приймається рішення про розробку нової інформаційної системи?
10. Як розрахувати необхідну кількість фахівців для обробки певного обсягу інформації?

ТЕМА 8. МОДЕЛЮВАННЯ ОПТИМАЛЬНОГО РОЗМІЩЕННЯ ФАЙЛІВ РОЗПОДІЛЕНОЇ БАЗИ ДАНИХ ВУЗЛАМИ ОБЧИСЛЮВАЛЬНОЇ МЕРЕЖІ

Мета: виконати моделювання та розробити програмне забезпечення для оптимізації розміщення файлів вузлами комп'ютерних мереж з різною топологією.

Математична модель розміщення файлів для мережі із зіркоподібною топологією

У даному розділі розглядається підхід до побудови моделі оптимального розміщення файлів розподіленої бази даних (РБД) в обчислювальній мережі із зіркоподібною топологією. У якості критерію оптимальності прийнятий об'єм даних, який пересилається за одиницю часу в результаті функціонування системи [28].

Розглянемо однорідну обчислювальну мережу, кожен вузол якої складається із ЕОМ та апаратури передачі даних. Припустимо, що всі вузли мережі утворюють зіркоподібну конфігурацію через комутатор даних, як показано на рисунку 8.1, де K_j (j = 1, 2, ..., n) – j-й вузол мережі, що складається із ЕОМ і апаратури передачі даних; B_j (j = 1, 2, ..., n) – локальна база даних, яка міститься на вузлі K_j.

Вважатимемо, що запит будь-якого вузла потребує доступу до певного файлу розподіленої бази даних; усі запити з фіксованого вузла до одного і того ж файлу мають однакову довжину і вимагають для відповіді однаковий об'єм даних; запити обслуговуються у вузлі у порядку їх надходження.

Будемо розрізняти два види запитів, що генеруються вузлами:

– тип 1 – запити, для обробки яких необхідні файли не містяться в базі даних вузла, на який вони надійшли;

– тип 2 – запити, для обслуговування яких потрібні файли містяться в базі даних відповідного вузла.

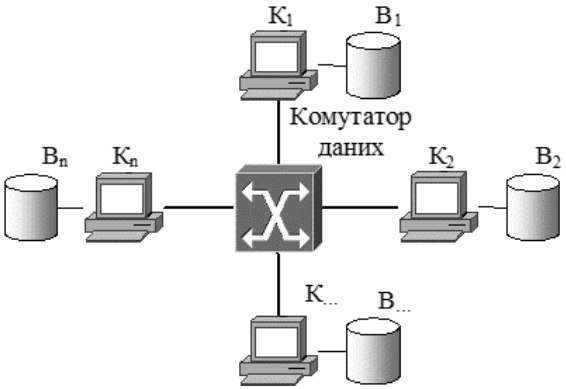


Рис. 8.1. Зіркоподібна топологія мережі

При функціонуванні обчислювальної системи в кожному вузлі утворюється дві вхідні черги повідомлень: перша – із запитів типу 1 і відповідей з інших вузлів на запити цього типу, друга – із запитів типу 2. Запити, що надходять у відповідні вхідні черги вузлів заздалегідь класифікуються. З цією метою використовуються довідники локальних баз даних вузлів. Час класифікації вважається дуже малим, тому він не враховується при побудові моделі. Обробка повідомлень двох вхідних черг кожного вузла здійснюється з використанням такої пріоритетної схеми: повідомлення першої вхідної черги мають вищий пріоритет обслуговування, ніж повідомлення другої вхідної черги. Повідомлення однієї і тієї ж черги обслуговуються в такому порядку: першим прийшов, першим обслуговується (FIFO – First Input First Output).

Розглянемо схему обробки запитів залежно від їх типу.

Якщо запит належить до типу 1, то він згідно з чергою пересилається із вузла в комутатор даних, де приєднується до вхідної черги комутатора. У комутаторі черга утворюється у порядку прибуття повідомлень. Процесор комутатора вибирає повідомлення з черги згідно з порядком: першим прийшов, першим обслуговується. Якщо вибране з черги комутатора повідомлення є запитом, то за довідником комутатора знаходиться вузол, що містить необхідний файл для

обслуговування даного запиту, і запит пересилається в другу вхідну чергу цього вузла (тобто на цьому етапі обслуговування запит типу 1 перетворюється на запит типу 2). Після його обслуговування (згідно з чергою) формується відповідь, якій привласнюється вищий пріоритет. Ця відповідь пересилається у вхідну чергу комутатора даних.

Якщо запит належить до типу 2, то він обслуговується згідно з чергою запитів цього типу і відповідь виводиться на ЕОМ. При цьому, якщо під час обслуговування запиту типу 2 у вхідну чергу вузла надходить повідомлення з вищим пріоритетом обслуговування (запит типу 1 або відповідь на запит цього типу), то обслуговування запиту уривається і негайно починається обслуговування повідомлення з вищим пріоритетом. Обслуговування запиту типу 2 поновлюється тільки в тому випадку, якщо немає більше повідомлень з вищим пріоритетом обслуговування у вхідній черзі вузла.

З описаної схеми обробки запитів у системі стає зрозумілим, що в процесі обслуговування запитів виникають затримки, обумовлені чергами у вузлах, чергою в комутаторі даних і часом обслуговування у вузлах і комутаторі. Загальна середня затримка при цьому залежить від розподілу файлів локальними базами даних. Тому задача полягає в такому розподілі файлів вузлами обчислювальної мережі, при якому загальна середня затримка обробки запитів буде мінімальною. Для знаходження середньої затримки використовується математичний апарат теорії черг.

Оскільки кожен запит залежно від його типу проходить свій шлях через мережу, то розглянемо окремо послідовність затримок, що виникають у мережі при обслуговуванні запитів кожного з двох типів.

1. Припустимо, що для обслуговування запиту, ініційованого вузлом K_i, використовується файл, що міститься в локальній базі даних вузла K_j. Тоді у порядку проходження повідомлення в мережі виникають такі затримки:

а) очікувана затримка у вхідній черзі вузла K_i. Ця затримка обумовлена запитами типу 1 і відповідями на запити

цього типу, що надійшли із вузла K_i і комутатора даних і знаходяться в черзі у момент прибуття даного запиту;

б) затримка, обумовлена часом обслуговування у вузлі K_i. Вказаний час необхідний для пересилки запиту з вузла у вхідну чергу комутатора даних;

в) очікувана затримка в комутаторі даних. Ця затримка є наслідком інших повідомлень, що знаходяться у вхідній черзі комутатора у момент прибуття даного повідомлення;

г) затримка, обумовлена часом обслуговування в комутаторі даних. Цей час необхідний для визначення вузла, що містить потрібний для обслуговування запиту файл, і пересилки запиту в цей вузол;

д) очікувана затримка у вхідній черзі вузла K_j. Запит, що надійшов у вузол K_j, стає в другу вхідну чергу цього вузла, тобто йому привласнюється нижчий рівень пріоритету обслуговування. Тому вказана затримка обумовлена запитами, що знаходилися в другій і першій чергах вузла K_j у момент прибуття даного, і всіма повідомленнями, що надійшли в першу вхідну чергу вузла K_j від часу надходження даного запиту до закінчення його обслуговування;

е) затримка, обумовлена часом обслуговування у вузлі K_j. Цей час необхідний для формування відповіді на запит, привласнення йому вищого пріоритету і пересилки у вхідну чергу комутатора даних;

ж) очікувана затримка в комутаторі даних. Ця затримка визначається, як у пункті в);

з) затримка, обумовлена часом обслуговування в комутаторі даних. Вона визначається аналогічно як в пункті г) і обумовлена часом, необхідним для визначення вузла призначення відповіді і пересилки відповіді в цей вузол;

и) очікувана затримка у вхідній черзі вузла K_i. Ця затримка визначається, як в пункті а);

к) затримка, обумовлена часом обслуговування у вузлі K_i. Вказаний час необхідний для виводу відповіді на первинний вузол K_i.

2. Якщо для обслуговування запиту, ініційованого на терміналі вузла $К_i$, використовується файл, що міститься в локальній базі даних цього вузла, то в процесі обслуговування запиту виникають такі затримки:

а) очікувана затримка у вхідній черзі вузла K_i. Ця затримка обумовлена обслуговуванням повідомлень з першої вхідної черги вузла K_i, які мають вищий пріоритет, і запитів другої вхідної черги, що знаходяться в черзі до моменту прибуття даного запиту;

б) затримка, обумовлена часом обслуговування у вузлі K_i. Цей час необхідний для формування відповіді на запит і виводу відповіді на первинний вузол.

Для запису виразів, що визначають середню затримку при обслуговуванні запитів кожного з двох типів, введемо такі позначення:

Q_{il} – середній час очікування в l-й (l = 1, 2) вхідній черзі вузла K_i,

R_{il} – середній час обслуговування повідомлення з l-ї (l = 1, 2) вхідної черги процесором ЕОМ вузла K_i,

Q – середній час очікування у вхідній черзі комутатора даних;

R – середній час обслуговування повідомлення процесором комутатора даних;

T_{il} – середній час відповіді на запит l-го типу, ініційованого вузлом K_i.

Тоді

$$T_{i1} = 2Q_{i1} + 2R_{i1} + 2Q + 2R + 2Q_{j2} + 2R_{j2} \quad (8.1)$$

$$T_{i2} = Q_{i2} + R_{i2} \quad (8.2)$$

При цьому враховується, що запит, ініційований вузлом K_i, використовує для свого обслуговування локальну базу даних вузла K_j або K_i, залежно від того, належить запит до типу 1 або типу 2.

Розглянемо достатньо ефективний підхід до побудови моделі оптимального розподілу РБД вузлами обчислювальної мережі із зіркоподібною топологією. В основі цього підходу

лежить застосування у якості критерію оптимальності сумарного потоку запитів типу 1, які надходять до вузлів за одиницю часу. При цьому чим менший сумарний потік таких запитів, тим менший об'єм даних, необхідних для пересилання, а значить, менший середній час реакції системи на запити.

Нехай задані величини:

n – число вузлів обчислювальної мережі;

m – число незалежних файлів, що входять в РБД;

F_j – j-й файл РБД;

K_i – i-й вузол мережі;

λ_i – середня інтенсивність запитів, ініційованих у вузлі K_i;

W_{ik} – середній час обробки запиту k-го (k = 1, 2) типу у вузлі K_i;

p_{ij} – ймовірність того, що для обслуговування запиту, ініційованого у вузлі K_j, необхідний файл p_j.

Введемо позначення:

q_{sr} – ймовірність того, що запит, ініційований у вузлі K_s, використовує для свого обслуговування файл, який знаходиться в локальній базі даних вузла K_r;

λ_{ik} – середня інтенсивність надходження запитів k-го (k = 1, 2) типу у вхідну чергу вузла K_i.

Очевидно,

$$\sum_{j=1}^{m} p_{ij} = 1 \ \ (i = 1,2,...n).$$

Використавши p_{ij}, визначимо множину величин q_{sr} (s, r = 1, 2, …, n), де q_{sr} є вірогідністю того, що запит, ініційований у вузлі K_s, використовує для свого обслуговування файл, який знаходиться в локальній базі даних вузла K_r. Значення q_{sr} залежать від розподілу файлів вузлами обчислювальної мережі і від значень p_{ij}. Якщо ввести змінні x_{ij} (i = 1, 2, ..., m; j = 1, 2, ..., n), де

$$x_{ij} = \begin{cases} 1, \text{файл } F_i \text{ знаходитьс я у вузлі } K_j \\ 0, \text{у зворотному випадку,} \end{cases}$$

то

$$q_{sr} = \sum_{j=1}^{m} p_{sl} x_{lr} \quad (s, r = 1,2,...n) \tag{8.3}$$

причому

$$\sum_{j=1}^{m} q_{sr} = 1 \quad (s = 1,2,...n).$$

Встановимо взаємозалежність між величинами λ_{ik} і λ_i. Очевидно, мають місце такі залежності

$$\lambda_{i1} = 2\lambda_i \left(1 - q_{ii}\right), \tag{8.4}$$

$$\lambda_{i2} = \sum_{s=1}^{n} \lambda_s q_{si}\,, \tag{8.5}$$

де коефіцієнт 2 у формулі (8.4) показує те, що всі повідомлення, які обслуговуються дистанційно, повертаються в той же вузол K_i після закінчення їх обробки в іншому вузлі, $\lambda_s q_{si}$ у формулі (8.5) означають потік повідомлень, ініційованих вузлом K_s, які для свого обслуговування використовують локальну базу даних вузла K_i.

Середню інтенсивність вхідного потоку повідомлень у комутаторі даних можна визначити за формулою

$$\lambda = 2\sum_{i=1}^{n} \lambda_i \left(1 - q_{ii}\right).$$

Добуток $\lambda_i q_{ij}$ визначає інтенсивність запитів до файлу F_j, які надходять з вузла K_i. Тому, якщо файл F_j зберігається у вузлі K_s, то сумарна інтенсивність запитів типу 1, ініційованих усіма вузлами мережі, до файлу F_j визначається формулою

$$Q_{js} = \sum_{\substack{i=1 \\ i \neq s}}^{n} \lambda_i p_{ij}\,. \tag{8.6}$$

З урахуванням Q_{js} (j = 1, 2, …, m; s = 1, 2, …, n) математична модель задачі буде мати вигляд

$$L = \sum_{j=1}^{m} \sum_{s=1}^{n} Q_{js} x_{js} \to \min\,, \tag{8.7}$$

при обмеженнях

$$\sum_{j=1}^{n} x_{ij} = 1, \ (i = 1, 2, \ldots, m) \qquad (8.8)$$

$$\lambda_{k1} W_{k1} + \lambda_{k2} W_{k2} < 1, \ (k = 1, 2, \ldots, n) \qquad (8.9)$$

$$x_{ij} = \{0 \cup 1\}, \ (i = 1, 2, \ldots, m; j = 1, 2, \ldots, n) \qquad (8.10)$$

Для реалізації моделі (8.7) – (8.10) розроблений евристичний алгоритм, який включає два етапи. На першому знаходиться початковий розподіл файлів, який завжди буде оптимальним, якщо не враховувати умову (8.9). На другому проводиться перерозподіл файлів, якщо для початкового розподілення існує хоча б один індекс k такий, що умова (8.9) не виконується. Другий етап складається із ряду кроків, причому на кожному кроці здійснюється перерозподіл одного файлу із переповненого вузла таким чином, щоб досягти мінімального збільшення цільової функції. Виконання другого етапу алгоритму продовжується до тих пір, поки не буде знайдений розподіл, який задовольняє умову (8.9).

І етап. Визначення початкового розподілу.

1. Визначимо значення Q_{js} ($j = 1, 2, \ldots, m$; $s = 1, 2, \ldots, n$) за формулою (2.6). Складаємо матрицю $Q = \{Q_{js}\}_{m, n}$.

2. Знаходимо $\min\limits_{1 \le s \le n} Q_{js}$ для всіх $j = 1, 2, \ldots, m$. Нехай $\min\limits_{1 \le s \le n} Q_{js} = Q_{js_j}$ ($j = 1, 2, \ldots, m$).

3. Визначаємо початковий розподіл, тобто знаходимо матрицю $X = \{x_{js}\}_{m,n}$, де

$$x_{js_j} = 1, \ (j = 1, 2, \ldots, m);$$

$$x_{js} = 0, \ (j = 1, 2, \ldots, m; \ s = 1, 2, \ldots, n; \ (j, s) \neq (j, s_j)).$$

Розподіл X завжди буде оптимальним, якщо не враховувати умову (8.9).

ІІ етап. Перерозподіл файлів.

1. Для всіх $k = 1, 2, \ldots, n$ перевіряємо виконання умови (8.9). Якщо ця умова виконується для всіх k, то на цьому робота алгоритму закінчується. Початковий розподіл X є

оптимальним. Якщо для деякого індексу $k = r$ умова (8.9) не виконується, то переходимо до операції 2.

2. Перерозподіл файлів із переповненого вузла K_r. Для кожного j, для якого $x_{jr} = 1$, знаходимо $\min_k\left(Q_{jk} - Q_{jr}\right)$, де мінімум береться за тими індексами $k \neq r$, для яких вузол K_k відкритий для перерозподілу. Нехай

$$\min_k\left(Q_{jk} - Q_{jr}\right) = Q_{jk_j} - Q_{jr}.$$

Тоді знаходимо $\min_j\left(Q_{jk_j} - Q_{jr}\right)$, де мінімум береться за всіма індексами j, для яких $x_{jr} = 1$. Нехай

$$\min_j\left(Q_{jk_j} - Q_{jr}\right) = Q_{lk_l} - Q_{lr}.$$

Тоді в матриці X покладемо $x_{lr} = 0$, $x_{lk_l} = 1$. Це означає, що файл F_l із вузла K_r перерозподілений у вузол K_{kl}. Такому перерозподілу відповідає мінімальне збільшення значення цільової функції.

3. Для індексу $k = r$ перевіряємо умову (2.9). Якщо вона не виконується, то переходимо до операції 2. Якщо умова виконується, то вузол K_r вважається закритим для перерозподілу і здійснюється перехід до виконання операції 4.

4. Для всіх індексів k, для яких вузол K_k не є зачиненим для перерозподілу файлів, перевіряємо виконання умови (8.9). Якщо для всіх таких індексів k умова виконується, то на цьому робота алгоритму завершується. Розподіл X приймається за оптимальний. Якщо для деякого індексу $k = r$ умова (8.9) не виконується, то переходимо до операції 2.

Таким чином, алгоритм дозволяє за кінцеве число кроків знайти оптимальний або майже оптимальний розподіл файлів вузлами обчислювальної мережі. Для роботи алгоритму повинні бути задані величини m, n, λ_i, W_{ik}, p_{ij} для $i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, m$; $k = 1, 2$. Результатом роботи алгоритму є матриця X.

Математична модель розміщення файлів для мережі із кільцевою топологією

Дано обчислювальну мережу з кільцевою топологією, кожен вузол якої складається з ЕОМ та апаратури передачі даних. Усі вузли мережі зв'язані між собою єдиним каналом передачі даних у вигляді замкненої лінії, як показано на рисунку 8.2, де K_j (j = 1, 2, ..., n) – j-й вузол мережі, що складається з ЕОМ та апаратури передачі даних; B_j (j = 1, 2, …, n) – локальна база даних, розташована в пристроях ЕОМ вузла K_j, що запам'ятовують.

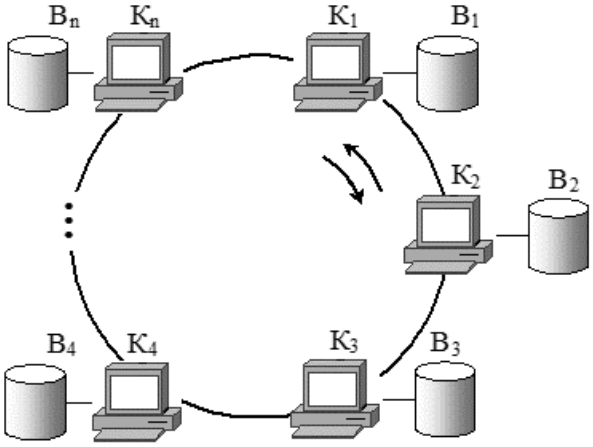


Рис. 8.2. Обчислювальна мережа з кільцевою топологією

Припустимо, що запит, який надходить на будь-який вузол, потребує доступу до певного файлу розподіленої бази даних (РБД) і всі запити з фіксованого вузла до одного і того ж файлу мають однакову довжину і вимагають для відповіді однакового об'єму даних. Останнє припущення не є суттєвим, оскільки можна розглянути різні типи запитів. Опишемо одну із можливих схем обробки запитів [28].

Всі запити, що надходять до вузлів, умовно можна розділити на два типи. Вважатимемо, що запит, ініційований

вузлом, належить до першого або другого типу запитів залежно від того, не міститься або міститься потрібний файл в локальній базі даних відповідного вузла.

При функціонуванні обчислювальної системи в кожному вузлі утворюються дві вхідні черги повідомлень, що вимагають обслуговування: перша – із запитів першого типу та відповідей на запити цього типу, друга – із запитів другого типу. Запити, що надходять від вузлів у відповідні вхідні черги, заздалегідь класифікуються. З цією метою використовуються довідники локальних баз даних вузлів. Обробка повідомлень двох вхідних черг кожного вузла здійснюється з використанням такої пріоритетної схеми: повідомлення першої вхідної черги мають вищий пріоритет обслуговування, ніж повідомлення другої вхідної черги. Повідомлення однієї і тієї ж черги обслуговуються згідно з порядком: першим прийшов, першим обслуговується (FIFO – First Input First Output).

Якщо запит належить до першого типу, то він згідно з чергою пересилається з даного вузла в другу вхідну чергу вузла, в якому знаходиться потрібний файл для його обслуговування. Після обслуговування запиту (згідно з чергою) формується відповідь, якій привласнюється вищий пріоритет. Ця відповідь надходить у першу вхідну чергу вузла і згідно з чергою пересилається в першу вхідну чергу вузла, з якого надійшов запит. Після цього відповідь (згідно з чергою) виводиться на відповідний вузол мережі.

Якщо запит належить до другого типу, то він обслуговується згідно з чергою запитів цього типу і відповідь виводиться на відповідний вузол мережі.

Оскільки лінією зв’язку інформація може пересилатися в обох напрямах (використовується один дуплексний канал передачі даних), і лінія зв’язку загальна для всіх вузлів, то для пересилки повідомлень з першої вхідної черги кожному вузлу по черзі виділяється необхідний час. При цьому, якщо пересилка повідомлень з першої вхідної черги вузла K_i закінчена, то здійснюється перехід до пересилки повідомлень з першої вхідної черги вузла K_{i+1}. Після пересилки повідомлень

з вузла K_n здійснюється пересилка повідомлень з вузла K_1. Під час очікування своєї черги для пересилки повідомлень з першої вхідної черги у вузлі здійснюється обслуговування запитів з другої вхідної черги. Це обслуговування уривається, як тільки вузол одержав дозвіл на пересилку повідомлень з першої вхідної черги.

З описаної схеми обробки запитів виходить, що в процесі функціонування системи виникають затримки при обслуговуванні запитів, обумовлені чергами у вузлах, чергою, згідно з якою кожному вузлу виділяється час для пересилки пріоритетних повідомлень, а також часом обслуговування повідомлень у вузлах. Загальна середня затримка при цьому залежить від розподілу файлів вузлами обчислювальної мережі. Оцінити величину цієї затримки загалом важко. Тому будемо пов'язувати її з об'ємом даних, які необхідно переслати в результаті функціонування системи протягом одиниці часу. Очевидно, чим менший об'єм таких даних та відстань, на яку вони пересилаються, тим менше загальна середня затримка в системі.

Таким чином, у якості критерію оптимальності розподілу файлів вузлами обчислювальної мережі прийнята загальна сума добутків об'ємів даних, які становлять запити і відповіді, породжені функціонуванням системи протягом одиниці часу, на відстані, на які ці дані пересилаються. При цьому вважатимемо, що відстань від вузла K_s до сусіднього вузла K_{s-1} або K_{s+1} рівна 1, а відстань між вузлами K_i і K_j рівна d_{ij}, де

$$d_{ij} = \min\left(|i-j|, n-|i-j|\right). \qquad (8.11)$$

Слід зауважити, що схема обробки запитів може бути іншою. Тому побудуємо модель оптимального розподілу файлів вузлами обчислювальної мережі, яка не залежить від схеми обробки запитів.

Нехай задані величини:

n – число вузлів обчислювальної мережі;

m – число незалежних файлів РБД;

K_j – j-й вузол мережі;

F_i – i-й файл РБД;

L_i – об'єм i-го файлу;

b_j – об'єм пам'яті вузла K_j, призначеної для розміщення файлів;

d_{sj} – відстань між вузлами K_s і K_j ($d_{ss} = 0$, $s = 1, 2, \ldots, n$);

λ_{ij} – інтенсивність запитів до файлу F_i, ініційованих у вузлі K_j;

α_{ij} – об'єм запиту до файлу F_i, ініційованого вузлом K_j;

β_{ij} – об'єм даних, який вимагає виконання запиту до файлу F_i, що надійшов на термінал вузла K_j;

x_{ij} ($i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$) – величини, які визначаються за формулою

$$x_{ij} = \begin{cases} 1, \text{файл } F_i \text{ знаходиться у вузлі } K_j \\ 0, \text{у зворотному випадку.} \end{cases}$$

Тоді об'єм даних, що пересилаються, при виконанні запиту до файлу F_i, ініційованого у вузлі K_j, рівний $(\alpha_{ij} + \beta_{ij}) \cdot (1 - x_{ij})$. Оскільки інтенсивність λ_{ij} породжує об'єм даних $\lambda_{ij}(\alpha_{ij} + \beta_{ij}) \cdot (1 - x_{ij})$, які необхідно пересилати, той добуток цього об'єму даних на відстань, на яку вони пересилаються, рівний

$$R_{ij} = \sum_{s=1}^{n} \lambda_{ij}(\alpha_{ij} + \beta_{ij}) d_{sj} (1 - x_{ij}) x_{is} .$$

Тому загальна сума добутків об'ємів даних, що становлять запити і відповіді, породжені функціонуванням системи протягом одиниці часу, на відстані, на які ці дані пересилаються, визначається формулою

$$R = \sum_{i=1}^{m} \sum_{j=1}^{n} R_{ij} .$$

Таким чином, математична модель задачі буде такою:

$$L = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{s=1}^{n} \lambda_{ij} (\alpha_{ij} + \beta_{ij}) d_{sj} (1 - x_{ij}) x_{is} \rightarrow \min \quad (8.12)$$

при обмеженнях

$$\sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1,2,..., m) \tag{8.13}$$

$$\sum_{i=1}^{m} L_i x_{ij} \le b_j \quad (j = 1,2,..., n) \tag{8.14}$$

$$x_{ij} = \{0 \cup 1\} \quad (i = 1,2,..., m;\ j = 1,2,..., n) \tag{8.15}$$

Перше обмеження означає, що кожен файл F_i (i = 1, 2, ..., m) повинен знаходитися в одному із вузлів, а друге вказує на те, що об'єм локальної бази даних кожного вузла K_j (j = 1, 2, ..., n) не повинен перевищувати об'єм пам'яті цього вузла, призначеної для розміщення файлів.

Якщо позначити

$$\lambda = \sum_{i=1}^{m} \sum_{j=1}^{n} \lambda_{ij},$$

то в якості цільової функції можна було б використовувати функцію

$$\overline{L} = \frac{1}{\lambda} \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{s=1}^{n} \lambda_{ij} (\alpha_{ij} + \beta_{ij}) d_{sj} (1 - x_{ij}) x_{is}.$$

Перепишемо цільову функцію (4.2) у вигляді

$$L = \sum_{i=1}^{m} \sum_{s=1}^{n} Q_{is} x_{is}, \tag{8.16}$$

де $Q_{is} = \sum_{j=1}^{n} \lambda_{ij} (\alpha_{ij} + \beta_{ij}) d_{js}$ – сума добутків об'ємів даних, що пересилаються з вузла K_s і в цей же вузол при функціонуванні системи протягом одиниці часу, на відстані, на які ці дані пересилаються, у випадку зберігання файлу F_i у вузлі K_s. Тоді для реалізації моделі (8.12) можна використовувати алгоритм, аналогічний алгоритму рішення задачі із зіркоподібною топологією. Опишемо цей алгоритм більш детально.

I етап. Визначення початкового розподілу.

1. Знаходимо значення Q_{is}, (i = 1, 2, ..., m; s = 1, 2, ..., n) і складаємо матрицю $Q = \{Q_{is}\}_{m,n}$.

2. Для всіх i = 1, 2, ..., m визначаємо $\min\limits_{1\le s\le n} Q_{is}$. Нехай

$$\min_{1\le s\le n} Q_{is} = Q_{i\,si},\ (i = 1, 2, ..., m).$$

3. Знаходимо початковий розподіл файлів, тобто визначаємо матрицю $X = \{x_{ij}\}_{m.n}$, де

$$x_{is_i} = 1\ (i = 1, 2, ..., m).$$

$$x_{ij} = 0\ \ (i = 1,2,..., m;\ j = 1,2,..., n; (i, j) \neq (i, s_l)).$$

Розподіл X буде завжди оптимальним у тому випадку, якщо на об'єм пам'яті кожного вузла, призначеної для зберігання файлів, не накладаються обмеження.

II етап. Перерозподіл файлів.

1. Для всіх j = 1, 2, ..., n перевіряємо виконання умови (2.14). Якщо ця умова для всіх j виконується, то на цьому робота алгоритму закінчується. Розподіл X є оптимальним. Якщо для деякого індексу $j = r$ маємо

$$\sum_{i=2}^{m} L_i x_{lr} > b_r, \qquad (8.17)$$

то формуємо вектор ознак $E = (\varepsilon_1, \varepsilon_2, ..., \varepsilon_n)$, де $\varepsilon_j = 0$ (j = 1, 2, ..., n), і переходимо до пункту 2.

У процесі роботи алгоритму, якщо деякий вузол K_j переповнений, то відбувається перерозподіл файлів з цього вузла. Після перерозподілу відповідному елементу ε_j вектора E привласнюється значення 1 і вузол K_j вважається закритим для перерозподілу.

2. Перерозподіл файлів із переповненого вузла K_r. Для кожного i, для якого x_{ir} = 1, знаходимо $\min\limits_j (Q_{ij} - Q_{ir})$, де мінімум береться за тими індексами $j \neq r$, для яких вузол $\varepsilon_j = 0$. Нехай

$$\min_j (Q_{ij} - Q_{ir}) = Q_{ij_i} - Q_{ir}.$$

Тоді знаходимо $\min_j\left(Q_{ij_i} - Q_{ir}\right)$, де мінімум береться за всіма індексами i, для яких $x_{ir} = 1$. Нехай

$$\min_i\left(Q_{ij_i} - Q_{ir}\right) = Q_{lj_l} - Q_{lr}.$$

Тоді в матриці X покладемо $x_{lr} = 0$, $x_{lj\,l} = 1$. Це означає, що файл F_l із вузла K_r перерозподілений у вузол K_{jl}. Такому перерозподілу відповідає мінімальне збільшення значення цільової функції.

3. Для індексу $j = r$ перевіряємо умову (8.14). Якщо вона не виконується, то переходимо до операції 2. Якщо умова виконується, то елементу ε_j вектора E привласнюється значення 1 і переходимо до пункту 4.

4. Для всіх індексів j, для яких $\varepsilon_j = 0$, перевіряємо виконання умови (8.14). Якщо для всіх таких індексів j умова виконується, то на цьому робота алгоритму закінчується. Розподіл X приймається за оптимальний. Якщо для деякого індексу $j = r$ умова (8.17) виконується, то переходимо до операції 2.

Таким чином, алгоритм дозволяє за кінцеве число кроків знайти оптимальний або майже оптимальний розподіл файлів вузлами обчислювальної мережі. Для роботи алгоритму повинні бути задані величини m, n, λ_{ij}, α_{ij}, β_{ij}, L_i, b_j, d_{sj} для $i = 1, 2, \ldots, m$; s, $j = 1, 2, \ldots, n$. Результатом роботи алгоритму є матриця X.

Слід зауважити, що сумарний об'єм пам'яті, призначений для зберігання файлів, необхідно брати таким чином, щоб його було достатньо для розміщення файлів при їх перерозподілі. Тут мається на увазі можливий випадок роботи алгоритму, коли $n–1$ вузлів заповнено, а в останній вузол не переміщуються останні файли.

Математична модель розміщення файлів для мережі з довільною топологією

Побудуємо декілька моделей оптимального розміщення файлів у обчислювальній мережі з довільною топологією.

У якості критерію оптимальності можна прийняти середній об’єм даних, що пересилаються лініями зв’язку при виконанні запиту, або загальну вартість трафіку, що генерується розподіленою обчислювальною системою за одиницю часу, або загальний час, що необхідний для обслуговування запитів, які надходять в обчислювальну систему за одиницю часу [17, 28].

1. Розглянемо обчислювальну мережу, кожен вузол якої складається з ЕОМ та апаратури передачі даних. Нехай топологія обчислювальної мережі є довільною (рис. 8.3).

Припустимо, що запит, який надходить з термінального пристрою будь-якого вузла, передбачає доступ до певного файлу РБД і всі запити до одного і того ж файлу мають однакову довжину і потребують для відповіді однаковий об’єм даних. Будемо вважати, що схема обробки запитів полягає у такому.

Запит, що ініційований на терміналі, надходить у вхідну чергу відповідного вузла. Процесор ЕОМ обробляє запити в порядку їх надходження. Якщо потрібний файл міститься в локальній базі даного вузла, на термінал якого надійшов запит, то запит обробляється і результат виводиться на даний термінал. Якщо потрібний файл міститься на іншому вузлі, то запит пересилається на цей вузол, там оброблюється і результат пересилається на початковий вузол. Порядок обслуговування запитів у вузлах не впливає на об’єм даних, що пересилаються каналами зв’язку. Тому він може бути прийнятий таким же чином, як і в моделі із зіркоподібною топологією. Для визначення типу запиту (локально чи дистанційно обслуговується) використовуються довідники локальних баз даних.

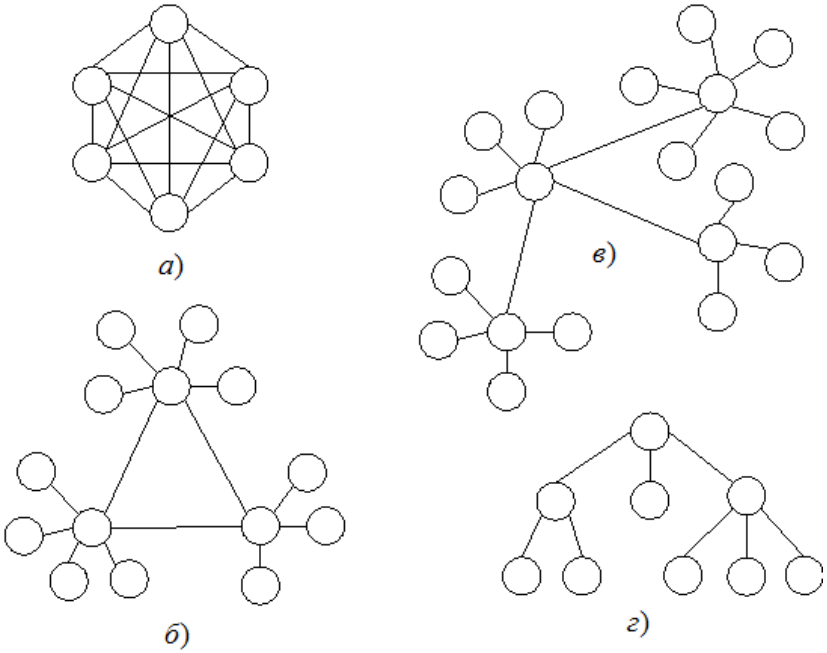


Рис. 8.3. Обчислювальні мережі з довільною топологією:
а – мережа з попарно зв'язаними вузлами;
б – об'єднання локальних мереж із зіркоподібною топологією у глобальну кільцеву мережу;
в – об'єднання локальних мереж із зіркоподібною топологією в глобальну мережу з тією ж топологією;
г – мережа з топологією у вигляді дерева

Із описаної схеми обробки запитів у системі можна стверджувати, що під час обслуговування протягом кожної одиниці часу каналами зв'язку пересилається деякий об'єм даних. Загальне значення цього об'єму залежить від розподілу файлів локальними базами даних. Чим менший середній об'єм даних, що пересилаються каналами зв'язку за одинцю часу, тим вища швидкість обробки запитів.

Нехай задані величини:

n – число вузлів обчислювальної мережі;

m – число незалежних файлів РБД;

K_j – j-й вузол мережі;

F_i – i-й файл РБД;

L_i – об'єм i-го файлу РБД;

b_j – об'єм пам'яті вузла K_j, призначеної для розміщення файлів;

λ_{ij} – інтенсивність запитів до файлу F_i, ініційованих у вузлі K_j;

α_i – об'єм запиту до файлу F_i;

β_i – об'єм даних, який вимагає виконання запиту до файлу F_i.

Тоді об'єм даних, що надходять на термінал вузла, який містить i-й файл, при виконанні запиту до файлу F_i дорівнює $\alpha_i + \beta_i$. При цьому, якщо x_{ij} (i = 1, 2, ..., m; j = 1, 2, ..., n) – величини, які визначаються за формулою

$$x_{ij} = \begin{cases} 1, \text{файл } F_i \text{ знаходиться у вузлі } K_j \\ 0, \text{у зворотному випадку.} \end{cases}$$

Тоді об'єм даних, що пересилаються, при виконанні запиту до файлу F_i, ініційованого у вузлі K_j, рівний $(\alpha_{ij} + \beta_{ij}) \cdot (1 - x_{ij})$.

Оскільки λ_{ij} – інтенсивність запитів до файлу F_i, ініційованих у вузлі K_j, то вона породжує об'єм даних $\lambda_{ij}(\alpha_{ij} + \beta_{ij}) \cdot (1 - x_{ij})$, які необхідно пересилати. Тому загальний об'єм даних, які необхідно пересилати каналами зв'язку між вузлами під час функціонування системи протягом одиниці часу, визначається формулою

$$S = \sum_{i=1}^{m} \sum_{j=1}^{n} \lambda_{ij} (\alpha_{ij} + \beta_{ij})(1 - x_{ij}) .$$

Якщо покласти

$$\lambda = \sum_{i=1}^{m} \sum_{j=1}^{n} \lambda_{ij} ,$$

то середній об'єм даних, які необхідно пересилати каналами зв'язку при виконанні запиту в системі, рівний

$$V = \frac{1}{\lambda} \sum_{i=1}^{m} \sum_{j=1}^{n} \lambda_{ij} (\alpha_{ij} + \beta_{ij})(1 - x_{ij}) . \qquad (8.18)$$

Очевидно, чим менше значення V, тим вища швидкість обслуговування запитів у системі.

Оскільки кожен файл F_i (i = 1, 2, ..., m) повинен знаходитися в одному із вузлів обчислювальної мережі, то

$$\sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1,2,..., m). \tag{8.19}$$

Крім того, об'єм локальної бази даних кожного вузла K_j (j = 1, 2, ..., n) не повинен перевищувати об'єм пам'яті цього вузла, призначений для розміщення файлів. Тому

$$\sum_{i=1}^{m} L_i x_{ij} \le b_j \quad (j = 1,2,..., n). \tag{8.20}$$

Таким чином, задача оптимального розподілу файлів вузлами обчислювальної мережі полягає в тому, щоб визначити значення змінних x_{ij}, де

$$x_{ij} = \{0 \cup 1\} \quad (i = 1,2,..., m;\ j = 1,2,..., n), \tag{2.21}$$

які задовольняють умовам (8.19), (8.20) і забезпечують мінімум лінійної функції (8.18). Отримана математична модель є задачею цілочисельного лінійного програмування з булевими змінними.

Запити до одного і того ж файлу F_i, які надходять на термінали різних вузлів, можуть бути різних типів. У цьому випадку об'єм запиту та об'єм даних, які вилучаються запитом із бази, залежать від вузла, на який надходить запит.

Нехай α_{ij} – об'єм запиту до файлу F_i, ініційованого на терміналі вузла K_j; β_{ij} – об'єм даних, який вимагає виконання запиту до файлу F_i, що надійшов на термінал вузла K_j. Тоді цільова функція задачі оптимального розподілу файлів вузлами обчислювальної мережі може бути записана у вигляді

$$V = \frac{1}{\lambda} \sum_{i=1}^{m} \sum_{j=1}^{n} \lambda_{ij} (\alpha_{ij} + \beta_{ij})(1 - x_{ij}).$$

Запишемо цільову функцію задачі оптимального розподілу файлів вузлами обчислювальної мережі у випадку, коли запити із фіксованого вузла до одного і того ж файлу можуть бути різного типу.

Нехай

s – число типів запитів;

λ_{kij} – інтенсивність запитів k-го типу до файлу F_i, ініційованих у вузлі K_j;

α_{kij} – об'єм запиту k-го типу до файлу F_i, ініційованого вузлом K_j;

β_{kij} – об'єм даних, який вимагає виконання запиту k-го типу до файлу F_i, що надійшов на термінал вузла K_j.

Тоді об'єм даних, що пересилаються при виконанні запиту k-го типу до файлу F_i, ініційованого вузлом Kj, дорівнює $(\alpha_{kij} + \beta_{kij}) \cdot (1 - x_{ij})$. Оскільки інтенсивність λ_{kij} породжує об'єм даних $\lambda_{kij}(\alpha_{kij} + \beta_{kij}) \cdot (1 - x_{ij})$, який необхідно пересилати, то загальний об'єм даних, які потрібно пересилати каналами зв'язку між вузлами у результаті функціонування розподіленої системи в одиницю часу, визначається за формулою

$$S = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{s} \lambda_{kij} (\alpha_{kij} + \beta_{kij})(1 - x_{ij}).$$

Якщо покласти

$$\lambda = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{s} \lambda_{kij},$$

то цільова функція задачі оптимального розподілу файлів вузлами обчислювальної мережі прийме вигляд

$$V = \frac{1}{\lambda} \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{s} \lambda_{kij} (\alpha_{kij} + \beta_{kij})(1 - x_{ij}).$$

Усі повідомлення, які надходять у вхідні черги вузлів, розділимо на два типи: тип 1 – повідомлення, що складають запити, для обробки яких необхідні файли не містяться у базі даних вузла; тип 2 – повідомлення, що складають запити, для обслуговування яких потрібні файли містяться в базі даних відповідного вузла. При цьому будемо вважати, що запит типу 1, який прибув для свого обслуговування у віддалений вузол, перетворюється у запит типу 2.

Введемо величини:

W_{sk} – середній час оброблення повідомлення k-го типу (k = 1, 2) у вузлі K_s,

U_{sk} – інтенсивність надходження повідомлень k-го типу (k = 1, 2) у вхідну чергу вузла K_s.

Тоді аналогічно, як для моделі із кільцевою топологією можна використати такі обмеження

$$U_{s1}W_{s1} + U_{s2}W_{s2} < 1, \ (k = 1, 2, \ldots, n),$$

де

$$U_{s1} = 2\sum_{i=1}^{m} \lambda_{is}\left(1 - x_{is}\right), \ (s = 1, 2, \ldots, n)$$

$$U_{s2} = \sum_{i=1}^{m} \lambda_i x_{is}, \ (s = 1, 2, \ldots, n)$$

$$\lambda_i = \sum_{j=1}^{n} \lambda_{ij} .$$

2. Розглянемо побудову моделі оптимального розподілу файлів вузлами обчислювальної мережі у випадку, коли у якості критерію оптимальності приймається загальна вартість трафіку, який породжується функціонуванням обчислювальної системи за одиницю часу. Припустимо, що топологія обчислювальної мережі і схема обробки запитів така ж, як і в пункті 1.

Нехай

n – число вузлів;

m – число незалежних файлів РБД;

K_j – j-й вузол мережі;

F_i – i-й файл РБД;

L_i – об’єм i-го файлу;

b_j – об’єм пам’яті вузла K_j, призначеної для розміщення файлів;

λ_{ij} – інтенсивність запитів до файлу F_i, ініційованих у вузлі K_j;

α_{ij} – об'єм запиту до файлу F_i, ініційованого вузлом K_j;

β_{ij} – об'єм даних, який вимагає виконання запиту до файлу F_i, що надійшов на термінал вузла K_j;

r_{sj} – вартість передачі одиниці інформації із вузла K_s у вузол K_j ($r_{sj} = 0$, $s = 1, 2, \ldots, n$);

x_{ij} ($i = 1, 2, ..., m$; $j = 1, 2, ..., n$) – величини, які визначаються за формулою

$$x_{ij} = \begin{cases} 1, \text{файл } F_i \text{ знаходиться у вузлі } K_j \\ 0, \text{у зворотному випадку.} \end{cases}$$

Тоді об'єм даних, що пересилаються, при виконанні запиту до файлу F_i, ініційованого у вузлі K_j, рівний $(\alpha_{ij} + \beta_{ij}) \cdot (1 - x_{ij})$. Оскільки інтенсивність λ_{ij} породжує об'єм даних $\lambda_{ij}(\alpha_{ij} + \beta_{ij}) \cdot (1 - x_{ij})$, які необхідно пересилати, то вартість пересилання даних дорівнює

$$C_{ij} = \sum_{s=1}^{n} \lambda_{ij}(\alpha_{ij} + \beta_{ij}) r_{sj}(1 - x_{ij}) x_{is} ,$$

а загальна вартість даних, які необхідно переслати каналами зв'язку між вузлами у результаті функціонування розподіленої системи за одиницю часу, визначається формулою

$$C = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{s=1}^{n} \lambda_{ij}(\alpha_{ij} + \beta_{ij}) r_{sj}(1 - x_{ij}) x_{is} .$$

Таким чином, задача полягає в тому, щоб знайти мінімум функції вартості C при тих же обмеженнях, що і в попередньому пункті.

3. У випадку неоднорідної мережі ЕОМ для більшого завантаження файлами вузлів з більш продуктивними ЕОМ можна використати модель, в якій у якості критерію оптимальності приймається загальний час, необхідний для обслуговування запитів, які надходять у обчислювальну систему в одиницю часу. Розглянемо побудову такої моделі.

Нехай

n – число вузлів;

m – число незалежних файлів РБД;

K_j – j-й вузол мережі;

F_i – i-й файл РБД;

λ_{ij} – інтенсивність запитів до файлу F_i, ініційованих у вузлі K_j;

t_{ijs} – час оброблення у вузлі K_s запиту до файлу F_i, який надійшов із вузла K_j;

$T_{ijs}^{(1)}$ – час, необхідний для пересилання із вузла K_j у вузол K_s запиту до файлу F_i ($T_{iss}^{(1)} = 0$, $s = 1, 2, \ldots, n$);

$T_{ijs}^{(2)}$ – час, необхідний для пересилання із вузла K_s у вузол K_j відповіді на запит до файлу F_i ($T_{iss}^{(2)} = 0$, $s = 1, 2, \ldots, n$);

t_s – час, виділений для оброблення всіх запитів у вузлі K_s, які надійшли за одиницю часу;

x_{ij} ($i = 1, 2, ..., m$; $j = 1, 2, ..., n$) – величини, які визначаються за формулою

$$x_{ij} = \begin{cases} 1, \text{файл } F_i \text{ знаходиться у вузлі } K_j \\ 0, \text{ у зворотному випадку.} \end{cases}$$

Тоді $\lambda_{ij} t_{ijs}$ – час, необхідний для оброблення λ_{ij} запитів, що надійшли із вузла K_j до файлу F_i у вузлі K_s, а з виразу отримаємо

$$t_{ij} = \sum_{s=1}^{n} \lambda_{ij} t_{ijs} x_{is} ,$$

t_{ij} – час, необхідний для оброблення запитів до файлу F_i, які надійшли із вузла K_j за одиницю часу. Тому загальний час, потрібний для обробки запитів, що надійшли до обчислювальної системи за одиницю часу, визначається за формулою

$$t = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{s=1}^{n} \lambda_{ij} t_{ijs} x_{is} .$$

Час, необхідний для пересилання інформації між вузлами K_j та K_s при виконанні λ_{ij} запитів, що надійшли із вузла K_j до файлу F_i, дорівнює $\left(T_{ijs}^{(1)} + T_{ijs}^{(2)}\right)\lambda_{ij}$, а час, який витрачається на пересилання інформації при виконанні запитів до файлу F_i,

котрі надійшли із вузла K_j за одиницю часу, визначається за формулою

$$T_{ij} = \sum_{s=1}^{n} \left(T_{ijs}^{(1)} + T_{ijs}^{(2)} \right) \lambda_{ij} x_{is} .$$

Тому загальна тривалість пересилання інформації при виконанні запитів, які надійшли до обчислювальної системи за одиницю часу, дорівнює

$$T = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{s=1}^{n} \lambda_{ij} \left(T_{ijs}^{(1)} + T_{ijs}^{(2)} \right) x_{is} .$$

Оскільки $\lambda_{ij} t_{ijs}$ – час, необхідний для оброблення λ_{ij} запитів, які надійшли із вузла K_j до файлу F_i, у вузлі K_s, то час, який витрачається на обробку у вузлі K_s усіх запитів, що надійшли до обчислювальної системи за одиницю часу, виражається залежністю

$$\overline{t_s} = \sum_{i=1}^{m} \sum_{j=1}^{n} \lambda_{ij} t_{ijs} x_{is} .$$

Отже, математична модель задачі буде такою:

$$L = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{s=1}^{n} \lambda_{ij} \left(t_{ijs} + T_{ijs}^{(1)} + T_{ijs}^{(2)} \right) x_{is} \to \min$$

при обмеженнях

$$\sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1,2,..., m) ,$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} \lambda_{ij} t_{ijs} x_{is} \le t_s , \text{ (s = 1, 2, …, n)}$$

$$x_{ij} = \{0 \cup 1\} \quad (i = 1,2,..., m; \; j = 1,2,..., n) .$$

Для реалізації побудованих у даному пункті математичних моделей оптимального розподілу файлів вузлами обчислювальної мережі з довільною топологією можна скористатися методом, запропонованим для рішення задачі оптимального розподілу фалів вузлами мережі з кільцевою топологією.

Приклади виконання завдань

Завдання №1

Нехай обчислювальна мережа із зіркоподібною топологією складається з трьох вузлів K_1, K_2, K_3, а РБД містить вісім файлів F_1, F_2, …, F_8. При цьому середні інтенсивності запитів λ_i (i = 1, 2, 3) мають значення: λ_1 = 18, λ_2 = 2, λ_3 = 2, а величини p_{ij} (i = 1, 2, 3; j = 1, 2, …, 8) і W_{ik} (i = 1, 2, 3; k = 1, 2) задані відповідними матрицями P, W:

$$P = \begin{pmatrix} 0{,}6 & 0{,}3 & 0{,}1 & 0 & 0 & 0 & 0 & 0 \\ 0{,}1 & 0{,}1 & 0{,}05 & 0 & 0{,}05 & 0 & 0{,}7 & 0 \\ 0 & 0 & 0{,}5 & 0{,}2 & 0 & 0{,}1 & 0{,}1 & 0 \end{pmatrix}; \; W = \begin{pmatrix} 0{,}1 & 0{,}05 \\ 0{,}1 & 0{,}05 \\ 0{,}1 & 0{,}05 \end{pmatrix}.$$

Знайдемо оптимальний розподіл файлів вузлами обчислювальної мережі. Застосувавши формулу (8.6), знаходимо Q_{js} (j = 1, 2, …, 8; s = 1, 2, 3). Ці величини мають значення, представлені матрицею Q. Знайшовши мінімальне завантаження, можемо виконати початкове розподілення файлів.

$$Q = \begin{pmatrix} 0{,}2 & 10{,}8 & 11 \\ 0{,}2 & 5{,}4 & 5{,}6 \\ 1{,}1 & 2{,}8 & 1{,}9 \\ 0{,}4 & 0{,}4 & 0 \\ 0{,}1 & 0 & 0{,}1 \\ 0{,}2 & 0{,}2 & 0 \\ 1{,}6 & 0{,}2 & 1{,}4 \\ 0 & 0 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} \min\limits_s Q_{1s} = Q_{11} = 0{,}2 \\ \min\limits_s Q_{2s} = Q_{21} = 0{,}2 \\ \min\limits_s Q_{3s} = Q_{31} = 1{,}1 \\ \min\limits_s Q_{4s} = Q_{43} = 0{,}0 \\ \min\limits_s Q_{5s} = Q_{52} = 0 \\ \min\limits_s Q_{6s} = Q_{63} = 0 \\ \min\limits_s Q_{7s} = Q_{72} = 0{,}2 \\ \min\limits_s Q_{8s} = Q_{8\forall} = 0 \end{pmatrix} \Rightarrow X = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

При цьому, оскільки $Q_{81} = Q_{82} = Q_{83} = 0$, файл F_8 можна розмістити в будь-якому вузлі.

Отриманий початковий розподіл є оптимальним. Оптимальне значення лінійної функції L рівне

$$\min L = Q_{11} + Q_{21} + Q_{31} + Q_{44} + Q_{52} + Q_{63} + Q_{72} + Q_{83} = 1{,}7 .$$

Завдання №2

Розглянемо розв'язання задачі оптимізації для змішаної топології засобами MathCad. У якості критерію оберемо мінімум сумарної вартості трафіку, що виникає у обчислювальній системі в результаті її функціонування.

Початковий індекс векторів та матриць: ORIGIN := 1

Кількість вузлів мережі: n := 3 j := 1 .. n

Кількість файлів РБД: m := 6 i := 1 .. m

Об'єми файлів РБД: $L := (0.5 \quad 4 \quad 1 \quad 5 \quad 8 \quad 2)^T$

Об'єми пам'яті фузлів: $b := (10 \quad 6 \quad 5)^T$

Інтенсивність запитів до файлів з вузлів:

$$\lambda := \begin{pmatrix} 0.25 & 0.5 & 0.2 \\ 0.5 & 0.25 & 0.3 \\ 0.1 & 1.5 & 1.2 \\ 2.1 & 1.5 & 0.2 \\ 0 & 0.2 & 0.1 \\ 0.8 & 0.6 & 0.1 \end{pmatrix}$$

Об'єми запитів до файлів з вузлів:

$$\alpha := \begin{pmatrix} 2 & 4 & 1 \\ 2 & 1.5 & 2 \\ 1.5 & 0.25 & 3 \\ 6 & 3 & 3 \\ 8 & 2 & 3 \\ 0.5 & 0.2 & 2 \end{pmatrix}$$

Об'єми відповідей на запити з вузлів:

$$\beta := \begin{pmatrix} 2 & 15 & 10 \\ 2 & 0.5 & 0.5 \\ 0.2 & 0.5 & 5 \\ 2 & 0.5 & 2 \\ 2 & 2 & 0.5 \\ 0.8 & 0.5 & 0.85 \end{pmatrix}$$

Вартість передачі одиниці інформації між вузлами: $r := \begin{pmatrix} 0 & 1 & 2 \\ 1.5 & 0 & 0.5 \\ 2 & 0.2 & 0 \end{pmatrix}$

Вартість трафіку за одиницю часу, який виникає при розміщенні файлу Fi на вузлі Kj: $Q_{i,j} := \sum_{s=1}^{n} \left[\lambda_{i,s}\left(\alpha_{i,s}\cdot r_{s,j} + \beta_{i,s}\cdot r_{j,s}\right)\right]$

Мартиця початкового розміщення файлів X знаходиться, виходячи із мінімального значення вартості трафіку у кожному рядку матриці Q:

$$Q = \begin{pmatrix} 14.9 & 2.29 & 4.5 \\ 2.188 & 2.695 & 4.213 \\ 20.512 & 3.9 & 0.678 \\ 9.5 & 19.22 & 36 \\ 1.7 & 0.085 & 0.28 \\ 1.05 & 1.443 & 2.2 \end{pmatrix} \qquad X0 := \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Перевірка переповнення пам'яті вузлів:

$F_j := \sum_{i=1}^{m} \left(L_i \cdot X0_{i,j}\right) \le b_j$ $\qquad F = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ - переповнення пам'яті вузла K1

- переповнення пам'яті вузла K2

- умова виконується для вузла K3

Загальна вартість трафіку за одиницю часу:

$$C(X) := \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{s=1}^{n}\left[\lambda_{i,j}\left(\alpha_{i,j}\cdot r_{j,s} + \beta_{i,j}\cdot r_{s,j}\right)\left(1 - X_{i,j}\right)X_{i,s}\right] \quad C(X0) = 15.79$$

1-ий крок перерозподілу файлів

Знаходимо файл, який найбільш доцільно перенести із перевантаженого вузла K1 на вузол K3. При цьому сумарна вартість (критерій) повинна збільшитися на мінімально можливе значення. Легко перевірити, що така ситуація можлива лише при переміщенні файла F6 на вузол K3.

$$X1 := \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Перевірка переповнення пам'яті вузлів:

$$F_j := \sum_{i=1}^{m} \left(L_i \cdot X1_{i,j}\right) \le b_j \qquad F = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

- умова виконується для вузла K1
- переповнення пам'яті вузла K2
- умова виконується для вузла K3

Загальна вартість трафіку за одиницю часу: $C(X1) = 16.94$

2-ий крок перерозподілу файлів

Знаходимо файл, який найбільш доцільно перенести із перевантаженого вузла K2 на вузол K3.

$$X2 := \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Перевірка переповнення пам'яті вузлів:

$$F_j := \sum_{i=1}^{m} \left(L_i \cdot X2_{i,j}\right) \le b_j \qquad F = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

- умова виконується для вузла K1
- умова виконується для вузла K2
- переповнення пам'яті вузла K3

Загальна вартість трафіку за одиницю часу: $C(X2) = 17.135$

Отже, знайти оптимальний розподіл файлів не вдалося у зв'язку з обмеженим обсягом памяті вузлів. Найбільш доцільно змінити апаратну частину комп'ютерів.

Завдання для самостійного виконання

№	Рівень завдань		
	Початковий	Базовий	Високий
1	Виконати аудиторні завдання	Представити блок-схему алгоритму розрахунку оптимального розміщення файлів бази даних згідно з варіантом та топологією мережі	
2	–	Знайти оптимальний розподіл файлів вузлами обчислювальної мережі за допомогою будь-якого програмного пакету Excel, MathCAD або Matlab.	Написати програму на будь-якій мові програмування (C/C++, Pascal, Excel+VBA, Java, C#), яка б розв'язувала задачу оптимізації для будь-якої кількості вузлів та файлів у БД. При цьому програма повинна відобра-жати початковий розподіл файлів, значення цільової функції та проміжні кроки перерозподілу
3	Зробити висновок за отриманими результатами та у разі необхідності запропонувати конкретні шляхи покращення обчислювальної мережі		

Обчислювальна мережа із топологією відповідно до варіанта складається з n вузлів K_1, K_2, …, K_n. Розподілена база даних містить m файлів F_1, F_2, …, F_m. При цьому середні інтенсивності запитів λ_i, λ_{ij}; кількість запитів за добу V_{ij}; середній час обробки запитів W_{ik}; об'єми запитів до файлів α_{ij}, об'єми даних запиту β_{ij}, об'єми файлів L_i; об'єми пам'яті вузлів b_j; вартість передачі одиниці інформації r_{sj} ($i = 1, 2, \ldots, m$; $j = 1,$

2, …, n; s = 1, 2, …, n; k = 1, 2) задані відповідними матрицями та векторами λ, V, W, α, β, L, b, r згідно з варіантом. Виконати оптимізацію розміщення файлів РБД вузлами мережі з урахуванням найбільш доцільного критерію оптимальності, детально розглянутого у теоретичній частині роботи.

Варіант №1

Топологія: зірка.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 8 \end{pmatrix};\ \lambda = \begin{pmatrix} 15 \\ 5 \\ 2 \end{pmatrix};\ V = \begin{pmatrix} 60 & 20 & 10 & 10 & 0 & 0 & 0 & 0 \\ 10 & 10 & 5 & 0 & 5 & 0 & 7 & 0 \\ 3 & 15 & 5 & 20 & 0 & 10 & 25 & 0 \end{pmatrix};$$

$$W = \begin{pmatrix} 0{,}1 & 0{,}05 \\ 0{,}15 & 0{,}1 \\ 0{,}1 & 0{,}05 \end{pmatrix}$$

Варіант №2

Топологія: кільце.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix};\ b = \begin{pmatrix} 10 \\ 8 \\ 2 \end{pmatrix};\ L = \begin{pmatrix} 0{,}25 \\ 5 \\ 2{,}3 \\ 8 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}5 & 0 \\ 0{,}5 & 0{,}2 & 0{,}3 \\ 0{,}2 & 0{,}2 & 0{,}6 \\ 0{,}25 & 0{,}25 & 0{,}5 \\ 0{,}1 & 0{,}8 & 0{,}1 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 2 & 1 \\ 2 & 1{,}5 & 2 \\ 1{,}25 & 0{,}25 & 6 \\ 5 & 2 & 3 \\ 1 & 2{,}5 & 2 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 10 & 15 \\ 2 & 0{,}5 & 0{,}6 \\ 0{,}85 & 5 & 10 \\ 2 & 2 & 0{,}5 \\ 0{,}8 & 0{,}6 & 0{,}85 \end{pmatrix}$$

Варіант №3

Топологія: змішана.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix};\ b = \begin{pmatrix} 10 \\ 6 \\ 5 \end{pmatrix};\ L = \begin{pmatrix} 0{,}5 \\ 4 \\ 1 \\ 5 \\ 8 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}5 & 0{,}2 \\ 0{,}5 & 0{,}25 & 0{,}3 \\ 0{,}1 & 1{,}5 & 1{,}2 \\ 2{,}1 & 1{,}5 & 0{,}2 \\ 0 & 0{,}2 & 0{,}1 \\ 0{,}8 & 0{,}6 & 0{,}1 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 4 & 1 \\ 2 & 1{,}5 & 2 \\ 1{,}25 & 0{,}25 & 3 \\ 6 & 3 & 3 \\ 8 & 2 & 3 \\ 0{,}5 & 0{,}2 & 2 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 15 & 10 \\ 2 & 0{,}5 & 0{,}5 \\ 0{,}2 & 0{,}5 & 5 \\ 2 & 0{,}5 & 2 \\ 2 & 2 & 0{,}5 \\ 0{,}8 & 0{,}5 & 0{,}85 \end{pmatrix};\ r = \begin{pmatrix} 0 & 1{,}0 & 0{,}2 \\ 1{,}25 & 0 & 0{,}15 \\ 0{,}1 & 0{,}15 & 0 \end{pmatrix}$$

Варіант №4

Топологія: зірка.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 4 \\ 8 \end{pmatrix};\ \lambda = \begin{pmatrix} 10 \\ 5 \\ 3 \\ 15 \end{pmatrix};$$

$$V = \begin{pmatrix} 50 & 20 & 70 & 0 & 0 & 0 & 0 & 0 \\ 0 & 25 & 60 & 23 & 25 & 0 & 0 & 0 \\ 0 & 0 & 0 & 12 & 15 & 0 & 0 & 0 \\ 10 & 0 & 0 & 0 & 20 & 30 & 15 & 25 \end{pmatrix};\ W = \begin{pmatrix} 0{,}2 & 0{,}01 \\ 0{,}5 & 0{,}01 \\ 0{,}8 & 0{,}02 \\ 0{,}5 & 0{,}01 \end{pmatrix}$$

Варіант №5

Топологія: кільце.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix};\ b = \begin{pmatrix} 3 \\ 8 \\ 10 \end{pmatrix};\ L = \begin{pmatrix} 0{,}5 \\ 4 \\ 2 \\ 5 \\ 8 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}5 & 0{,}2 \\ 0{,}5 & 0{,}25 & 0{,}3 \\ 0{,}1 & 0{,}5 & 1{,}2 \\ 2{,}1 & 2{,}5 & 0{,}2 \\ 0 & 0{,}2 & 0{,}1 \\ 0{,}8 & 0{,}6 & 0{,}1 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 5 & 1 \\ 2 & 1{,}5 & 2 \\ 1{,}25 & 0{,}25 & 3 \\ 6 & 5 & 3 \\ 10 & 2 & 3 \\ 0{,}5 & 0{,}2 & 0{,}2 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 15 & 10 \\ 2 & 0{,}5 & 0{,}5 \\ 0{,}2 & 6 & 5 \\ 2 & 0{,}5 & 2 \\ 2 & 2 & 0{,}5 \\ 0{,}8 & 0{,}5 & 0{,}85 \end{pmatrix}$$

Варіант №6

Топологія: змішана.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix};\ b = \begin{pmatrix} 10 \\ 8 \\ 2 \end{pmatrix};\ L = \begin{pmatrix} 0{,}25 \\ 5 \\ 2{,}3 \\ 8 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}5 & 0 \\ 0{,}5 & 0{,}2 & 0{,}3 \\ 0{,}2 & 0{,}2 & 0{,}6 \\ 0{,}25 & 0{,}25 & 0{,}5 \\ 0{,}1 & 0{,}8 & 0{,}1 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 2 & 1 \\ 2 & 1{,}5 & 2 \\ 1{,}25 & 0{,}25 & 6 \\ 5 & 2 & 3 \\ 1 & 2{,}5 & 2 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 10 & 15 \\ 2 & 0{,}5 & 0{,}6 \\ 0{,}85 & 5 & 10 \\ 2 & 2 & 0{,}5 \\ 0{,}8 & 0{,}6 & 0{,}85 \end{pmatrix};\ r = \begin{pmatrix} 0 & 1{,}5 & 0{,}25 \\ 1{,}5 & 0 & 0{,}15 \\ 0{,}25 & 0{,}15 & 0 \end{pmatrix}$$

Варіант №7

Топологія: зірка.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 4 \\ 8 \end{pmatrix};\ \lambda = \begin{pmatrix} 10 \\ 5 \\ 3 \\ 15 \end{pmatrix};$$

$$V = \begin{pmatrix} 5 & 0 & 10 & 80 & 50 & 12 & 26 & 35 \\ 0 & 25 & 0 & 0 & 25 & 0 & 0 & 0 \\ 0 & 0 & 0 & 12 & 15 & 0 & 0 & 0 \\ 15 & 0 & 20 & 0 & 20 & 5 & 15 & 25 \end{pmatrix};\ W = \begin{pmatrix} 0{,}2 & 0{,}01 \\ 0{,}3 & 0{,}05 \\ 0{,}6 & 0{,}05 \\ 0{,}1 & 0{,}01 \end{pmatrix}$$

Варіант №8

Топологія: кільце.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix};\ b = \begin{pmatrix} 3 \\ 6 \\ 10 \end{pmatrix};\ L = \begin{pmatrix} 0{,}5 \\ 4 \\ 1 \\ 5 \\ 8 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}5 & 0{,}2 \\ 0{,}5 & 0{,}25 & 0{,}3 \\ 0{,}1 & 1{,}5 & 1{,}2 \\ 2{,}1 & 1{,}5 & 0{,}2 \\ 0 & 0{,}2 & 0{,}1 \\ 0{,}8 & 0{,}6 & 0{,}1 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 4 & 1 \\ 2 & 1{,}5 & 2 \\ 1{,}25 & 0{,}25 & 3 \\ 6 & 3 & 3 \\ 8 & 2 & 3 \\ 0{,}5 & 0{,}2 & 2 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 15 & 10 \\ 2 & 0{,}5 & 0{,}5 \\ 0{,}2 & 0{,}5 & 5 \\ 2 & 0{,}5 & 2 \\ 2 & 2 & 0{,}5 \\ 0{,}8 & 0{,}5 & 0{,}85 \end{pmatrix}$$

Варіант №9

Топологія: змішана.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix};\ b = \begin{pmatrix} 3 \\ 8 \\ 10 \end{pmatrix};\ L = \begin{pmatrix} 0{,}5 \\ 4 \\ 2 \\ 5 \\ 8 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}5 & 0{,}2 \\ 0{,}5 & 0{,}25 & 0{,}3 \\ 0{,}1 & 0{,}5 & 1{,}2 \\ 2{,}1 & 2{,}5 & 0{,}2 \\ 0 & 0{,}2 & 0{,}1 \\ 0{,}8 & 0{,}6 & 0{,}1 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 5 & 1 \\ 2 & 1{,}5 & 2 \\ 1{,}25 & 0{,}25 & 3 \\ 6 & 5 & 3 \\ 10 & 2 & 3 \\ 0{,}5 & 0{,}2 & 0{,}2 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 15 & 10 \\ 2 & 0{,}5 & 0{,}5 \\ 0{,}2 & 6 & 5 \\ 2 & 0{,}5 & 2 \\ 2 & 2 & 0{,}5 \\ 0{,}8 & 0{,}5 & 0{,}85 \end{pmatrix};\ r = \begin{pmatrix} 0 & 0{,}2 & 0{,}25 \\ 1{,}5 & 0 & 0{,}5 \\ 0{,}35 & 0{,}15 & 0 \end{pmatrix}$$

Варіант №10

Топологія: зірка.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 4 \\ 8 \end{pmatrix};\ \lambda = \begin{pmatrix} 5 \\ 15 \\ 10 \\ 15 \end{pmatrix};$$

$$V = \begin{pmatrix} 0 & 15 & 20 & 0 & 0 & 0 & 20 & 0 \\ 0 & 30 & 0 & 23 & 25 & 0 & 25 & 0 \\ 5 & 0 & 0 & 5 & 10 & 0 & 15 & 0 \\ 10 & 0 & 20 & 0 & 50 & 15 & 20 & 15 \end{pmatrix};\ W = \begin{pmatrix} 0{,}2 & 0{,}01 \\ 0{,}3 & 0{,}01 \\ 0{,}4 & 0{,}05 \\ 0{,}5 & 0{,}05 \end{pmatrix}$$

Варіант №11

Топологія: кільце.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \end{pmatrix};\ b = \begin{pmatrix} 3 \\ 10 \\ 8 \\ 13 \end{pmatrix};\ L = \begin{pmatrix} 0{,}25 \\ 6 \\ 1 \\ 5 \\ 10 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}5 & 0{,}25 & 0 \\ 1 & 1 & 0{,}5 & 0 \\ 2{,}5 & 1{,}5 & 0{,}25 & 0{,}2 \\ 1 & 0{,}2 & 0{,}2 & 0{,}3 \\ 0{,}2 & 0{,}3 & 0{,}2 & 0{,}2 \\ 0{,}1 & 1 & 0{,}5 & 0{,}6 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 2 & 1 & 1 \\ 2 & 1{,}5 & 2 & 1 \\ 1{,}5 & 0{,}5 & 3 & 3 \\ 4 & 2 & 5 & 0{,}2 \\ 2{,}5 & 2 & 5 & 1 \\ 0 & 2{,}5 & 2 & 3 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 5 & 10 & 15 \\ 2 & 0{,}2 & 0{,}5 & 2{,}5 \\ 1 & 2 & 1{,}5 & 2 \\ 0{,}5 & 2 & 3 & 1 \\ 2 & 3 & 0{,}2 & 6 \\ 1 & 2 & 5 & 1 \end{pmatrix}$$

Варіант №12

Топологія: змішана.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix};\ b = \begin{pmatrix} 3 \\ 6 \\ 10 \end{pmatrix};\ L = \begin{pmatrix} 0{,}5 \\ 4 \\ 1 \\ 5 \\ 8 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}5 & 0{,}2 \\ 0{,}5 & 0{,}25 & 0{,}3 \\ 0{,}1 & 1{,}5 & 1{,}2 \\ 2{,}1 & 1{,}5 & 0{,}2 \\ 0 & 0{,}2 & 0{,}1 \\ 0{,}8 & 0{,}6 & 0{,}1 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 4 & 1 \\ 2 & 1{,}5 & 2 \\ 1{,}25 & 0{,}25 & 3 \\ 6 & 3 & 3 \\ 8 & 2 & 3 \\ 0{,}5 & 0{,}2 & 2 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 15 & 10 \\ 2 & 0{,}5 & 0{,}5 \\ 0{,}2 & 0{,}5 & 5 \\ 2 & 0{,}5 & 2 \\ 2 & 2 & 0{,}5 \\ 0{,}8 & 0{,}5 & 0{,}85 \end{pmatrix};\ r = \begin{pmatrix} 0 & 1{,}5 & 0{,}25 \\ 2{,}5 & 0 & 0{,}5 \\ 0{,}5 & 0{,}5 & 0 \end{pmatrix}$$

Варіант №13

Топологія: зірка.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 8 \end{pmatrix};\ \lambda = \begin{pmatrix} 5 \\ 15 \\ 4 \end{pmatrix};$$

$$V = \begin{pmatrix} 10 & 25 & 40 & 0 & 0 & 0 & 60 & 0 \\ 10 & 10 & 0 & 0 & 5 & 0 & 100 & 0 \\ 30 & 15 & 50 & 5 & 0 & 15 & 25 & 0 \end{pmatrix};\ W = \begin{pmatrix} 0{,}1 & 0{,}05 \\ 0{,}15 & 0{,}05 \\ 0{,}05 & 0{,}01 \end{pmatrix}$$

Варіант №14

Топологія: кільце.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \end{pmatrix};\ b = \begin{pmatrix} 12 \\ 2 \\ 8 \\ 12 \end{pmatrix};\ L = \begin{pmatrix} 0{,}25 \\ 6 \\ 1 \\ 4 \\ 8 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}5 & 0{,}25 & 0 \\ 1 & 1 & 0{,}5 & 0 \\ 2{,}5 & 1{,}5 & 0{,}25 & 0{,}2 \\ 1 & 0{,}2 & 0{,}2 & 0{,}3 \\ 0{,}2 & 0{,}3 & 0{,}2 & 0{,}2 \\ 0{,}1 & 1 & 0{,}5 & 0{,}6 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 2 & 1 & 1 \\ 2 & 1{,}5 & 0 & 1 \\ 1{,}5 & 0{,}5 & 3 & 3 \\ 4 & 2 & 4 & 0{,}2 \\ 2{,}5 & 2 & 5 & 1 \\ 0 & 2{,}5 & 2 & 3 \end{pmatrix}; \ \beta = \begin{pmatrix} 2 & 5 & 5 & 15 \\ 2 & 0{,}2 & 0{,}5 & 2{,}5 \\ 1 & 2 & 1{,}5 & 2 \\ 0{,}5 & 2 & 3 & 4 \\ 2 & 3 & 0{,}2 & 6 \\ 2 & 2 & 5 & 1 \end{pmatrix}$$

Варіант №15

Топологія: змішана.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}; \ b = \begin{pmatrix} 12 \\ 10 \\ 2 \end{pmatrix}; \ L = \begin{pmatrix} 0{,}25 \\ 5 \\ 2{,}3 \\ 10 \\ 2 \end{pmatrix}; \ \lambda = \begin{pmatrix} 0{,}25 & 0{,}75 & 0 \\ 0{,}5 & 0{,}2 & 0{,}3 \\ 0{,}2 & 0{,}2 & 0{,}6 \\ 0{,}25 & 0{,}25 & 0{,}5 \\ 0{,}1 & 0{,}8 & 0{,}1 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 2 & 1 \\ 2 & 1{,}5 & 2 \\ 1{,}25 & 0{,}25 & 6 \\ 10 & 2 & 3 \\ 1 & 2{,}5 & 2 \end{pmatrix}; \ \beta = \begin{pmatrix} 2 & 10 & 15 \\ 2 & 0{,}5 & 0{,}6 \\ 0{,}85 & 5 & 6 \\ 2 & 2 & 0{,}5 \\ 0{,}8 & 0{,}6 & 0{,}85 \end{pmatrix}; \ r = \begin{pmatrix} 0 & 1{,}5 & 0{,}25 \\ 1{,}5 & 0 & 0{,}15 \\ 0{,}25 & 0{,}15 & 0 \end{pmatrix}$$

Варіант №16

Топологія: зірка.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 10 \end{pmatrix}; \ \lambda = \begin{pmatrix} 10 \\ 8 \\ 3 \end{pmatrix};$$

$$V = \begin{pmatrix} 10 & 50 & 20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 25 & 15 & 25 & 26 & 0 & 5 & 10 & 0 \\ 15 & 25 & 0 & 25 & 0 & 10 & 12 & 18 & 0 & 5 \end{pmatrix}; \ W = \begin{pmatrix} 0{,}1 & 0{,}05 \\ 0{,}15 & 0{,}1 \\ 0{,}1 & 0{,}05 \end{pmatrix}$$

Варіант №17

Топологія: кільце.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix};\ b = \begin{pmatrix} 10 \\ 6 \\ 5 \end{pmatrix};\ L = \begin{pmatrix} 0{,}5 \\ 4 \\ 1 \\ 5 \\ 8 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}5 & 0{,}2 \\ 0{,}5 & 0{,}25 & 0{,}3 \\ 0{,}1 & 1{,}5 & 1{,}2 \\ 2{,}1 & 1{,}5 & 0{,}2 \\ 0 & 0{,}2 & 0{,}1 \\ 0{,}8 & 0{,}6 & 0{,}1 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 4 & 1 \\ 2 & 1{,}5 & 2 \\ 1{,}25 & 0{,}25 & 3 \\ 6 & 3 & 3 \\ 8 & 2 & 3 \\ 0{,}5 & 0{,}2 & 2 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 15 & 10 \\ 2 & 0{,}5 & 0{,}5 \\ 0{,}2 & 0{,}5 & 5 \\ 2 & 0{,}5 & 2 \\ 2 & 2 & 0{,}5 \\ 0{,}8 & 0{,}5 & 0{,}85 \end{pmatrix}$$

Варіант №18

Топологія: змішана.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix};\ b = \begin{pmatrix} 10 \\ 8 \\ 3 \end{pmatrix};\ L = \begin{pmatrix} 0{,}5 \\ 4 \\ 2 \\ 5 \\ 8 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}5 & 0{,}2 \\ 0{,}5 & 0{,}25 & 0{,}3 \\ 0{,}1 & 0{,}5 & 1{,}2 \\ 2{,}1 & 2{,}5 & 0{,}2 \\ 0 & 0{,}2 & 0{,}1 \\ 0{,}8 & 0{,}6 & 0{,}1 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 5 & 1 \\ 2 & 1{,}5 & 2 \\ 1{,}25 & 0{,}25 & 3 \\ 6 & 5 & 3 \\ 10 & 2 & 3 \\ 0{,}5 & 0{,}2 & 0{,}2 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 15 & 10 \\ 2 & 0{,}5 & 0{,}5 \\ 0{,}2 & 6 & 5 \\ 2 & 0{,}5 & 2 \\ 2 & 2 & 0{,}5 \\ 0{,}8 & 0{,}5 & 0{,}85 \end{pmatrix};\ r = \begin{pmatrix} 0 & 1{,}5 & 0{,}25 \\ 0{,}5 & 0 & 0{,}15 \\ 1{,}2 & 0{,}55 & 0 \end{pmatrix}$$

Варіант №19

Топологія: зірка.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 4 \\ 8 \end{pmatrix};\ \lambda = \begin{pmatrix} 10 \\ 5 \\ 3 \\ 15 \end{pmatrix};$$

$$V = \begin{pmatrix} 0 & 25 & 35 & 0 & 0 & 15 & 0 & 0 \\ 5 & 10 & 30 & 0 & 25 & 0 & 25 & 0 \\ 0 & 0 & 0 & 10 & 15 & 0 & 10 & 0 \\ 10 & 0 & 80 & 0 & 20 & 15 & 0 & 25 \end{pmatrix};\ W = \begin{pmatrix} 0{,}15 & 0{,}01 \\ 0{,}2 & 0{,}01 \\ 0{,}3 & 0{,}05 \\ 0{,}5 & 0{,}08 \end{pmatrix}$$

Варіант №20

Топологія: кільце.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix};\ b = \begin{pmatrix} 10 \\ 8 \\ 3 \end{pmatrix};\ L = \begin{pmatrix} 0{,}5 \\ 5 \\ 0{,}3 \\ 5 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}75 & 0{,}2 \\ 0{,}5 & 0{,}2 & 0{,}3 \\ 0 & 0{,}8 & 0{,}6 \\ 0{,}25 & 0{,}25 & 0{,}5 \\ 0{,}2 & 0{,}8 & 0{,}1 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 2 & 1 \\ 2 & 2 & 2 \\ 1{,}25 & 0{,}5 & 6 \\ 10 & 2 & 3 \\ 1 & 2{,}5 & 2 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 10 & 15 \\ 2 & 0{,}5 & 0{,}6 \\ 0{,}85 & 8 & 3 \\ 2 & 2 & 0{,}5 \\ 0{,}8 & 0{,}6 & 0{,}85 \end{pmatrix}$$

Варіант №21

Топологія: змішана.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \end{pmatrix};\ b = \begin{pmatrix} 13 \\ 8 \\ 2 \\ 10 \end{pmatrix};\ L = \begin{pmatrix} 0{,}25 \\ 6 \\ 1 \\ 5 \\ 10 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}5 & 0{,}25 & 0 \\ 1 & 1 & 0{,}5 & 0 \\ 2{,}5 & 1{,}5 & 0{,}25 & 0{,}2 \\ 1 & 0{,}2 & 0{,}2 & 0{,}3 \\ 0{,}2 & 0{,}3 & 0{,}2 & 0{,}2 \\ 0{,}1 & 1 & 0{,}5 & 0{,}6 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 2 & 1 & 1 \\ 2 & 1{,}5 & 2 & 1 \\ 1{,}5 & 0{,}5 & 3 & 3 \\ 4 & 2 & 5 & 0{,}2 \\ 2{,}5 & 2 & 5 & 1 \\ 0 & 2{,}5 & 2 & 3 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 5 & 10 & 15 \\ 2 & 0{,}2 & 0{,}5 & 2{,}5 \\ 1 & 2 & 1{,}5 & 2 \\ 0{,}5 & 2 & 3 & 1 \\ 2 & 3 & 0{,}2 & 6 \\ 1 & 2 & 5 & 1 \end{pmatrix};\ r = \begin{pmatrix} 0 & 0{,}2 & 0{,}5 & 1{,}0 \\ 0{,}2 & 0 & 1{,}0 & 0{,}6 \\ 0{,}5 & 1{,}0 & 0 & 0{,}2 \\ 0{,}2 & 0{,}4 & 0{,}1 & 0 \end{pmatrix}$$

Варіант №22

Топологія: зірка.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 8 \end{pmatrix};\ \lambda = \begin{pmatrix} 12 \\ 8 \\ 10 \end{pmatrix};$$

$$V = \begin{pmatrix} 40 & 30 & 20 & 10 & 0 & 0 & 5 & 0 \\ 25 & 10 & 10 & 0 & 0 & 0 & 20 & 0 \\ 20 & 15 & 15 & 20 & 0 & 10 & 0 & 0 \end{pmatrix};\ W = \begin{pmatrix} 0{,}15 & 0{,}1 \\ 0{,}15 & 0{,}05 \\ 0{,}12 & 0{,}05 \end{pmatrix}$$

Варіант №23

Топологія: кільце.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix};\ b = \begin{pmatrix} 10 \\ 6 \\ 5 \end{pmatrix};\ L = \begin{pmatrix} 0{,}5 \\ 4 \\ 1 \\ 5 \\ 8 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}5 & 0{,}2 \\ 0{,}5 & 0{,}25 & 0{,}3 \\ 0{,}1 & 1{,}5 & 1{,}2 \\ 2{,}1 & 1{,}5 & 0{,}2 \\ 0 & 0{,}2 & 0{,}1 \\ 0{,}8 & 0{,}6 & 0{,}1 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 4 & 1 \\ 2 & 1{,}5 & 2 \\ 1{,}25 & 0{,}25 & 3 \\ 6 & 3 & 3 \\ 8 & 2 & 3 \\ 0{,}5 & 0{,}2 & 2 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 15 & 10 \\ 2 & 0{,}5 & 0{,}5 \\ 0{,}2 & 0{,}5 & 5 \\ 2 & 0{,}5 & 2 \\ 2 & 2 & 0{,}5 \\ 0{,}8 & 0{,}5 & 0{,}85 \end{pmatrix}$$

Варіант №24

Топологія: змішана.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix};\ b = \begin{pmatrix} 10 \\ 8 \\ 3 \end{pmatrix};\ L = \begin{pmatrix} 0{,}5 \\ 5 \\ 0{,}3 \\ 5 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}75 & 0{,}2 \\ 0{,}5 & 0{,}2 & 0{,}3 \\ 0 & 0{,}8 & 0{,}6 \\ 0{,}25 & 0{,}25 & 0{,}5 \\ 0{,}2 & 0{,}8 & 0{,}1 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 2 & 1 \\ 2 & 2 & 2 \\ 1{,}25 & 0{,}5 & 6 \\ 10 & 2 & 3 \\ 1 & 2{,}5 & 2 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 10 & 15 \\ 2 & 0{,}5 & 0{,}6 \\ 0{,}85 & 8 & 3 \\ 2 & 2 & 0{,}5 \\ 0{,}8 & 0{,}6 & 0{,}85 \end{pmatrix};\ r = \begin{pmatrix} 0 & 1{,}0 & 0{,}2 \\ 0{,}5 & 0 & 0{,}15 \\ 1{,}0 & 0{,}15 & 0 \end{pmatrix}$$

Варіант №25

Топологія: зірка.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 4 \\ 8 \end{pmatrix};\ \lambda = \begin{pmatrix} 5 \\ 10 \\ 5 \\ 15 \end{pmatrix};$$

$$V = \begin{pmatrix} 0 & 50 & 20 & 0 & 0 & 7 & 20 & 0 \\ 0 & 30 & 0 & 23 & 25 & 0 & 25 & 0 \\ 15 & 0 & 0 & 10 & 15 & 0 & 50 & 0 \\ 10 & 0 & 20 & 0 & 50 & 30 & 20 & 15 \end{pmatrix};\ W = \begin{pmatrix} 0{,}2 & 0{,}01 \\ 0{,}5 & 0{,}01 \\ 0{,}8 & 0{,}02 \\ 0{,}5 & 0{,}01 \end{pmatrix}$$

Варіант №26

Топологія: кільце.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix};\ b = \begin{pmatrix} 10 \\ 8 \\ 3 \end{pmatrix};\ L = \begin{pmatrix} 0{,}5 \\ 4 \\ 2 \\ 5 \\ 8 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}5 & 0{,}2 \\ 0{,}5 & 0{,}25 & 0{,}3 \\ 0{,}1 & 0{,}5 & 1{,}2 \\ 2{,}1 & 2{,}5 & 0{,}2 \\ 0 & 0{,}2 & 0{,}1 \\ 0{,}8 & 0{,}6 & 0{,}1 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 5 & 1 \\ 2 & 1{,}5 & 2 \\ 1{,}25 & 0{,}25 & 3 \\ 6 & 5 & 3 \\ 10 & 2 & 3 \\ 0{,}5 & 0{,}2 & 0{,}2 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 15 & 10 \\ 2 & 0{,}5 & 0{,}5 \\ 0{,}2 & 6 & 5 \\ 2 & 0{,}5 & 2 \\ 2 & 2 & 0{,}5 \\ 0{,}8 & 0{,}5 & 0{,}85 \end{pmatrix}$$

Варіант №27

Топологія: змішана.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix};\ b = \begin{pmatrix} 10 \\ 6 \\ 5 \end{pmatrix};\ L = \begin{pmatrix} 0{,}5 \\ 4 \\ 1 \\ 5 \\ 8 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}5 & 0{,}2 \\ 0{,}5 & 0{,}25 & 0{,}3 \\ 0{,}1 & 1{,}5 & 1{,}2 \\ 2{,}1 & 1{,}5 & 0{,}2 \\ 0 & 0{,}2 & 0{,}1 \\ 0{,}8 & 0{,}6 & 0{,}1 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 4 & 1 \\ 2 & 1{,}5 & 2 \\ 1{,}25 & 0{,}25 & 3 \\ 6 & 3 & 3 \\ 8 & 2 & 3 \\ 0{,}5 & 0{,}2 & 2 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 15 & 10 \\ 2 & 0{,}5 & 0{,}5 \\ 0{,}2 & 0{,}5 & 5 \\ 2 & 0{,}5 & 2 \\ 2 & 2 & 0{,}5 \\ 0{,}8 & 0{,}5 & 0{,}85 \end{pmatrix};\ r = \begin{pmatrix} 0 & 1{,}0 & 2{,}0 \\ 1{,}5 & 0 & 0{,}15 \\ 2{,}0 & 0{,}15 & 0 \end{pmatrix}$$

Варіант №28

Топологія: зірка.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 8 \end{pmatrix};\ \lambda = \begin{pmatrix} 10 \\ 15 \\ 2 \end{pmatrix};$$

$$V = \begin{pmatrix} 100 & 50 & 100 & 0 & 0 & 20 & 0 & 0 \\ 10 & 10 & 50 & 0 & 5 & 0 & 70 & 0 \\ 30 & 15 & 50 & 0 & 0 & 10 & 25 & 0 \end{pmatrix};\ W = \begin{pmatrix} 0{,}1 & 0{,}05 \\ 0{,}15 & 0{,}1 \\ 0{,}05 & 0{,}01 \end{pmatrix}$$

Варіант №29

Топологія: кільце.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix};\ b = \begin{pmatrix} 12 \\ 10 \\ 2 \end{pmatrix};\ L = \begin{pmatrix} 0{,}25 \\ 5 \\ 2{,}3 \\ 10 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}75 & 0 \\ 0{,}5 & 0{,}2 & 0{,}3 \\ 0{,}2 & 0{,}2 & 0{,}6 \\ 0{,}25 & 0{,}25 & 0{,}5 \\ 0{,}1 & 0{,}8 & 0{,}1 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 2 & 1 \\ 2 & 1{,}5 & 2 \\ 1{,}25 & 0{,}25 & 6 \\ 10 & 2 & 3 \\ 1 & 2{,}5 & 2 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 10 & 15 \\ 2 & 0{,}5 & 0{,}6 \\ 0{,}85 & 5 & 6 \\ 2 & 2 & 0{,}5 \\ 0{,}8 & 0{,}6 & 0{,}85 \end{pmatrix}$$

Варіант №30

Топологія: змішана.

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \end{pmatrix};\ b = \begin{pmatrix} 12 \\ 2 \\ 8 \\ 12 \end{pmatrix};\ L = \begin{pmatrix} 0{,}25 \\ 6 \\ 1 \\ 4 \\ 8 \\ 2 \end{pmatrix};\ \lambda = \begin{pmatrix} 0{,}25 & 0{,}5 & 0{,}25 & 0 \\ 1 & 1 & 0{,}5 & 0 \\ 2{,}5 & 1{,}5 & 0{,}25 & 0{,}2 \\ 1 & 0{,}2 & 0{,}2 & 0{,}3 \\ 0{,}2 & 0{,}3 & 0{,}2 & 0{,}2 \\ 0{,}1 & 1 & 0{,}5 & 0{,}6 \end{pmatrix};$$

$$\alpha = \begin{pmatrix} 2 & 2 & 1 & 1 \\ 2 & 1{,}5 & 0 & 1 \\ 1{,}5 & 0{,}5 & 3 & 3 \\ 4 & 2 & 4 & 0{,}2 \\ 2{,}5 & 2 & 5 & 1 \\ 0 & 2{,}5 & 2 & 3 \end{pmatrix};\ \beta = \begin{pmatrix} 2 & 5 & 5 & 15 \\ 2 & 0{,}2 & 0{,}5 & 2{,}5 \\ 1 & 2 & 1{,}5 & 2 \\ 0{,}5 & 2 & 3 & 4 \\ 2 & 3 & 0{,}2 & 6 \\ 2 & 2 & 5 & 1 \end{pmatrix};$$

$$r = \begin{pmatrix} 0 & 0,2 & 0,3 & 0,25 \\ 0,2 & 0 & 1,0 & 0,8 \\ 0,3 & 1,0 & 0 & 0,2 \\ 0,25 & 0,8 & 0,2 & 0 \end{pmatrix};$$

Контрольні питання

1. Дайте визначення розподіленої бази даних.
2. Як можна оцінити обсяг інформаційних потоків у мережі?
3. Які критерії розподілу файлів можуть бути використані в обчислювальній мережі із зіркоподібною, кільцевою та змішаною топологією?
4. Які обмеження виникають у моделі при оптимізації розміщення розподіленої бази даних?
5. Які різновиди запитів існують в мережі, що істотно відрізняються часом оброблення?
6. Які переваги та недоліки централізованого та децентралізованого способу збереження інформації у БД?
7. Як у мережі із топологією «зірка» навантаження на центральний комутатор залежить від місця розташування файлів на робочих станціях?
8. Які алгоритми та структури даних є найбільш доцільними при програмному вирішенні даної задачі?
9. Які промислові БД забезпечують розподілене зберігання інформації?
10. Що таке технологія реплікації БД?

ПРЕДМЕТНИЙ ПОКАЖЧИК

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Арлоу Д. UML 2 и унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Д. Арлоу, И. Нейштадт; пер. с англ. – [2-е изд.]. – СПб. : Символ-Плюс, 2007. – 624 с.
2. Байдачный С. SQL Server 2005. Новые возможности для разработчиков / Байдачный С., Маленко Д., Лозинский Ю. – М. : Солон-Пресс, 2006. – 208 с.
3. Бишоп Дж., Хорспул Н. C# в кратком изложении / Дж. Бишоп, Н. Хорспул. – М. : Бином. Лаборатория знаний, 2005. – 472 с.
4. Браст Э. Дж. Разработка приложений на основе Microsoft SQL Server 2005. Мастер-класс / Э. Дж. Браст, С. Форте; пер. с англ. – М. : Русская редакция, 2007. – 880 с.
5. Буч Г. Язык UML. Руководство пользователя / Буч Г., Рамбо Дж., Якобсон И. – М. : ДМК Пресс, 2006. – 496 с.
6. Ватсон Б. C# 4.0 на примерах / Б. Ватсон. – СПб. : БХВ-Петербург, 2011. – 608 с.
7. Вигли Э. Microsoft Mobile и .Net Compact Framework. Руководство разработчика / Вигли Э., Мот Д., Фут П. – СПб.: Питер, М. : Русская редакция, 2009. – 672 с.
8. Виейра Р. Программирование баз данных Microsoft SQL Server 2005 для профессионалов / Р. Виейра; пер. с англ. К. А. Птицына. – М. : Вильямс, 2008. – 1072 с.
9. Гамильтон Б. ADO.NET Сборник рецептов. Для профессионалов / Б. Гамильтон. – СПб. : Питер, 2005. – 576 с.
10. Гома Х. UML. Проектирование систем реального времени, параллельных и распределенных приложений / Х. Гома; пер. с англ. – М. : ДМК Пресс, 2011. – 704 с.
11. Дудзяний І. М. Об’єктно-орієнтоване моделювання програмних систем: [навч. посіб.] / І. М. Дудзяний. – Львів : Видавничий центр ЛНУ ім. Івана Франка, 2007. – 108 с.
12. Избачков Ю. С. Информационные системы: учебник для вузов / Ю. С. Избачков, В. Н. Петров. – [2-е изд.]. – СПб. : Питер, 2006. – 656 с.
13. Купін А.І. Деякі особливості новітніх мережних інформаційних технологій / А.І. Купін, І.О. Музика // Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій

та інформаційних технологій: IV міжнар. наук.-практ. конф., 19–21 вересня 2012 р.: тези доп. – Запоріжжя: ЗНТУ, 2012. – С. 172–173.

14. Лавинский Г. В. Теоретические основы автоматизации управления в экономических системах / Г. В. Лавинский, А. Д. Шарапов. – К. : Вища шк., 1988. – 178 с.

15. Магдануров Г. И. ASP.NET MVC Framework / Г. И. Магдануров, В. А. Юнев. – СПб. : БХВ-Петербург, 2010. – 320 с.

16. Малик С. Microsoft ADO.NET 2.0 для профессионалов / Сахил Малик; пер. с англ. А. А. Моргунова, А. А. Шило. – М. : Вильямс, 2006. – 560 с.

17. Музика І.О. Перспективи паралельних та розподілених обчислень на базі технології NVIDIA CUDA / І.О. Музика, Вільям Таді // Інтелектуальні технології в системному програмуванні (ІТСП-2014): ІІІ Всеукраїнська. наук.-практ. конф. молодих учених та студентів, 23–25 квітня 2014 р.: тези доп. – Хмельницький: ХНУ, 2014. – С. 99–100.

18. Нэш Т. C# 2010: ускоренный курс для профессионалов / Т. Нэш; пер. с англ. – М. : Вильямс, 2010. – 592 с.

19. Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы: [учебник для вузов. 4-е изд.] / В. Г. Олифер, Н. А. Олифер. – СПб. : Питер, 2010. – 944 с.

20. Петкович Д. Microsoft SQL Server 2008. Руководство для начинающих / Д. Петкович; пер. с англ. – СПб. : БХВ-Петербург, 2009. – 752 с.

21. Петцольд Ч. Программирование в тональности C# / Ч. Петцольд – М. : Русская Редакция, 2004. – 512 с.

22. Рамбо Дж. UML 2.0. Объектно-ориентироанное моделирование и разработка / Дж. Рамбо, М. Блаха. – [2-е изд.]. – СПб. : Питер, 2007. – 544 с.

23. Сандерсон С. ASP.NET MVC Framework / Стивен Сандерсон; пер. с англ. Н. А. Мухина. – М. : Вильямс, 2010. – 560 с.

24. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4 / Э. Троелсен; пер. с англ. – [5-е изд.]. – М. : Вильямс, 2011. – 1392 с.

25. Устинова Г. М. Информационные системы менеджмента / Г. М. Устинова. – СПб. : ДиаСофтЮП, 2000. – 357 с.

26. Файлер М. UML. Основы / М. Фаулер; пер. с англ. – [3-е изд.]. – СПб. : Символ-Плюс, 2004. – 192 с.

27. Фленов М. Е. Библия C# / Фленов М. Е. – [2-е изд. перераб. и доп.]. – СПб. : БХВ-Петербург, 2011. – 560 с.

28. Цегелик Г. Г. Системы распределенных баз данных / Г. Г. Цегелик. – Львов : Свит, 1990. – 167 с.

29. Цымбал В. П. Теория информации и кодирование / В. П. Цымбал. – К. : Вища шк., 1992. – 263 с.

30. Шилдт Г. C# 4.0: полное руководство / Г. Шилдт; пер. с англ. – М. : Вильямс, 2011. – 1056 с.

31. Aitchison A. Pro Spatial with SQL Server 2012 / Alastair Aitchison. – New York : Apress, 2012. – 554 p.

32. ASP.NET MVC 4 in Action / [Palermo Jeffrey, Bogard Jimmy, Hexter Eric and others]; foreword by Phil Haack. – Shelter Island : Manning Pablications Co., 2012. – 406 p.

33. Barker J. Beginning C# Objects: From Concepts to Code / J. Barker, G. Palmer. – New York : Apress, 2004. – 848 p.

34. Daniel S. Illustrated C# 2012. C# 5.0 presented clearly, concisely and visually / Solis Daniel. – [4th edition]. – New York : Apress, 2012. – 750 p.

35. Drayton P., Albahari B., Neward T. C # in a Nutshell, Second Edition. – O'Reilly, 2003. – 928 p.

36. C# и платформа .NET 4 для профессионалов / К. Нейгел, Б. Ивьен, Дж. Глини, К. Уотсон; пер. с англ. – М. : Вильямс, 2011. – 1440 с.

37. Liberty J. Learning C# / J. Liberty. – O'Reilly, 2002. – 368 p.

38. Patrick T. Microsoft ADO.NET 4. Step by Step / Tim Patrick. – California : O'Reilly Media, 2012. – 441 p.

39. Sells C. Windows Forms Programming in C# / C. Sells. – Addison-Wesley Professional, 2004. – 681 p.

40. Visual C# 2010: полный курс / К. Уотсон, К. Нейгел, Я. Педерсен, Дж. Рид, М. Скиннер; пер. с англ. – М. : Вильямс, 2011. – 960 с.

НАВЧАЛЬНЕ ВИДАННЯ

Андрій Іванович КУПІН
Іван Олегович МУЗИКА

МЕРЕЖНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

ПРАКТИКУМ
НАВЧАЛЬНИЙ ПОСІБНИК

Коректор Л. Б. Басюк
Комп'ютерний набір та верстка авторів

Підп. до друку 27.01.2015. Формат 60×84/16.
Папір офсетний 80 г/м^2.
Об'єм 15 ум. друкованих аркушів.
Друк ротаційний трафаретний, цифровий
Наклад 300 прим. Замовлення №27-01/15-02.

Видавництво «Діоніс» (ФО-П Чернявський Д.О.)
пр. 200 річчя Кривого Рогу, 17 (зуп. «Спаська»),
тел.: (056) 440-21-63; 404-05-92; 442-71-11.
Свідоцтво ДК 3449 від 02.04.2009 р.