МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни "ОСНОВИ ПРОГРАМУВАННЯ"

для студентів напряму підготовки 6.051501 "Видавничо-поліграфічна справа" всіх форм навчання

Харків. ХНЕУ ім. С. Кузнеця, 2015

Затверджено на засіданні кафедри комп'ютерних систем і технологій.
Протокол № 4 від 11.11.2014 р.

Укладач Браткевич В. В.

М 54 Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни "Основи програмування" для студентів напряму підготовки 6.051501 "Видавничо-поліграфічна справа" всіх форм навчання / уклад. В. В. Браткевич. – Х. : ХНЕУ ім. С. Кузнеця, 2015. – 112 с. (Укр. мов.)

Подано рекомендації до виконання лабораторних робіт із розробки типових програм мовою C#. Надано основний навчальний матеріал, наведено порядок виконання лабораторних робіт, запропоновано структуру матеріалу, який необхідно включати у звіт. Microsoft Visual Studio .NET (Ultimate 2012) обрано як середовище розробки.

Рекомендовано для студентів напряму підготовки 6.051501 "Видавничо-поліграфічна справа" всіх форм навчання.

Вступ

Навчальна дисципліна "Основи програмування" вивчається студентами напряму підготовки 6.051501 "Видавничо-поліграфічна справа" усіх форм навчання протягом третього семестру.

Метою викладання навчальної дисципліни "Основи програмування" є формування у студентів системи теоретичних знань і прикладних умінь в області застосування сучасних мов програмування для інструментальної підтримки технологічного процесу виробництва видавничо-поліграфічних і мультимедійних продуктів; підготовка студентів до самостійного освоєння вмонтованих сучасних програмних засобів (скриптів) розробки мультимедіа і Web-дизайну.

Основними завданнями вивчення дисципліни "Основи програмування" є оволодіння навичками в області застосування сучасних мов програмування для інструментальної підтримки технологічного процесу виробництва видавничо-поліграфічних і мультимедійних продуктів.

Предметом дисципліни є алфавіт, синтаксис і семантика мови C#, типові структури даних, а також середовище програмування Visual Studio.NET.

Результатами виконання лабораторних робіт є засвоєння студентами принципів процедурного і структурованого програмування; синтаксису і семантики мови C#; концепції типів даних та їх класифікації; типових алгоритми обробки чисельної інформації; технології розробки та налагоджування в середовищі Visual Studio.NET процедурних орієнтованих додатків мовою C#.

У результаті вивчення навчальної дисципліни студенти оволодіють навичками складання та налагоджування відповідних програм мовою C#, які забезпечують:

обробку лінійних процесів і процесів із розгалуженням та ітераціями;

реалізацію типових алгоритмів пошуку та сортування в одновимірних та двовимірних масивах;

використання вмонтованих функцій;

оголошення і використання функцій користувача;

користуватися раніше складеними програмами і здійснювати супровід програм, вносити зміни в програму, виконувати налагоджування програм за допомогою вбудованих інструментальних засобів.

Змістовий модуль 1. Організація програм

Лабораторна робота № 1

Інтегроване середовище системи програмування Visual Studio.NET

Мета роботи – ознайомлення з інтерфейсом, основними поняттями та можливостями системи програмування Visual C# .NET на прикладі виконання найпростіших програм.

Дана лабораторна робота має демонстраційний характер, тобто індивідуальне завдання відсутнє. Вона сприяє напрацюванню таких **компетентностей** відповідно до Національної рамки кваліфікацій:

знання:

основних компонентів інтерфейсу системи програмування Visual C# .NET;

порядку підготовки C# програми для подальшого виконання;

структури типової програми мовою C# у консольному та графічному виконаннях;

уміння:

налаштовувати систему програмування з урахуванням заданих вимог;

працювати з редактором тексту в середовищі Visual C# - NET;

виконувати компіляцію, налагодження і запуск готових демонстраційних програм;

отримати роздруківку вихідного тексту програми і результату її роботи;

комунікації:

аргументована взаємодія з клієнтами та замовниками під час вибору середовища розробки та виконання програмних продуктів;

рекомендації команді учасників проекту щодо налаштовування системи програмування Visual C# - NET;

автономність і відповідальність:

самостійне формулювання рекомендацій щодо оптимізації процесу підготовки програм до виконання;

прогнозування вигляду результатів виконання консольних та графічних програм у середовищі Visual C# .NET.

Основні положення

Базова технологія безпосередньо пов'язана з мовою C#, має назву .NET (вимовляється як "дот нет").

.NET – це загальний термін для багатьох служб, які надаються й використовуються під час створення та виконання програми на C#.

Особливості інфраструктури .NET-платформи.

C# повністю залежить від .NET і тому походження багатьох концепцій C# уходить своїми коренями в. NET.

Далі перераховані особливості інфраструктури .NET-платформи:

.NET надає засоби для виконання інструкцій, що містяться в програмі, яка написана мовою C#. Ця частина .NET називається середовищем виконання (execution engine).

.NET допомагає реалізувати так зване середовище, безпечне до невідповідності типів даних (type safe environment).

.NET звільняє програміста від стомлюючого процесу і такого, що нерідко призводять до помилок у ході керування комп'ютерною пам'яттю, яка використовується програмою.

До складу .NET-платформи входить бібліотека, котра містить масу готових програмних компонентів, які можна використовувати у власних програмах. Вона заощаджує чимало часу, тому що програміст може скористатися готовими фрагментами коду. Фактично він повторно використовує код, створений та ретельно перевірений професійними програмістами Microsoft.

У .NET спрощена підготовка програми до використання (розгортання).

.NET забезпечує перехресну взаємодію програм, написаних різними мовами. Будь-яка мова, що підтримується .NET, може взаємодіяти з іншими мовами цієї платформи.

У даний момент на платформу .NET перенесено близько 20 мов. Оскільки для виконання коду, написаного будь-якою мовою, що підтримується платформою .NET, використовується те саме середовище виконання, його часто називають єдиним середовищем виконання (Common Language Runtime, CLR).

Програма, під час створення якої була передбачена можливість повторного використання, називається компонентом (програмним компонентом).

Мови програмування та компілятори.

Програмувати на перших комп'ютерах, виготовлених у сорокових роках минулого сторіччя, було дуже непросто. Для цього використовувалася машинна мова, що складалася з послідовностей бітів, які прямо керували простими діями процесора. Програмування на рівні машинної мови – заняття надзвичайно трудомістке та стомлююче.

Незабаром програмісти почали шукати альтернативи машинній мові, більш близькі до людської мови, щоб підвищити продуктивність своєї праці.

Першим результатом пошуків стали так звані асемблери – мови, в яких використовувалися більш зрозумілі людині команди, наприклад move, getint або putint. Але, незважаючи на те, що асемблери були трохи простіші для читання й розуміння, їх поєднувала з машинним кодом одна важлива загальна риса: розроблювач під час написання програми повинен був мислити термінами низькорівневих операцій процесора і пам'яті.

Однак еволюція комп'ютерів і зростаючі потреби у все більш складних програмах привели до появи повністю машинно-незалежних мов програмування. Першою з них стала FORTRAN, створена в середині п'ятдесятих років. Незабаром з'явилися інші мови високого рівня. Сьогодні їхня кількість за деякими оцінками перевищує дві тисячі.

Одним із останніх доповнень у родині мов високого рівня стала C#. Однак як би високо людина не відходила від базових інструкцій процесора, їй, як і раніше, потрібен машинний код, який апаратне забезпечення комп'ютера може розуміти, а виходить, ще й виконувати.

Традиційно перетворення вихідного коду, написаного мовою високого рівня, в машинний код здійснювали системні програми, які називаються *компіляторами*.

На рис. 1 наведено ілюстрацію того, як вихідний код типової мови високого рівня перетворюється у програму, що виконується.

Написаний текст, який містить інструкції мови високого рівня, називається *вихідним кодом*.

У випадку C# цей вихідний код зберігається в файлі з розширенням .cs.

Результатом компіляції стає *програма, що виконується*, яка складається з інструкцій машинної мови.

Рис. 1. **Традиційний процес компіляції**

Компіляція в .NET вихідного коду C#.

Традиційний процес компіляції вихідного коду, написаного мовою програмування високого рівня, в програму, що виконується, мав кілька недоліків. Найбільш істотними з них є такі.

Недолік 1. Для конкретної апаратної платформи, що характеризується типом процесора тощо, потрібен свій компілятор, оскільки в кожної з них своя машинна мова. Відповідно, якщо потрібно виконувати програму, написану на FORTRAN, на чотирьох комп'ютерах із процесорами різних виробників, буде потрібно чотири різних компілятори FORTRAN. Більш того, кожного разу, коли виробник апаратного забезпечення випускає нове покоління своїх процесорів, у компілятор доводиться вносити зміни та доповнення.

Недолік 2. У більшості програмістів є улюблена мова програмування, якій вони віддають перевагу перед усіма іншими. Можливо, кращим рішенням було б дозволити кожному члену команди писати своєю улюбленою мовою, але це нелегко, якщо дотримуватися процесу компіляції, який розглянуто на рис. 1.

Різні мови на машинному рівні реалізують функціональність різними способами. Частково це залежить від особливостей компіляторів. У свою чергу, це унеможливлює взаємодію різних мов між собою.

Цю проблему були покликані вирішити так звані компонентні системи (такі, як CORBA і COM). Вони обумовлювали стандарти взаємодії між різними частинами програм.

Програміст A писав компонент X, наприклад, мовою Visual Basic, і цей компонент міг взаємодіяти з компонентом Y, що був розроблений програмістом B на C++.

Компонентні системи мали комерційний успіх. Однак їхнє поширення викликало ряд інших проблем, однією з яких стала неможливість забезпечити взаємодію "зовнішнього" компонента з іншими частинами програми на такому ж рівні, якби всі частини програми були написані однією мовою.

У C# і .NET реалізовані рішення описаних двох проблем. Слід розглянути всі складові процесу компіляції програми в .NET (рис. 2).

Насамперед, варто звернути увагу на появу на рис. 2 ще двох мов – C++ і Visual Basic. І поза залежністю від того, якою мовою (з підтримуваних .NET) написана програма, це ніяк не впливає на процес її компіляції в .NET.

Після того, як написаний вихідний код, його потрібно відкомпілювати в машинний код. Однак спочатку він компілюється в іншу мову, що нази-

вається Microsoft Intermediate Language (MSIL). Більш того, всі компілятори, орієнтовані на .NET-платформу, повинні генерувати на виході код даної проміжної мови MSIL.

Як ясно з назви, MSIL є проміжною ланкою між мовами високого рівня (вихідний код) і машинними мовами (називаними також природним кодом).

Код MSIL можна швидко та ефективно транслювати в машинну мову за допомогою JIT-компілятора (Just in Time-Compiler).

Код, що генерується JIT-компілятором, нічим не відрізняється від машинного коду, що генерується звичайним компілятором, однак JIT-компілятор використовує трохи іншу стратегію. Замість того, щоб інтенсивно використовуючи пам'ять і, затрачаючи значний час, перетворити в машинний код відразу весь код MSIL, він компілює в машинний код лише ті частини додатка, які реально необхідні в даний момент. У результаті код компілюється "на ходу", безпосередньо перед виконанням, і JIT-компілятор не витрачає час на компіляцію MSIL-коду, що не використовується.

Переваги архітектури .NET.

1. Під час введення MSIL (рис. 2) між мовою високого рівня та машинною мовою фактично відокремлюються мови одна від одної.

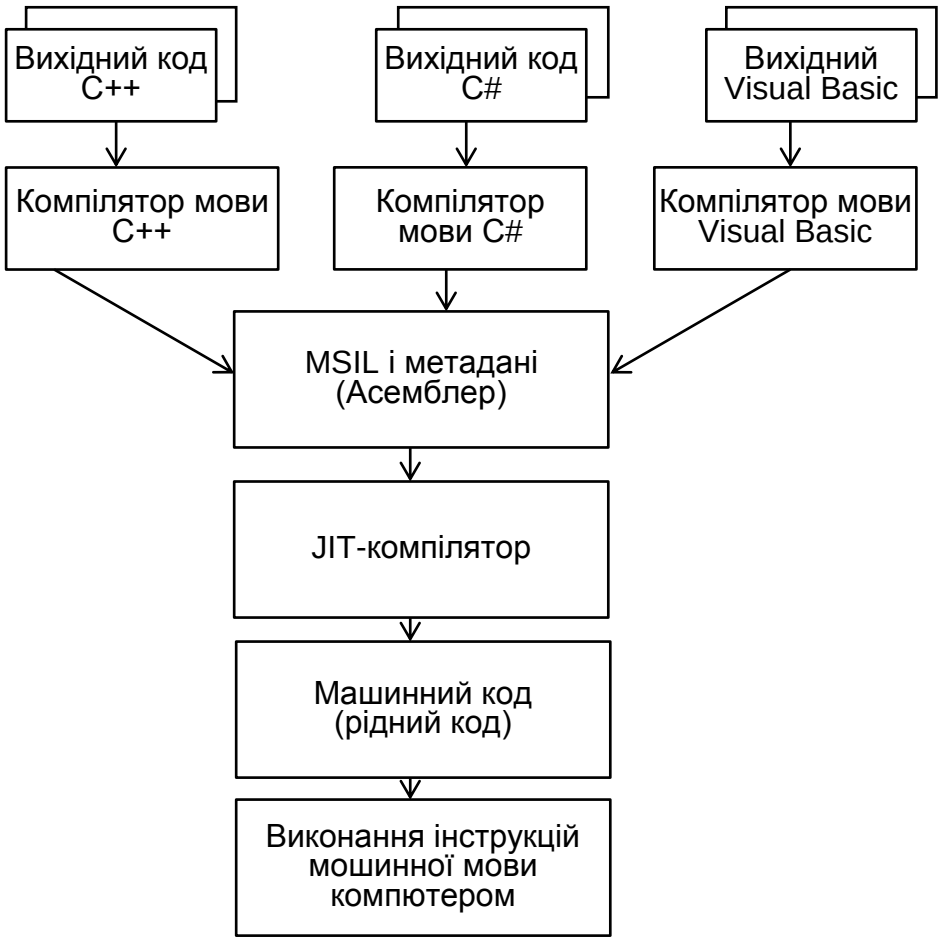


Рис. 2. **Процес компіляції в .NET**

Код MSIL залишається незмінним, на якій би апаратній платформі він не використовувався. Єдиним машинно-залежним елементом є JIT-компілятор, і у ході зміни обладнання лише він має потребу в модифікації.

На кожному комп'ютері застосовується свій JIT-компілятор, що перетворює код MSIL у машинний код, сумісний з даною конкретною конфігурацією. В результаті все, що потрібно, – це компілювати код, написаний мовою високого рівня, в універсальний код, мова якого залишається незмінною. Це і є рішенням згаданої першої проблеми.

Варто розглянути блок "MSIL і метадані", наведений на рис. 2. Термін "*метадані*" можна перевести як "дані про дані". Метадані генеруються компілятором мови високого рівня і містять докладний опис елементів вихідного коду. Опис цей настільки докладний, що вихідний код інших мов зможе використовувати даний код так, якби він був написаний тією ж мовою. Тепер програмісти, що пишуть мовою C++, C# і Basic, реально зможуть працювати в рамках одного проекту і, отже, у такий спосіб долається другий недолік.

Важливо знати про існування MSIL, але в повсякденному програмуванні зіштовхуватися з ним прямо не доводиться. Зазвичай застосовуються дві команди – одна компіляції програми в MSIL-код і метадані, а інша – для виконання програми (при цьому буде викликатися JIT-компілятор). Фактично виконання програми – це виконання кінцевого результату роботи компіляторів. MSIL у цьому процесі залишається "невидимим" для користувача.

Основні елементи інтерфейсу Visual Studio .NET (Ultimate 2012) містять величезний набір інструментів, покликаних спростити життя програмісту. У даній роботі не буде зроблено огляд усіх можливостей середовища Visual Studio .NET, проте слід зазначити основні елементи. Крім того, багато пунктів меню і керуючі вікна будуть описані далі у міру виконання поточних лабораторних робіт.

Стартова сторінка.

Для запуску Visual Studio .NET слід вибрати пункт меню Пуск / Програми / Microsoft Visual Studio .NET / Microsoft Visual Studio .NET.

На екрані з'явиться стартова сторінка Visual Studio Ultimate 2012, фрагмент якої зображено на рис. 3.

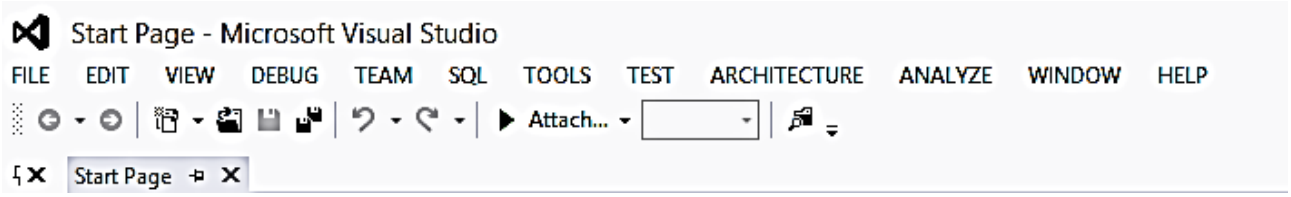


Рис. 3. **Фрагмент стартової сторінка Visual Studio .NET (Ultimate 2012)**

Visual Studio.NET — це не тільки середовище для розробки додатків мовою C#. Воно дозволяє створювати додатки мовами VB, C#, C ++, формувати Setup (інсталяційний пакет) програм та багато іншого.

Для того щоб реально побачити, як створюється новий проект у Visual Studio .NET слід вибрати пункт меню Start / New / Project. Після його виклику з'явиться вікно, аналогічне зображеному на рис. 4.

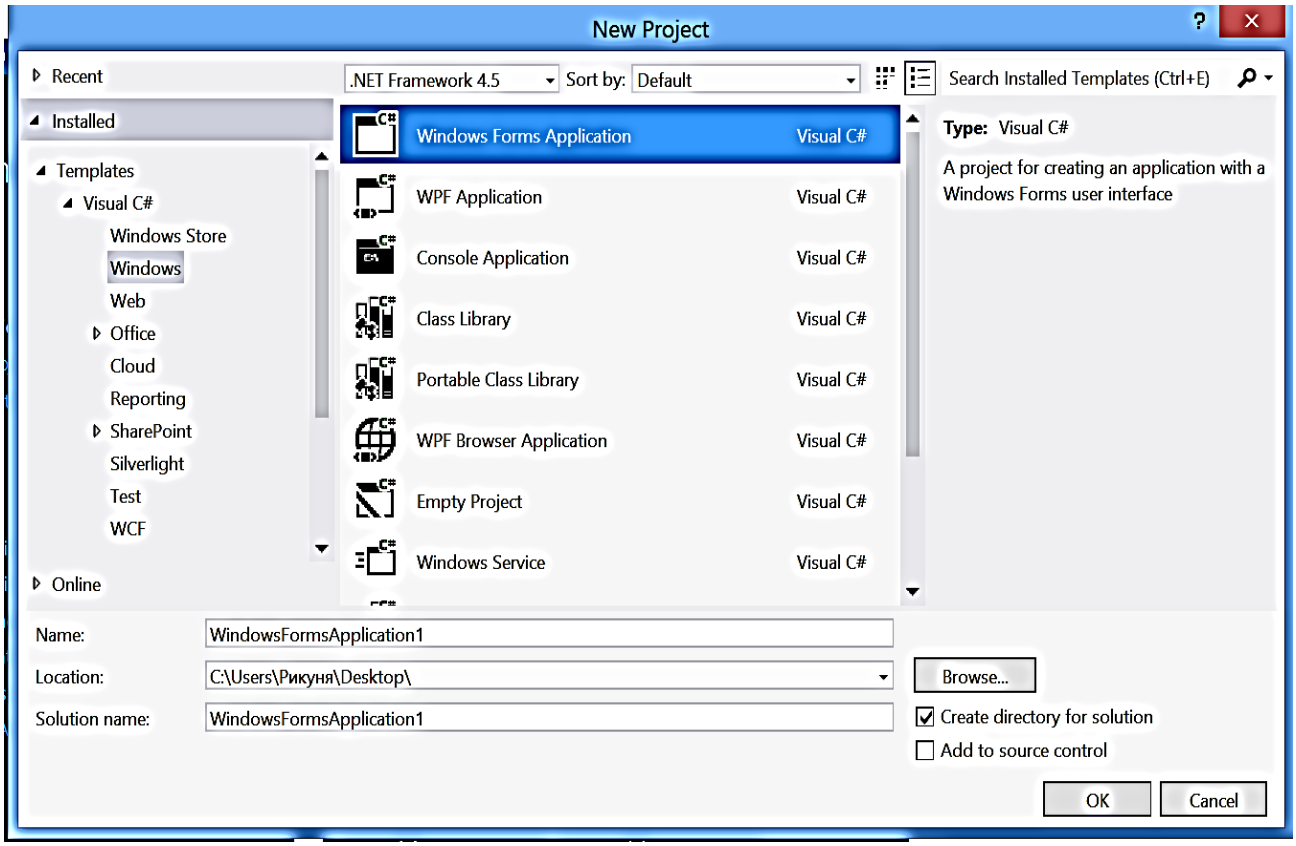


Рис. 4. **Вікно вибору типу проекту**

Тут можна вибрати потрібну мову програмування (в лівій частині вікна) або якийсь спеціальний майстер створення додатків — цей список може поповнюватися інструментами незалежних розробників. Оскільки вивчається мова програмування C#, слід вибрати пункти Visual C# / Windows.

У правій частині вікна потрібно вказати тип створюваного проекту. Це може бути Windows-додаток (Windows Form Application), додаток для Інтернет (WPF Browser Application), консольний додаток (Console Application) і деякі інші.

Вибрати в лівій частині вікна пункт Windows Application, а в правій частині — Windows Form Application. Крім того, можна вказати назву

створюваного проекту і шлях до каталогу, в якому він буде розташовуватися. Натиснути ОК.

Тепер можна побачити основні частини візуального середовища розробки проекту. Вони зображені на рис. 5.

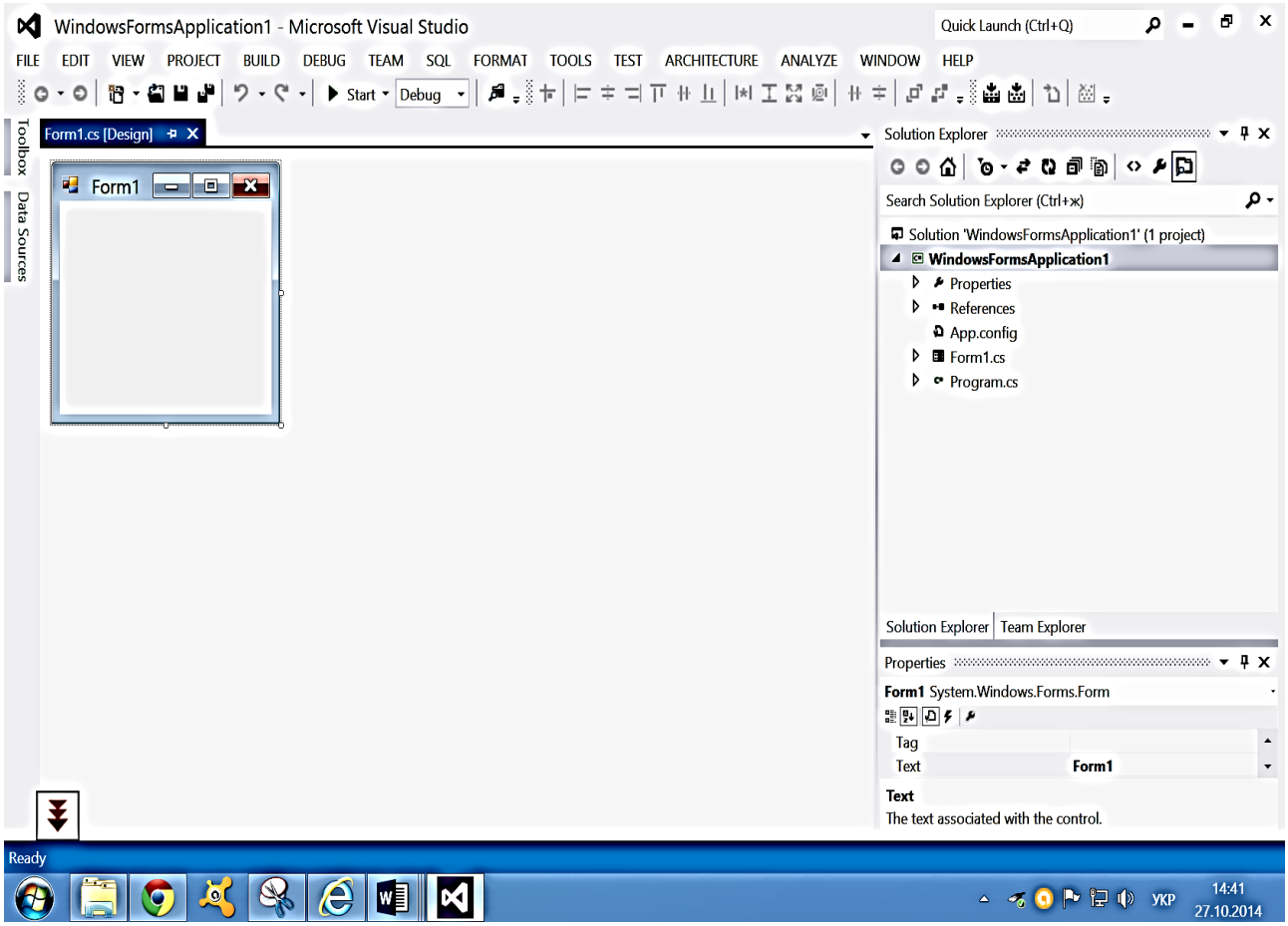


Рис. 5. **Основні частини візуальної середовища розробки**

У центрі знаходиться головне вікно для створення візуальних форм і написання коду.

Праворуч розміщується вікно Solution Explorer для управління проектами і вікно властивостей Properties.

Solution Explorer.

Solution Explorer дозволяє управляти компонентами, включеними в проект.

Наприклад, для того щоб додати в проект нову форму, слід встановити курсор на опцію WindowsApplication_ у вікні Solution Explorer і вибрати в контекстному меню, що відкривається кліком правої кнопки миші, пункти Add / Add Windows Form (рис. 6). Далі (рис. 7) потрібно вибрати тип компонента, який потрібно додати (у даному разі Windows Form) і натиснути кнопку Add.

Про інші пункти цього меню йтиметься в наступних лабораторних роботах. Крім контекстного меню проекту існує ще ряд контекстних меню, що дозволяють управляти окремими елементами проекту.

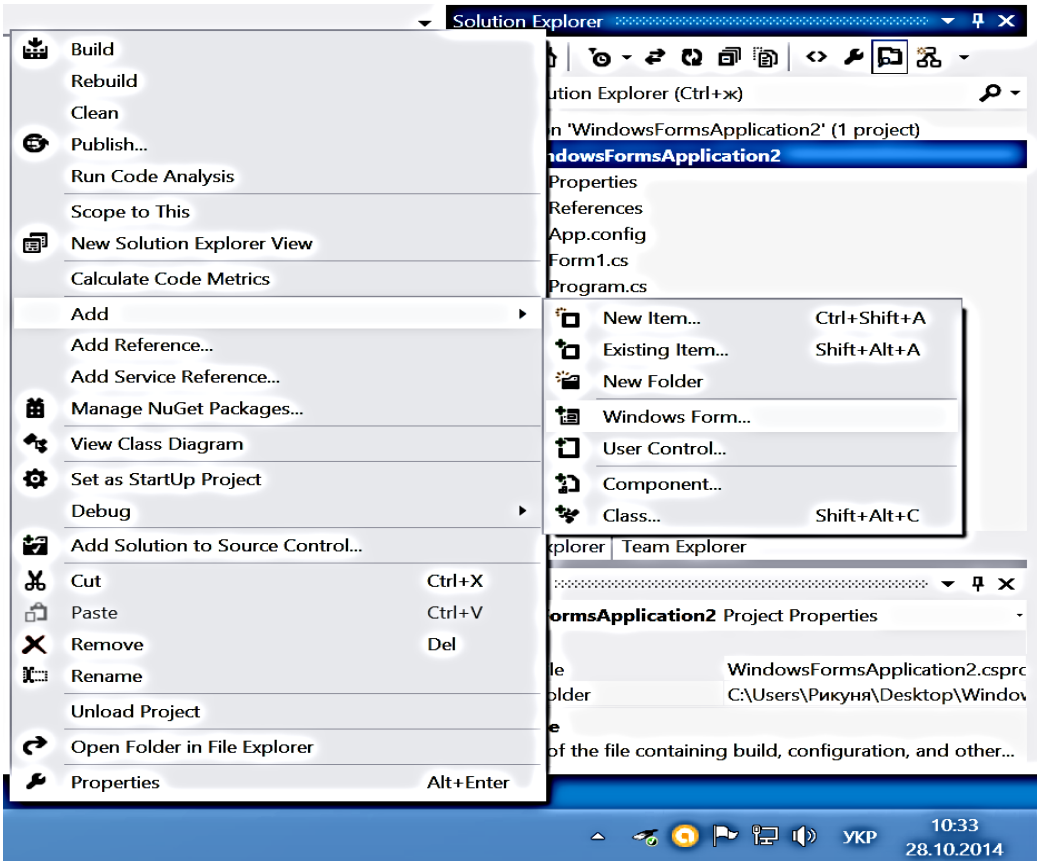


Рис. 6. **Контекстне меню для додання в проект нового компонента**

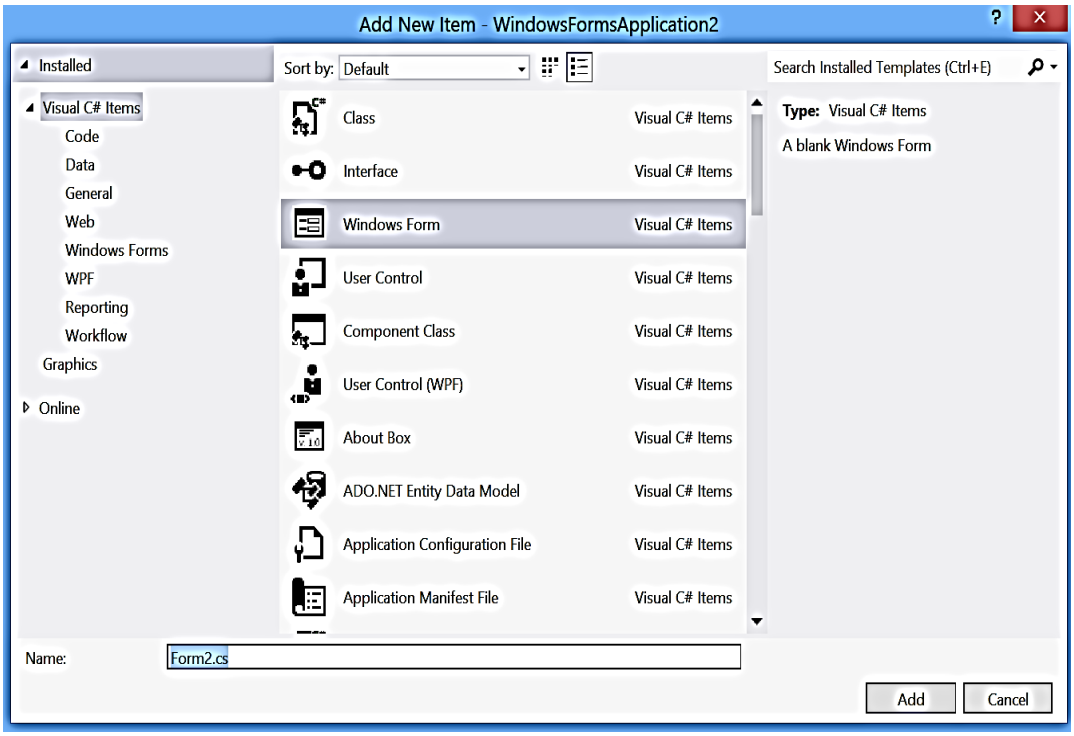


Рис. 7. **Додання в проект нової форми**

Так, щоб переключитися з вікна дизайнера в вікно коду проекту, виберіть в контекстному меню для Form1.cs у вікні Solution Explorer (див. рис. 5) пункт View Code.

Відповідно, щоб переключитися назад — View Designer. Контекстне меню є у кожного елемента дерева проекту. Використання контекстного меню — це швидкий інтерактивний спосіб навігації по проекту.

Class View.

Це вікно дозволяє переміщуватися по всіх елементах програмного проекту, включаючи окремі процедури.

За допомогою Class View можна додавати нові методи, класи, дані.

Кожен елемент дерева проекту має контекстне меню. Якщо вікно Class View відсутній на екрані, слід вибрати пункт меню View / Class View.

Properties Explorer.

Це вікно дозволяє працювати з властивостями форм і їх компонентів. Properties Explorer містить список усіх властивостей обраного в поточний момент компонента.

Наприклад, якщо вибрати форму створеного додатка, вікно Properties Explorer набуде вигляду, який наведено на рис. 8. Тут подано два списки: список усіх властивостей форми і їх значень.

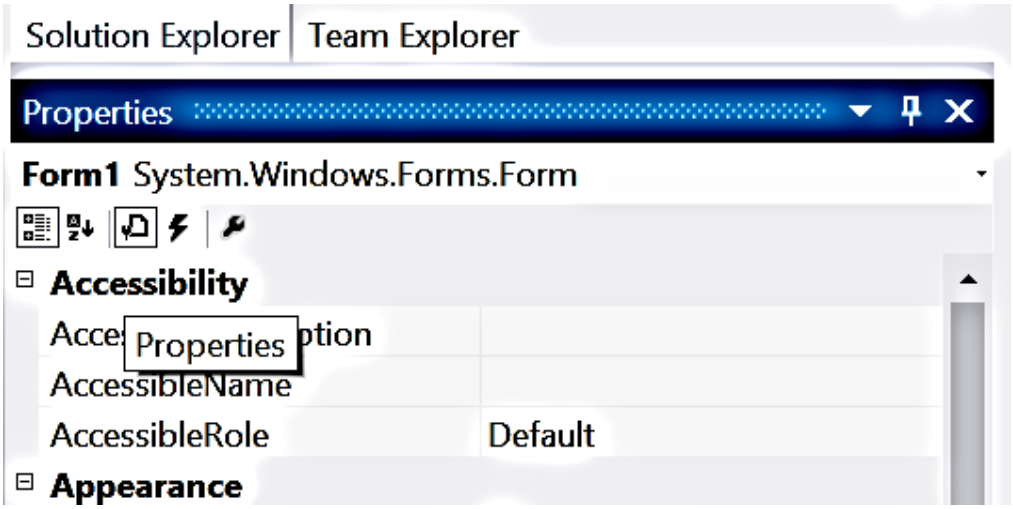


Рис. 8. **Список усіх властивостей форми і їх значень**

При цьому значення властивостей можуть бути подані в будь-якій формі, наприклад, як текстове поле, як випадок допустимі значення, як вікно вибору кольору і т. д. Якщо змінити значення властивості за замовчуванням, то воно буде виділено жирним кольором. У цьому випадку контроль за змінами, що вносяться в проект, стане більш наочним.

Крім того, Properties Explorer дозволяє сортувати якості або за алфавітом, або за належністю до певної групи.

Другим важливим завданням, яке виконує Properties Explorer, є управління подіями. Для того щоб переключитися на закладку подій, слід натиснути кнопку із зображенням блискавки вгорі вікна (рис. 9).

Вікно подій дозволяє налаштовувати реакцію форми або компонента на різні дії з боку користувача або операційної системи, наприклад створити обробник подій від миші або клавіатури. У лівій частині вікна міститься список всіх доступних подій, а в правій — імен методів, що обробляють події. За замовчуванням список методів порожній. Можна додати новий обробник, вписавши ім'я методу в відповідному полі, або створити обробник з ім'ям за замовчуванням, клацнувши два рази по комірці лівою кнопкою миші.

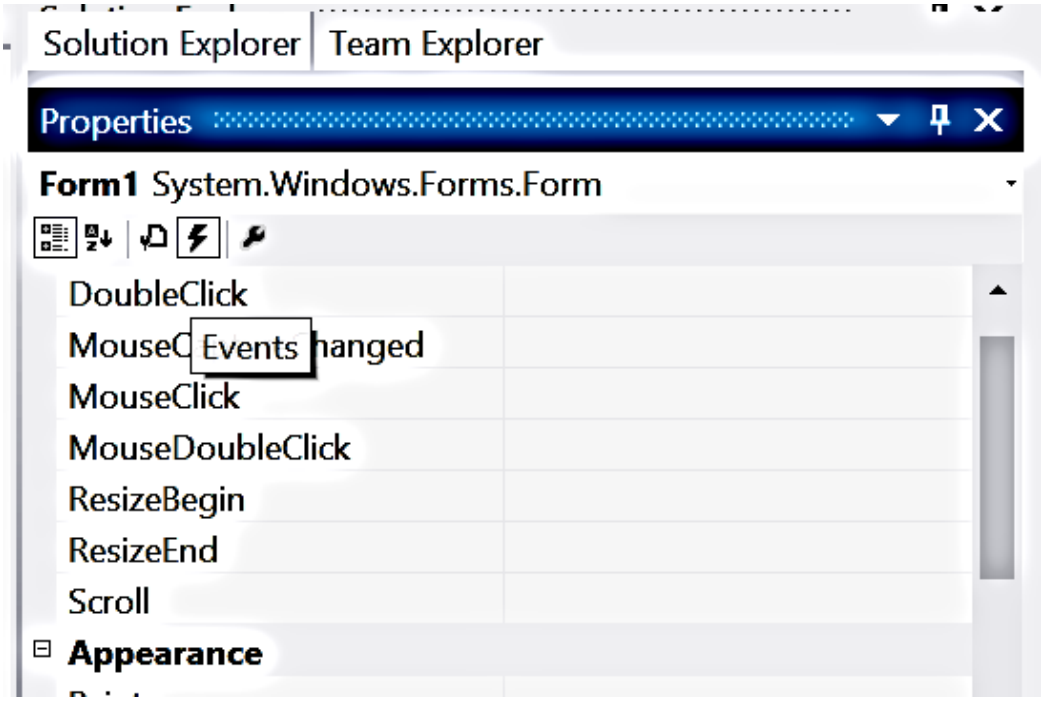


Рис. 9. **Список реакцій форми або компонента на різні дії**

Toolbox.

Це вікно містить Windows Forms компоненти, які можна розмістити на своїй формі. Якщо такого вікна у Visual Studio немає, виберіть в головному меню пункт View / Toolbox.

Вікно (рис. 10) візуально відображає найбільш часто використовувані .NET компоненти для створення додатків Windows.

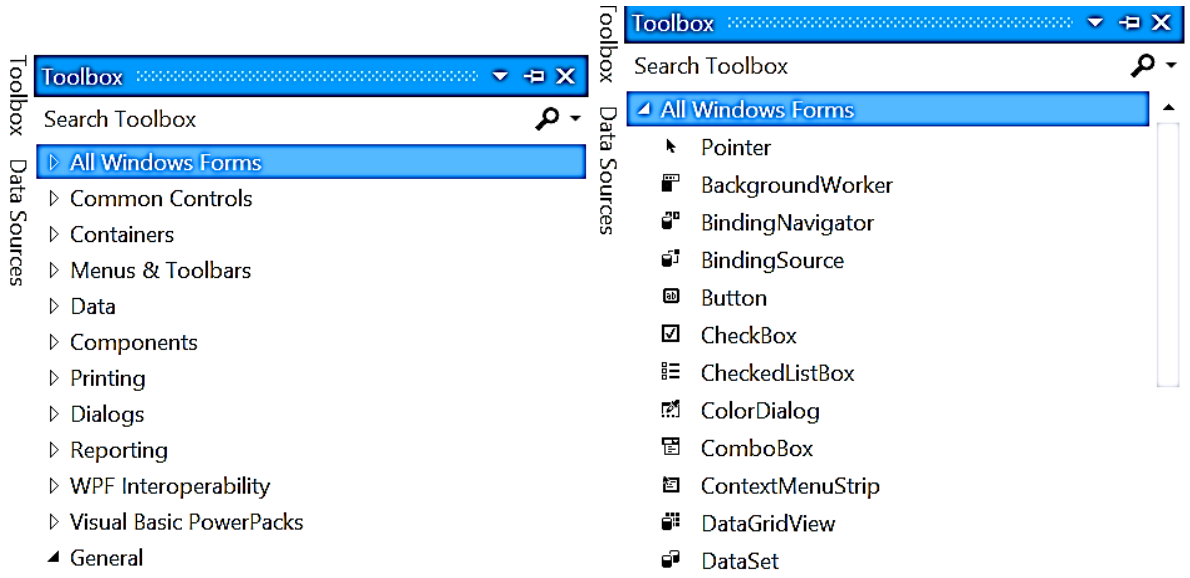


Рис. 10. **.NET компоненти для створення додатків Windows**

Toolbox має кілька закладок. Наприклад, закладка All Windows Forms включає візуальні елементи управління, такі як кнопки, списки, дерева. Закладка Data присвячена базам даних. Закладка Components містить невізуальні компоненти, найбільш представницьким серед яких є Timer.

Візуальні властивості допоміжних вікон.

Усі візуальні вікна вони можуть "прилипати" до будь-якої сторони головного вікна Visual Studio .NET.

Візуальні вікна можуть ховатися ппід час втрати активності. Для того, щоб наділити цією властивістю, наприклад, Solution Explorer, слід вибрати у контекстному меню цього вікна пункт Auto Hide або натиснути відповідну кнопку поруч із кнопкою заголовка "Закрити".

Щоб повернути вікно в первинний стан, варто просто клацнути лівою кнопкою миші по відповідній назві в панелі.

Меню і панель інструментів.

Усі дії, які можна виконувати в середовищі Visual Studio .NET, розташовуються в головному меню. Головне меню має контекстну залежність від поточного стану середовища, тобто містить різні пункти залежно від того, чим зараз займається користувач і в якому вікні він знаходиться.

Крім того, більшість пунктів меню продубльовано в панелі інструментів.

Visual Studio .NET має безліч панелей інструментів. Можна включити або виключити панель інструментів за допомогою меню View / Toolbars.

Ті панелі інструментів, які вже відкриті, позначені в меню "пташками". Також можна створювати власні панелі інструментів, скориставшись пунктом цього ж меню Customize.

Головне меню Visual Studio .NET.

Меню Visual Studio .NET знаходиться у верхній частині середовища. В меню є всі команди, призначені для виконання дій над елементами проектів. Пункти меню бувають командними і груповими (що містять інші пункти меню).

Назва кожного групового пункту меню відображає команди, що містяться в ньому. Наприклад, меню File містить команди, призначені для роботи з файлами проекту. Деякі пункти меню включають вкладені пункти з більш докладними командами. Наприклад, команда New з меню File показує меню вибору типів файлів. Найбільш часто вживані пункти меню мають "гарячі" клавіші. Так, для створення нового файла потрібно натиснути клавіші CTRL + N.

Основні пункти головного меню Visual Studio .NET буде розглянуто у міру виконання лабораторних робіт.

Перелік можливостей Visual Studio .NET.

Далі наведено короткий перелік можливостей Visual Studio .NET (VS), завдяки яким ця система є найбільш привабливим засобом розробки в .NET.

1. VS автоматично виконує всі кроки, необхідні для компіляції вихідного коду, й одночасно дозволяє управляти всіма використовуваними опціями, якщо буде побажання їх перевизначити.

2. Текстовий редактор VS налаштований для роботи з тими мовами, які підтримуються VS (включаючи C#), тому він може інтелектуально виявляти помилки і підказувати в процесі введення, який саме код необхідний.

3. До складу VS входять програми, що дозволяють створювати додатки в Windows Forms і Web Forms шляхом простого перетягування мишею елементів користувальницького інтерфейсу.

4. Багато типів проектів, створення яких можливе на C#, можуть розроблятися на основі "каркасного" коду, який заздалегідь включається в програму, замість того, щоб щоразу починати з нуля,

5. VS дозволяє використовувати вже наявні файли з вихідним кодом, що зменшує тимчасові витрати на створення проекту.

6. До складу VS входить кілька допоміжних програм, які дозволяють автоматизувати виконання найбільш поширених завдань; причому бага-

то з цих програм можуть додавати необхідний код у вже існуючі файли, так що програмісту не доведеться турбуватися (а в деяких випадках і взагалі згадувати) про дотримання синтаксичних правил.

7. VS має велику кількість потужних інструментів, завдяки яким можна переглядати окремі елементи проекту або здійснювати в них пошук, незалежно від того, чи є ці елементи файлами з кодами мовою C# або становлять будь-які інші ресурси, наприклад, двійкові графічні або звукові файли.

8. Поширювати додатки в VS настільки ж просто, як і писати їх.

9. VS полегшує передачу коду клієнтам і дозволяє їм інсталювати його без будь-яких проблем.

10. VS допускає використання досконалих методів налагодження у процесі розробки проектів: наприклад, покрокове виконання коду, коли виконується один оператор за раз, що дає можливість стежити за поточним станом додатка.

Порядок виконання лабораторної роботи

Проект 1. Створення простого консольного додатка.

Буде використовано консольні додатки впродовж всього лабораторного практикуму першого семестру.

Процес створення консольного додатка складається з таких кроків.

1. Вибрати пункт меню File / New / Project.

2. У вікні New Project встановити відповідні (рис. 11) настройки: Visual C# / Windpws; тип проекту Console Application.

3. У полі Location вказати шлях до папки, в якій буде створено проект (якщо цієї папки не існує, вона буде створена автоматично).

4. У поле Name записати ім'я проекту (наприклад, Lab_1-1_Fandorin, або залишити без змін, див. рис.11).

4. Клацніть по кнопці ОК.

5. Після того, як шаблон проекту буде створено, додати в файл, виведений в основному вікні (рис. 12), рядок коду.

```
Console.WriteLine("Моя перша програма!");
```

6. Вибрати пункт меню Debug / Start Without Debugging (або натиснути Ctrl + F5).

Через кілька секунд з'явиться наступне вікно (рис. 13).

На даному етапі не буде проаналізовано сам код, оскільки поки становить інтерес власне використання VS для введення і запуску програми.

Як можна помітити, VS робить величезний обсяг роботи, істотно спрощуючи процес компіляції і виконання коду. Однак навіть ці прості кроки можна виконувати різними способами.

Так, наприклад, створення нового проекту може бути здійснено за допомогою пункту меню File / New / Project (як було показано), за допомогою натиснення клавіш Ctrl + Shift + N або клацанням мишею по відповідному значку на панелі інструментів.

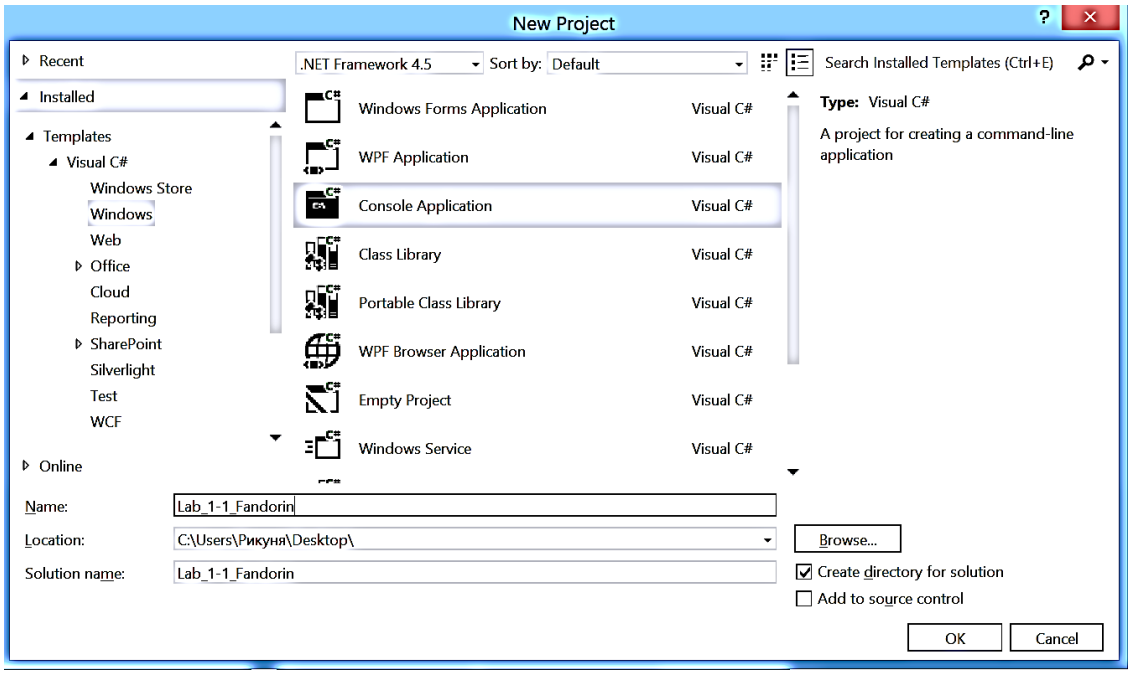


Рис. 11. **Вікно New Project для створення простого консольного додатка**

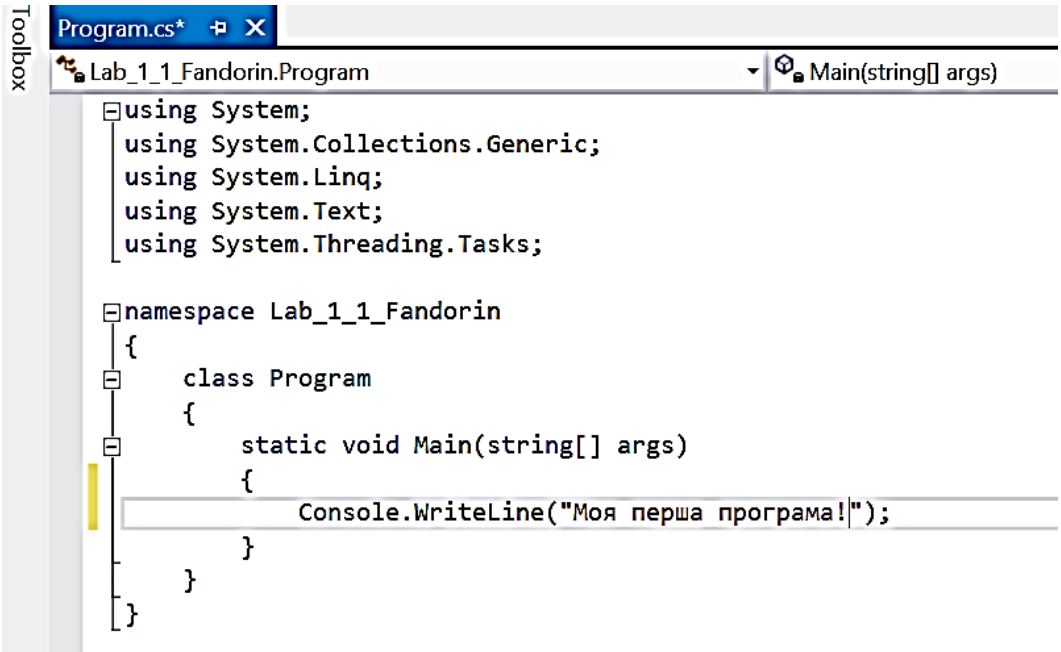


Рис. 12. **Перша консольна програма**

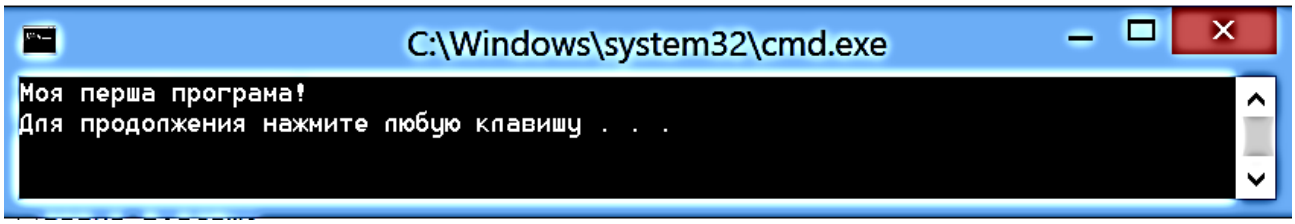


Рис. 13. **Результат виконання першої консольної програми**

Дана програма також може бути відкомпільована і виконана кількома різними способами. Метод, який був використаний, вибравши пункт меню Debug / Start Without Debugging, має два спрощених способи виклику: з клавіатури (Ctrl + F5) і за допомогою іконки на панелі інструментів. Крім цього, існує можливість запускати код у режимі налагодження за допомогою пункту меню Start Debugging (той же результат можна отримати під час натискання клавіші F5).

Після того, як процес компіляції закінчився, можна виконати згенерований файл із розширенням .exe, запустивши його з директорії, зазначеної (для розглянутого прикладу) на рис. 14.

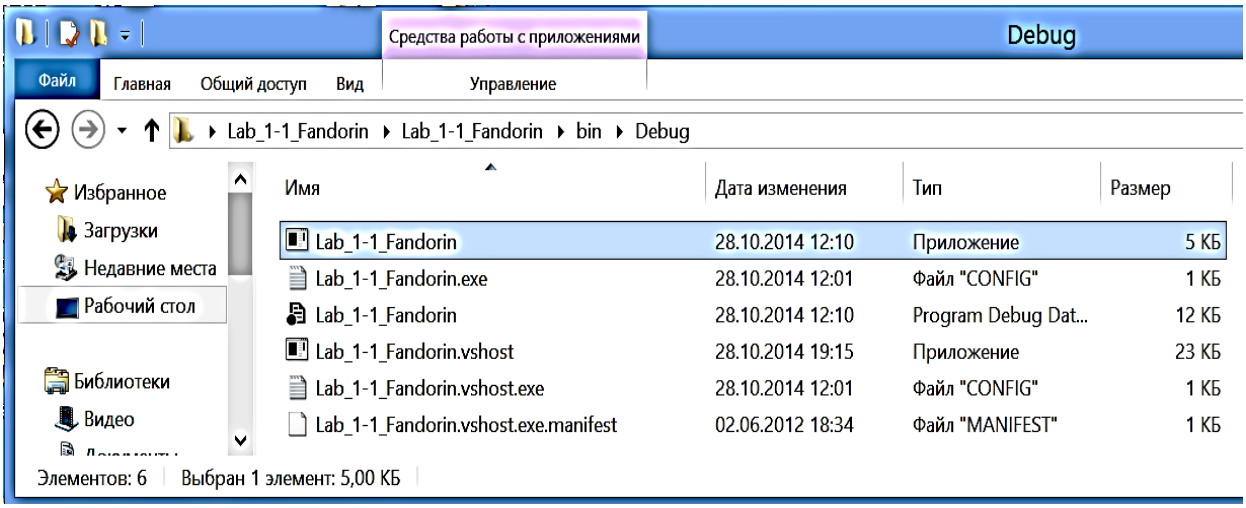


Рис. 14**. Директорія з файлами, які отримано в результаті компіляції проекту**

Слід познайомитися більш детально з деякими аспектами середовища розробки.

Аналіз структури проекту з консольним додатком.

Перше вікно, яке буде розглянуто, це вікно Solution Explorer (рис. 15), яке розташоване в правому верхньому куті екрану (див. рис. 5).

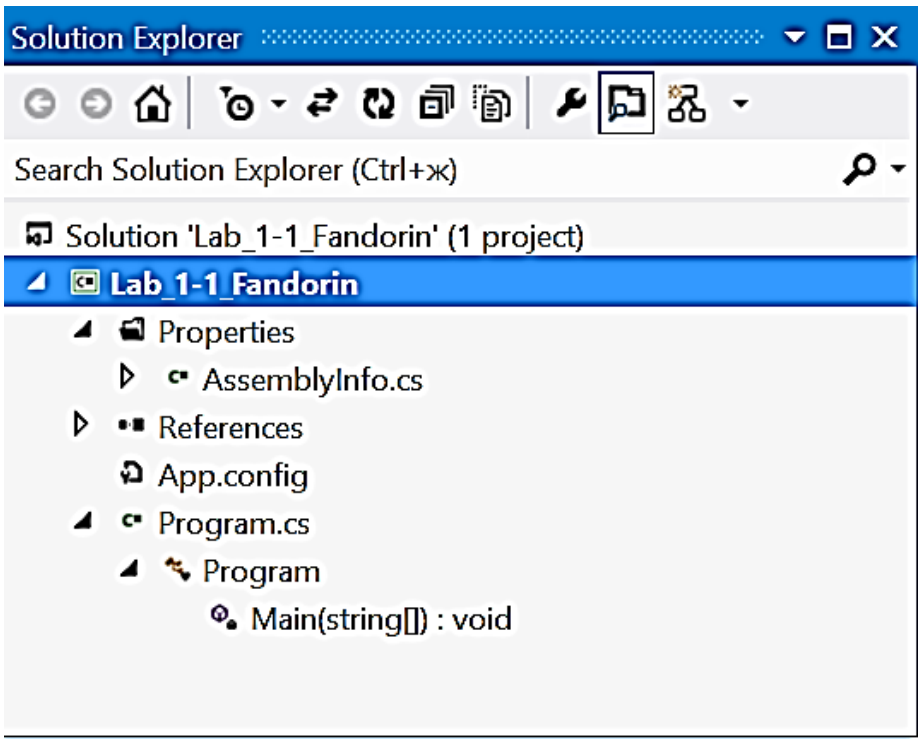


Рис. 15. **Вікно Solution Explorer консольного додатка**

У цьому вікні відображаються файли, з яких складається проектLab_1-1_Fandorin.

Поряд із файлом Program.cs, в який додано код, там показаний ще один програмний файл — AssemblInfo.cs. Усі файли, в яких містяться коди на C #, мають розширення .cs. Цей файл поки не повинен потребувати уваги; в ньому міститься додаткова інформація про проект.

Можна використовувати це вікно для внесення змін у код, який виводиться в головному вікні; для цього слід або двічі клацнути мишею по потрібним .cs-файлам, а потім вибрати в контекстному меню View Code, або спочатку виділити їх, а потім натиснути кнопку панелі інструментів. У цьому ж вікні можна провести ряд інших операцій з цими файлами, наприклад, перейменувати їх або видалити з проекту. Тут можуть з'являтися й інші типи файлів, наприклад, ресурси проектів (ресурсами називаються файли, які використовувані проектом, але не є файлами C#, наприклад, виконавчі файли зображень або звукові файли). Можна працювати з ними за допомогою того ж самого інтерфейсу.

Папка References (посилання) містить список тих бібліотек, які використовуються в даному проекті. Це теж стосується того, що буде розглянуто далі, оскільки поки достатньо стандартних посилань.

Вікно Output.

Вікно Output слугує для виведення повідомлень звіту про результат компіляції. Для виклику вікна слід виконувати команди VIEW / Output.

На рис. 16 подано вигляд вікна після успішного закінчення процесу компіляції.

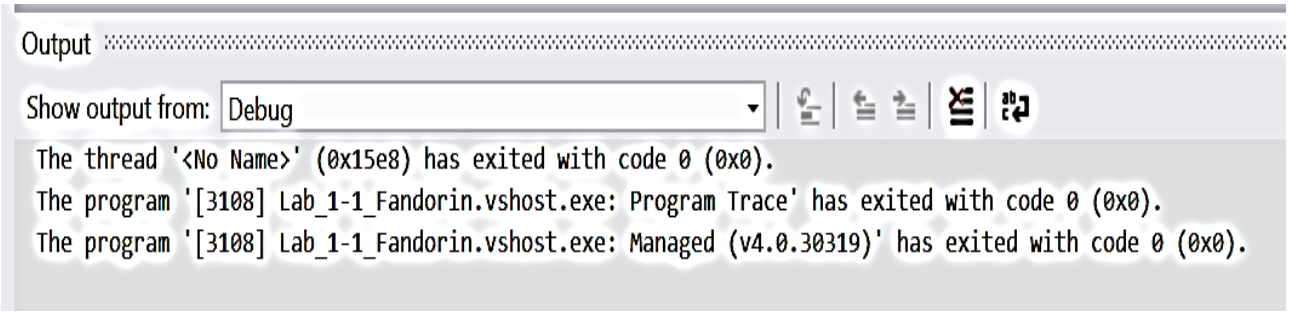


Рис. 16. **Вікно Output після успішного закінчення процесу компіляції**

Це звіт про компіляції файлів. У цьому ж звіті розміщуватимуться повідомлення про всі помилки, які виникли в процесі компіляції.

Для наочності необхідно спробувати прибрати крапку з комою з рядка, яку було додано в попередньому розділі, і відкомпілювати програму заново. Результат наведено на рис. 17.

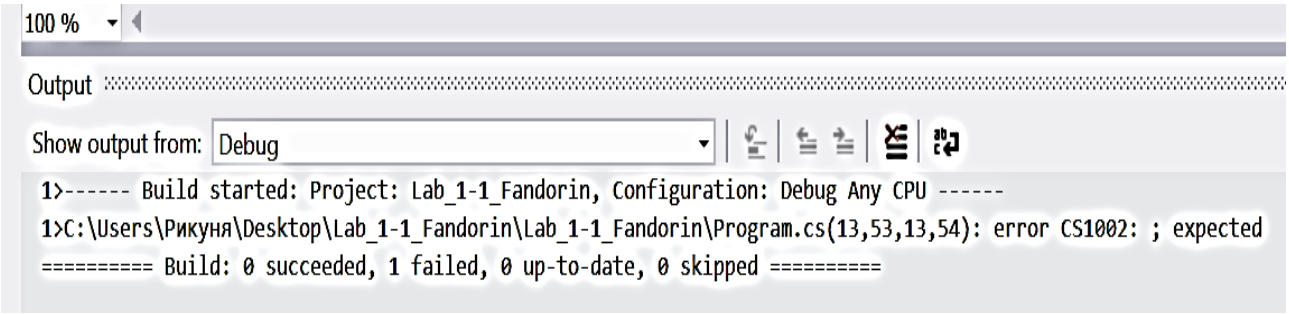


Рис. 17. **Вікно Output після закінчення процесу компіляції з помилкою**

А якщо викликати командою VIEW / Error List вікно зображення помилок (рис. 18), можна побачити кротке повідомлення про помилку (у даному випадку – відсутність крапки з комою).

У цьому разі не вдалося запустити додаток.

Коли буде розпочато розгляд синтаксису C#, стане зрозумілим, що наявність крапки з комою потрібна майже всюди — в кінці практично кожного рядка коду.

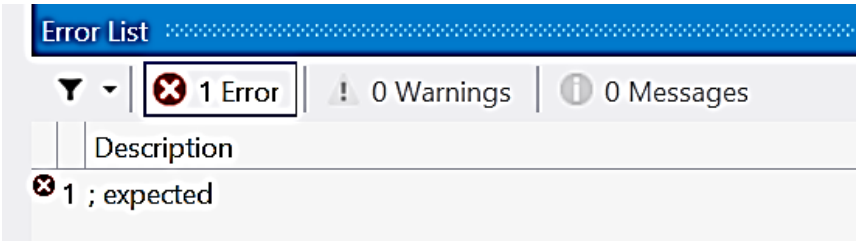


Рис. 18. **Вікно Error List з повідомленням про відсутність у першому рядку програми крапки з комою**

Якщо два рази клацнути мишею на помилку, зазначену в даному вікні, то курсор переміститься на ту позицію в рядку вихідного коду, де міститься помилка (якщо файл із вихідним текстом не був відкритий раніше, він відкриється в цей момент); це дозволяє швидко виправляти помилки.

У місцях, де виявлені помилки, також можна побачити хвилясті лінії червоного кольору, завдяки чому можна швидко переглядати вихідний код для виявлення помилок.

Слід звернути увагу на те, що місцезнаходження помилки вказується в вигляді номера рядка. В текстовому редакторі VS номери рядків за замовчуванням не виводяться.

Для переходу в режим виведення рядків буде потрібно встановити відповідний прапорець у діалоговому вікні Options (опції), в яке можна перейти через пункт меню Tools / Options ....(рис. 19).

За допомогою цього діалогового вікна можна отримати доступ до багатьох корисних опцій; деякі з них буде використано й надалі.

Проект 2. Створення графічного додатка Windows Forms

Часто виявляється зручніше продемонструвати роботу коду, запускаючи його як частину віконної (графічної) програми, а не через вікно консолі або через командний рядок. Досягти цього можна, скориставшись окремими "будівельними блоками" для створення користувальницького інтерфейсу.

Зараз відбудеться ознайомлення з основами цього процесу. Буде розглянуто, як створити і запустити віконний додаток, але що цей додаток виконує насправді, буде вивчатися у другому семестрі.

1. Для створення віконного програми виконати кроки (див. рис. 4), для отримання середовища розробки графічного додатка. Надати ім'я проекту Lab-1_2_Fandorin. Можливий вид вікна наведено на рисунку (див. рис. 5).

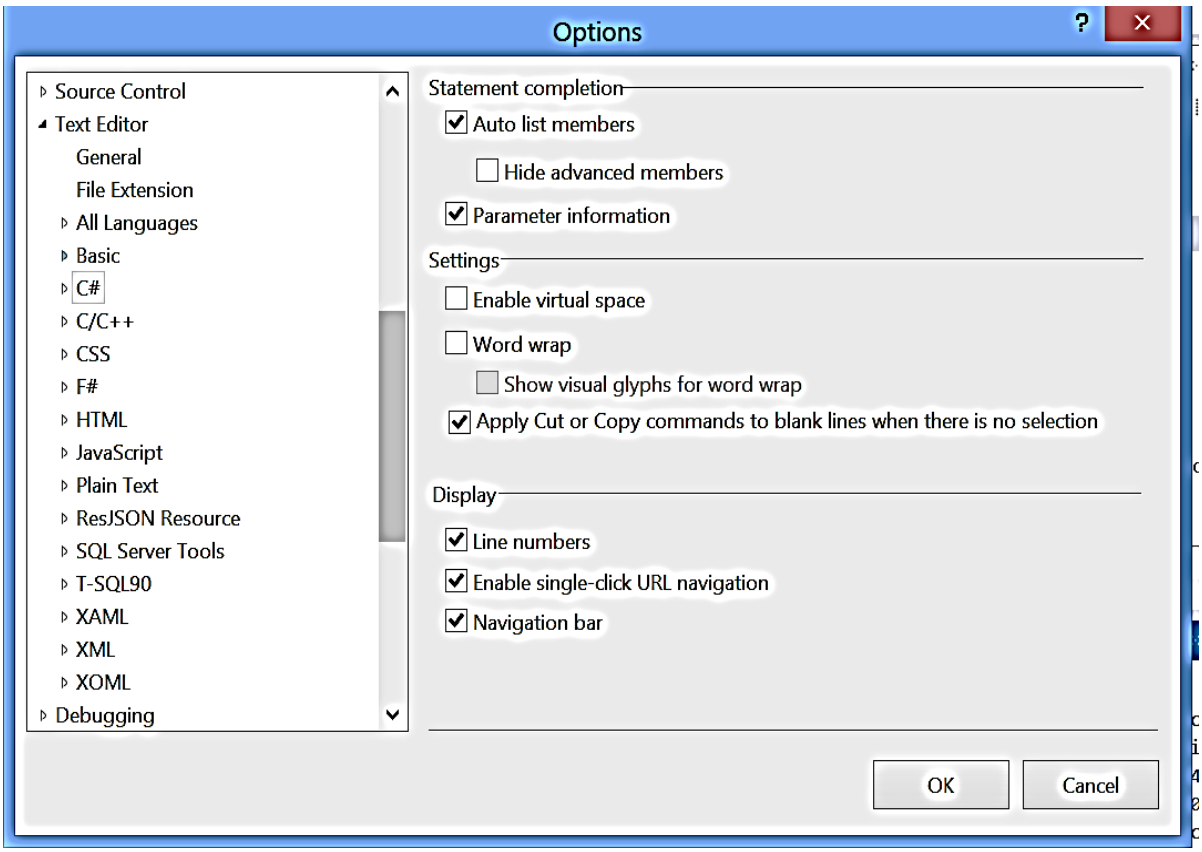


Рис. 19. **Вікно Options після вибору прапорця Line numbers**

2. Перемістити покажчик миші спочатку на панель Toolbox, яка розташована в лівій частині екрана, потім на входження Button (кнопка) вкладки All Windows Forms (рис. 20), і двічі клацніть лівою кнопкою миші: кнопка додається в основну форму додатка (Form1).

За допомогою курсору візуально встановити розміри кнопки і її місце на формі. Потім змінити ім'я форми і назву кнопки.

Ці зміни виконуються у вікні Properties. Виклик вікна можна здійснити з контекстного меню форми (рис. 21).

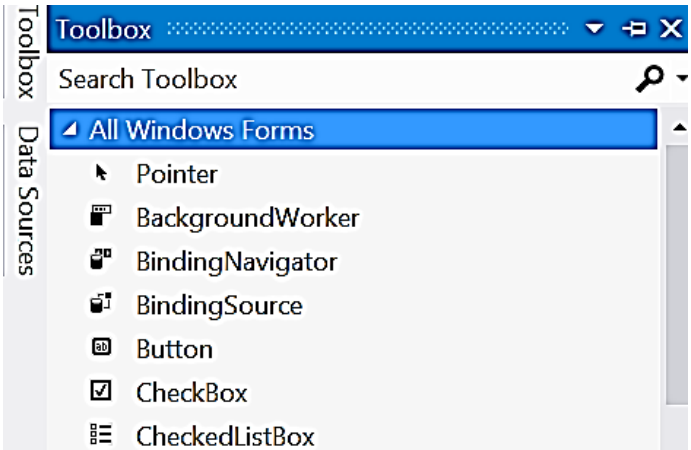


Рис. 20. **Вкладка All Windows Forms панелі Toolox**

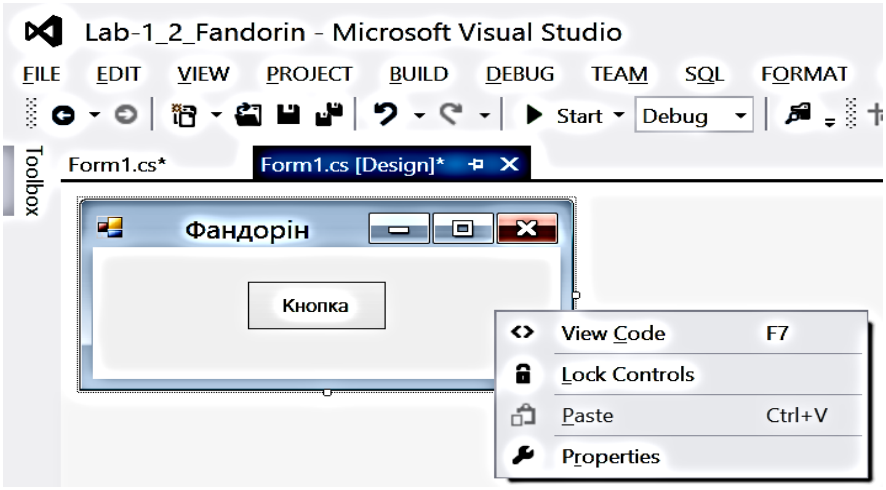


Рис. 21. **Контекстне меню форми**

Далі слід змінити властивість Text для форми і для кнопки відповідним чином (рис. 22). Результат наведено на рис. 23.

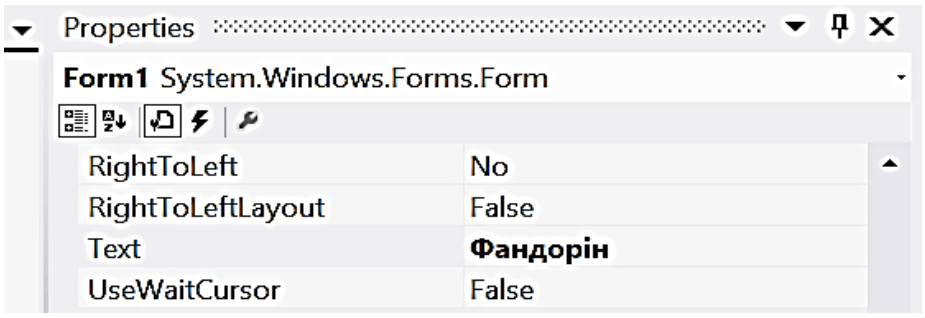


Рис. 22. **Зміна властивості Text для форми Form1**

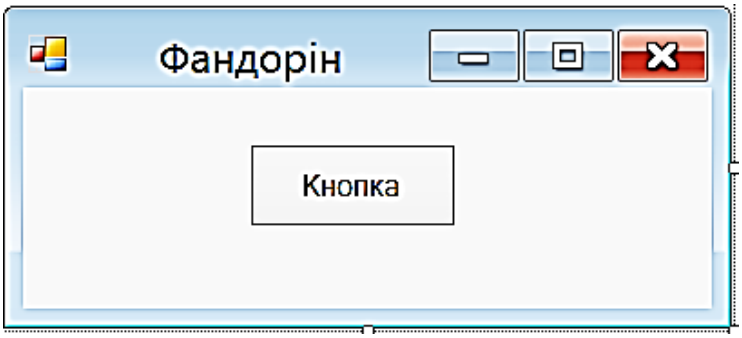


Рис. 23. **Графічний інтерфейс першого віконного додатка**

3. Два рази клацнути мишею по кнопці, доданої в форму.

4. На екрані має з'явитися код на C#, який знаходиться в цій формі. Зробити в ньому такі зміни (для стислості тут наводиться тільки частина того коду, який зберігається в файлі):

```
private void button1_Click(object sender, EventArgs e)
        {
          MessageBox.Show("Перший віконний додаток студента Фандоріна А.В.");
        }
```

5. Запустити додаток.

Натиснути кнопку, що з'явилася. Повинне відкритися діалогове вікно з повідомленням (рис. 24).

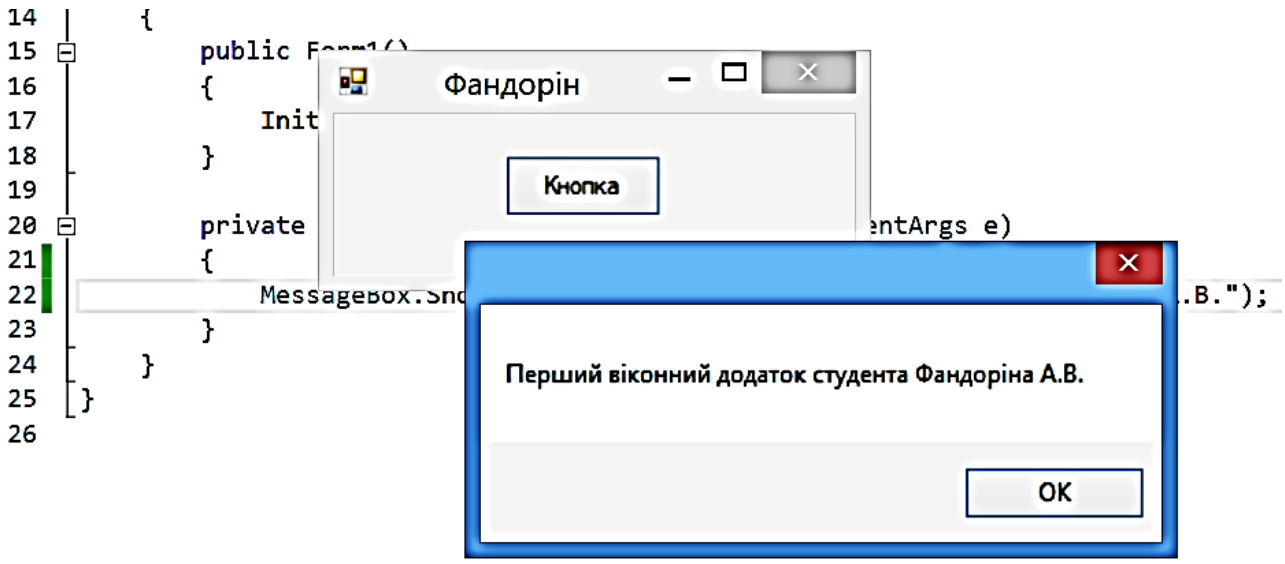


Рис. 24. **Результат виконання першого віконного додатка**

Висновки

У загальних рисах розглянуто середовище розробки Visual Studio.NET.

Показані приклади використання Visual Studio.NET для створення двох типів додатків. Більш простий з них — консольний додаток, його цілком достатньо для вирішення більшості завдань на початковому етапі вивчення мови C#, і він дозволяє зосередитися на основах програмування на C#.

Віконні додатки дещо складніші, проте візуально вони виявляються більш вражаючими і наочними для користувачів, знайомих із віконним середовищем.

Тепер, після знайомлення з основними етапами створення простих додатків, можна перейти до детального вивчення C#.

Наступні лабораторні заняття присвячені основам синтаксису C# і структурі програм на C#. Цей матеріал необхідно освоїти, перш ніж переходити до складніших об'єктно-орієнтованих додатків.

Зміст звіту

1. Титульний лист.

2. Цілі лабораторного заняття і вказівка, які навички та вміння передбачається отримати в результаті його виконання.

3. Тексти налагоджених демонстраційних програм лабораторного заняття з необхідними коментарями і результатом виконання.

4. Висновки.

Контрольні запитання

1. Що означає .NET? У чому особливість .NET-платформи?
2. Перерахуйте основні етапи розробки програми, що виконується.
3. Навіщо необхідний етап компіляції?
4. У чому особливість компіляції в .NET вихідного коду C#?
5. У чому суть процедурно-орієнтованого стилю програмування? Яка структура процедурно-орієнтованої програми?
6. Укажіть недоліки процедурного стилю програмування та шляхи їх подолання.
7. Дайте визначення об'єкта і наведіть приклад з описом його властивостей і поводження.
8. Навіщо необхідне повторне використання програмного забезпечення? Наведіть приклади повторного використання.
9. Призначення бібліотеки класів .NET Framework.
10. Опишіть можливі типи C#-додатків і області їхнього застосування.

Лабораторна робота № 2
Програмування лінійних обчислювальних процесів

Мета роботи – набуття практичних навичок із підготовки, налагодження і виконання лінійних програм.

Дана лабораторна робота сприяє напрацюванню таких **компетентностей** відповідно до Національної рамки кваліфікацій:

знання:

класифікації базових типів даних та їх основні характеристики;

лексичних основ мови C# - поняття: змінна, вираз, операнд, константа, оператор (інструкція);

методів Read (), ReadLine (), Write (), WriteLine ();

пріоритетів операцій;

правил перетворення типів;

основних бібліотечних математичних функції мови C#;

уміння:

складати лінійні програми з використанням стандартних бібліотечних функцій;

виконувати налагодження та покрокове тестування лінійних програм у середовищі налаштовування системи Visual C# .NET;

комунікації:

рекомендації команді учасників проекту щодо доцільності застосування поточного сценарію у вигляді лінійної алгоритмічної структури та найбільш відповідних базових типів даних;

автономність і відповідальність:

прийняття рішення щодо вибору реалізації поточного сценарію у вигляді лінійної послідовності операторів;

самостійне формулювання рекомендацій щодо обґрунтування відповідних бібліотечних математичних функції мови C#.

Основні положення

Лексичні елементи мови C#.

Будь яка алгоритмічна мова містить три складові частини: алфавіт (кінцева множина відмінних між собою символів, що використовуються у даній мові); синтаксис (сукупність правил, що визначають припустимі (правильні) конструкції даної мови. Синтаксис мови C# визначений у спеціальній граматиці); семантика (сукупність правил, що визначають значеннєвий зміст окремих конструкцій. Семантика забезпечує однозначність тлумачення всіх понять мови).

Алфавіт – це символи, що використовуються в мові C# під час написанні програм. Кожний файл – це текст. Для запису програм використовуються знаки у відповідному кодуванні. Українські букви можна використовувати в коментарях і літералах.

Коментарі. Будь-який текст, починаючи із двох знаків ділення \\ і до кінця рядка є коментарем, ніяк не аналізується комп'ютером і слугує лише для пояснень. Крім того, будь-який текст, розміщений між символами /* і */ також є коментарем. Три знаки \\\ також є ознакою коментаря, що може бути використаний під час компіляції програми для виділення фрагментів документації до програми у форматі XML.

Ідентифікатори. Послідовність символів із латинських букв, символів підкреслення і арабських цифр, що починається з букви та слугує для іменування різних елементів програми.

Програма – це запис алгоритму однією із мов програмування. Програма містить розділ команд і розділ опису даних.

Дані – це формалізоване подання всіх тих об'єктів (предметів, фактів, ідей), з якими може оперувати ПК. Включають у себе змінні та константи. Перш ніж задавати в програмі дії з даними, змінні та константи повинні бути визначені.

Змінна – символічне позначення величини в програмі. З погляду архітектури ПК, змінна – це символічне позначення комірки ОП, в якій зберігаються дані. Безпосередньо записати величину в програмі можна за допомогою літерної константи (як константа використовуються символи відповідного коду).

Вираження – це послідовність операндів, знаків операцій, круглих дужок, що задає обчислювальний процес одержання результату певного типу.

Операнд – це елемент-учасник операції. Операндами можуть бути: *константи* (це лексема, що становить зображення фіксованого числового, строкового або символьного (літерного) значення); *змінні*; *виклики функцій* – вказівка ім'я викликуваної функції, за яким у круглих дужках вказується список аргументів (можливо, порожній). Під час виконання програми результат, що повертається викликаною функцією, заміняє виклик функції; вираження.

Приклади перерахованих лексичних елементів C# буде розглянуто, аналізуючи базову структуру C#-програми (рис. 25) і результат її виконання (рис. 26).

Program.cs*

Program | Main(string[] args)

```
1   // Проста програма на C#
2   class Program
3       {
4           // Виклику методу Main()
5           static void Main(string[] args)
6           {
7               string answer;
8               System.Console.WriteLine("Як вас звуть?");
9               answer = System.Console.ReadLine();
10              System.Console.WriteLine("Добрий день, шановний " + answer);
11          }
12      }
```

Рис. 25. **Базова структура C# програми**

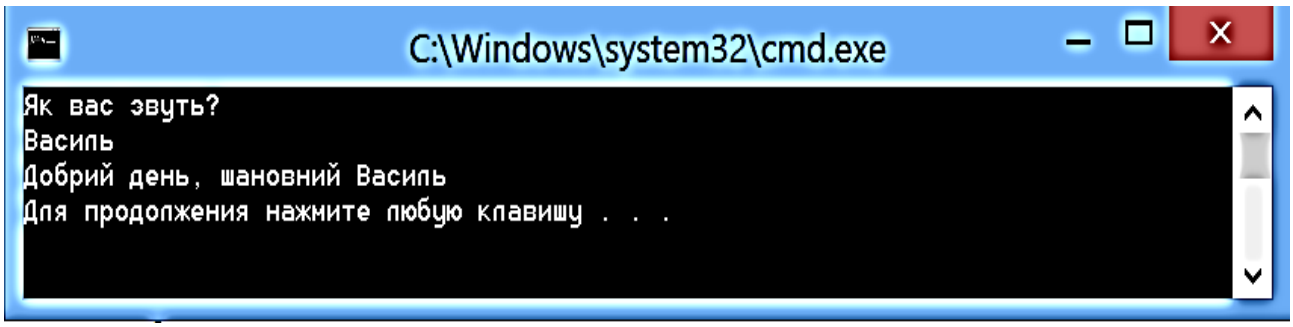


Рис. 26. **Результат виконання базової структури програми**

Слід розглянути кожну частину програми більш докладно.

Коментарі.

Рядок 1 містить коментар, вміст якого ігнорується компілятором. Він використовується для опису дій, які виконує програма. В даному випадку він просто повідомляє про те, що програма написана на C#.

01: // Проста програма на C#

Подвійний символ косої риски (//) змушує компілятор ігнорувати текст до кінця рядка. Рядок 1 містить тільки коментар, однак останній можна розмістити й у рядку з кодом. Рядки 1 та 2 можна об'єднати в такий спосіб:

class Program // Проста програма на C#

Інший варіант коду некоректний:

// Проста програма на C# class Program

тому що весь рядок, включаючи й class Program, розглядається компілятором як коментар.

Визначення класу.

Для пояснення рядка 2 необхідно звернутися до концепції ключового, або зарезервованого, слова. Ключове слово має спеціальне значення в мові C# і розпізнається компілятором.

У рядку 2 для визначення класу використовується ключове слово class.

02: class Program

Program – це ім'я класу, що розташовується безпосередньо за class.

У лістингу наведено кілька ключових слів: class, static, void, string та Main. Ключові слова мають для компілятора спеціальне значення. Їх не можна використовувати для інших цілей в C# (на що вказує термін "зарезервовані"). Слід зазначити, що ключове слово може бути частиною ім'я, тому назва classVariable цілком коректна.

Ідентифікатори (імена).

Імена у вихідному коді часто називають ідентифікаторами. Багато елементів – класи, об'єкти, методи, змінні екземпляра – повинні завжди мати ідентифікатори. На відміну від ключових слів C# вибір усіх ідентифікаторів залишається за програмістом. Тут існує декілька правил.

Ідентифікатор може складатися тільки з букв, цифр (0 – 9) і символу підкреслення (_). Ідентифікатор не може починатися з цифри та збігатися з одним із ключових слів. У C# ураховується регістр, тому прописні та малі літери вважаються різними символами.

Фігурні дужки та блоки вихідного коду.

Рядок 3 містить фігурну дужку ({), що вказує на початок блоку.

Блок – це фрагмент вихідного коду C#, поміщений у фігурні дужки. Блок є логічною одиницею коду. Фігурні дужки завжди застосовуються в парах. Коли в коді зустрічається {, це значить, що десь далі обов'язково знаходиться }, що їй відповідає. Дужка }, що відповідає рядку 3, 03: { знаходиться в рядку 12. Ще одна пара фігурних дужок перебуває в рядках 6 та 11.

Оскільки дужка { в рядку 3 розташована відразу після визначення в рядку 2, компілятор знає, що визначення всього класу Program міститься між дужкою { в рядку 3 і дужкою } в рядку 12.

У блоці визначення класу (рис. 27) тепер можна розмістити методи та змінні екземпляра за умови, що всі оголошення перебувають усередині блоку.

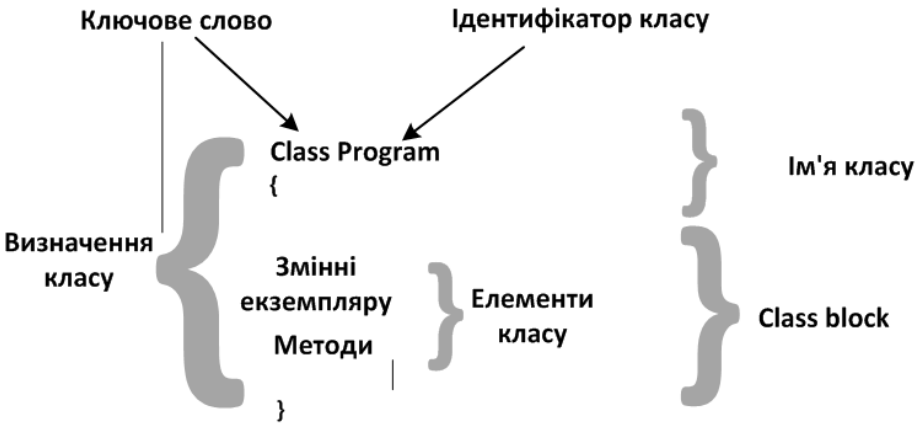


Рис. 27. **Визначення класу**

У рядку 4

```
04:         // Виклик методу Main()
```

знаходиться вже знайомий символ коментаря //.

Метод Main() та його визначення.

У рядку 5 починається визначення методу під назвою Main. В C# немає ключового слова на зразок "method", що вказує на те, що конструкція є методом. Компілятор розпізнає метод за круглими дужками, наступними за його ім'ям, зокрема, ( ) після Main.

05: static void Main(string[] args)

Метод Main має в C# спеціальне значення. З цього методу починає виконання кожний додаток – він викликається середовищем виконання при запуску програми.

Точне значення всіх елементів рядка 5 поки що не буде обговорюватися, оскільки вимагає більш детального розуміння певних об'єктно-орієнтованих принципів C#.

Отже, клас складається з інтерфейсу, реалізованого за допомогою відкритих методів і схованої частини, що складається з закритих методів і змінних екземпляра.

Основні елементи визначення методу ілюструються на рис. 28.

Кожна програма на C# повинна містити метод Main().Під час запуску середовище виконання .NET, у першу чергу, шукає цей метод. Якщо він знайдений, з нього починається виконання, якщо ні – виводиться повідомлення про помилку.

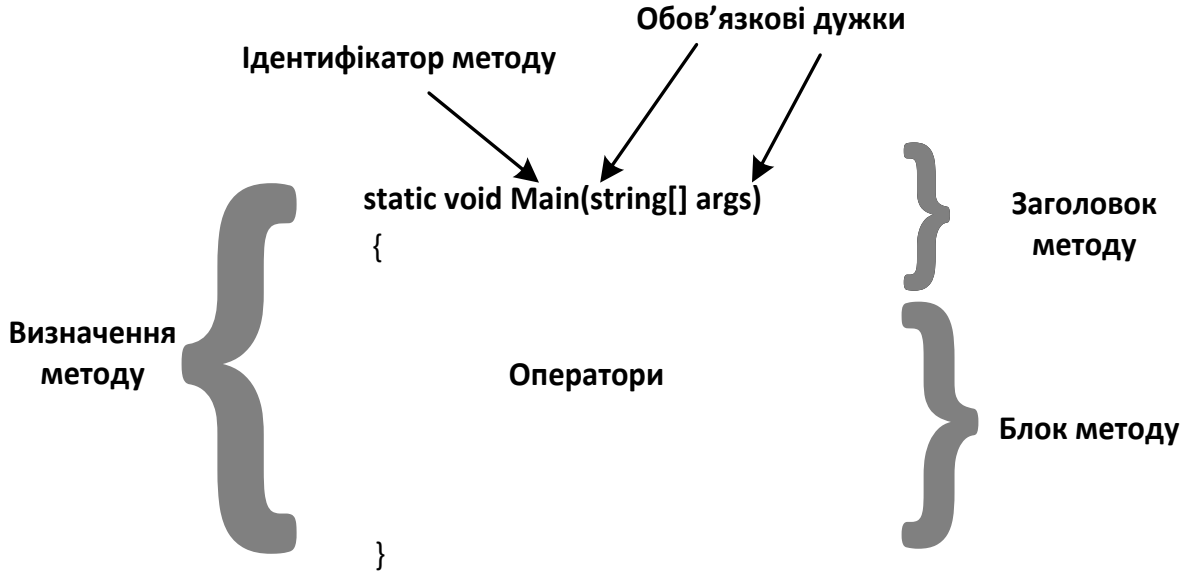


Рис. 28. **Визначення методу**

Main() на лістингу рис. 25 розташований усередині класу Program, а середовище виконання -.NET – зовні. Під час спроби запуску Main() середовище буде розглядатися як ще один об'єкт, що запитує доступ до методу класу. Тому його необхідно відкрити, зробивши частиною інтерфейсу класу.

Для початкового розгляду ключового слова static слід звернутися знову до обговорення розходжень між класом та об'єктом.

Клас є специфікацією того, як створити об'єкт, так само, як креслення є просто планом реального будинку. Клас звичайно не може робити будь-яких дій. Ключове слово static дозволяє відійти від цієї схеми і скористатися методами класу, не створюючи конкретного екземпляра об'єкта.

Коли static включено в заголовок методу, це повідомляє клас про те, що для використання методу не потрібно створювати екземплярів за межами класу. Таким чином, метод Main() може використовуватися до створення певного об'єкта класу Program. У даному випадку це обов'язково, тому що Main() викликається середовищем виконання .NET до того, як створюються які-небудь об'єкти.

Щоб зрозуміти значення ключового слова void в рядку 5, необхідно звернутися безпосередньо до того, як працюють методи. У цьому розділі буде наведено лише коротке пояснення: void означає, що Main() не повертає значення в точку виклику.

У рядку 6 дужка { вказує на початок блоку Main(), де міститься тіло методу. Блок закінчується дужкою } в рядку 11.

Для поліпшення читаності коду варто вибирати значущі імена змінних і уникати абревіатур.

Змінні.

Змінна є іменованою позицією в пам'яті, що становить збережений блок даних. Ключове слово string вказує, що answer належить типу string

```
07:    string answer;
```

Ідентифікатор (answer) програміст вибирає за своїм розсудом, а string є зарезервованим словом.

Розміщення answer після string в рядку 7 означає, що оголошена змінна answer типу string.

Кожна змінна, що використовується в програмі на C#, повинна бути оголошена.

Змінна answer застосовується в рядках 9 та 10.

Змінна типу string може містити будь який текст. У C# рядки тексту позначаються " " (подвійні лапки). Складові частини визначення змінної показані на рис. 29.

З рис. 29 видно, що змінна складається з трьох елементів:

ідентифікатора (в даному випадку, answer);

типу, тобто виду інформації, що вона може зберігати (в даному випадку – string, тобто послідовність символів);

значення, тобто збереженої інформації. Поточне значення на рисунку дорівнює "Julian is a boy".

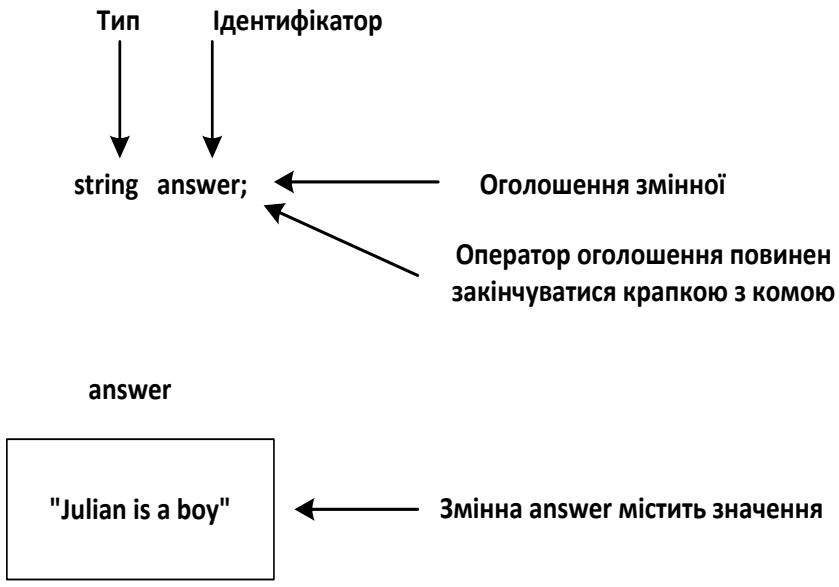


Рис. 29. **Тип, ідентифікатор та значення змінної**

Будь-яку задачу, що виконується програмою на C#, можна розбити на послідовність інструкцій. Найпростіша інструкція називається оператором. Усі оператори закінчуються символом крапки з комою.

Рядок 7 містить оператор оголошення змінної, тому він, як і інші, закінчується крапкою з комою.

Запуск методів .NET-платформи.

Оператор в рядку 8

```
08:  System.Console.WriteLine("Як вас звуть? ");
```

змушує програму вивести на екран таке:

Як вас звуть?

На даний момент досить розглядати виклик System. Console.WriteLine( ) як просто спосіб виведення, що має сенс: "вивести все, що міститься в дужках після WriteLine на екран та перейти на один рядок нижче".

System.Console – це клас .NET Framework. .NET Framework є бібліотекою, яка містить множину корисних класів, створених розроблювачами з Microsoft. Таким чином, для виведення тексту на екран повторно використовується клас System.Console. Він містить метод WriteLine( ), що й викликається командою System.Console.WriteLine( ).

Коли метод виконує певну задачу в програмі, це називають викликом. Елемент усередині круглих дужок (текст " Як вас звуть? " в прикладі) називається аргументом. Аргумент містить інформацію, необхідну методу, що вікликається, для виконання завдання. Аргумент передається методу WriteLine під час виклику. Після цього метод звертається до даних вже за допомогою своїх внутрішніх операторів. Рядок 8, як і 7, містить оператор і тому закінчується крапкою з комою.

Слід розглянути, як саме в рядку 8 використовується метод класу System.Console. Як уже підкреслювалося, класи є "схемами", а об'єкти – "виконавцями".

Метод класу можна використовувати в тому випадку, коли в його оголошенні присутнє ключове слово static, яке згадувалося раніше. Таким чином, метод WriteLine доступний без створення конкретного екземпляра об'єкта System.Console.

Загальний механізм виклику методу.

Інструкції методу містяться усередині його визначення у формі операторів. Викликати метод – означає виконати його інструкції. Виконання відбуваються послідовно в тому порядку, в якому вони написані у вихідному коді.

Метод можна визначити тільки всередині класу. Він є дією, яку здатен виконати об'єкт.

Виклик методу має такий синтаксис: ім'я об'єкта (або класу, якщо метод оголошений як static), точка, ім'я методу та завершальна пара круглих дужок ( ), в яких можуть міститися аргументи. Останні становлять дані, передані методу.

Виклик нестатичного методу:

ІмяОб'єкта.ІмяМетоду(Необов'язкові_аргументи)

Виклик статичного методу:

ІмяКласу.ІмяМетоду (Необов'язкові_аргументи)

Замінивши загальні елементи реальними іменами, легко отримати оператор з рядка 10 лістингу на рис. 25.

System.Console.WriteLine("Добрий день, шановний " + answer);

Знак "+" в даному контексті (коли він розташований праворуч від текстової змінної) позначає операцію контактенації (не плутати з арифметичною операцією складання). У ході виконання праворуч від рядка, яка виводиться на екран, приєднується зміст строкової змінної, тобто — ім'я користувача.

Після завершення методу потік керування вертається в точку, з якої відбувся виклик (рис. 30).

На рис. 30 можна виділити такі кроки:

1. Виконати оператори в рядках, що знаходяться вище рядка передують 8.

2. Запустити рядок 8.

3. Викликати оператор System.Console.WriteLine("Як вас звуть");

4. Виконати оператори всередині System.Console WriteLine (...).

5. Повернути керування операторові в рядку після опреатора 8.

6. Виконати інші оператори методу Main.

Присвоювання значення змінної.

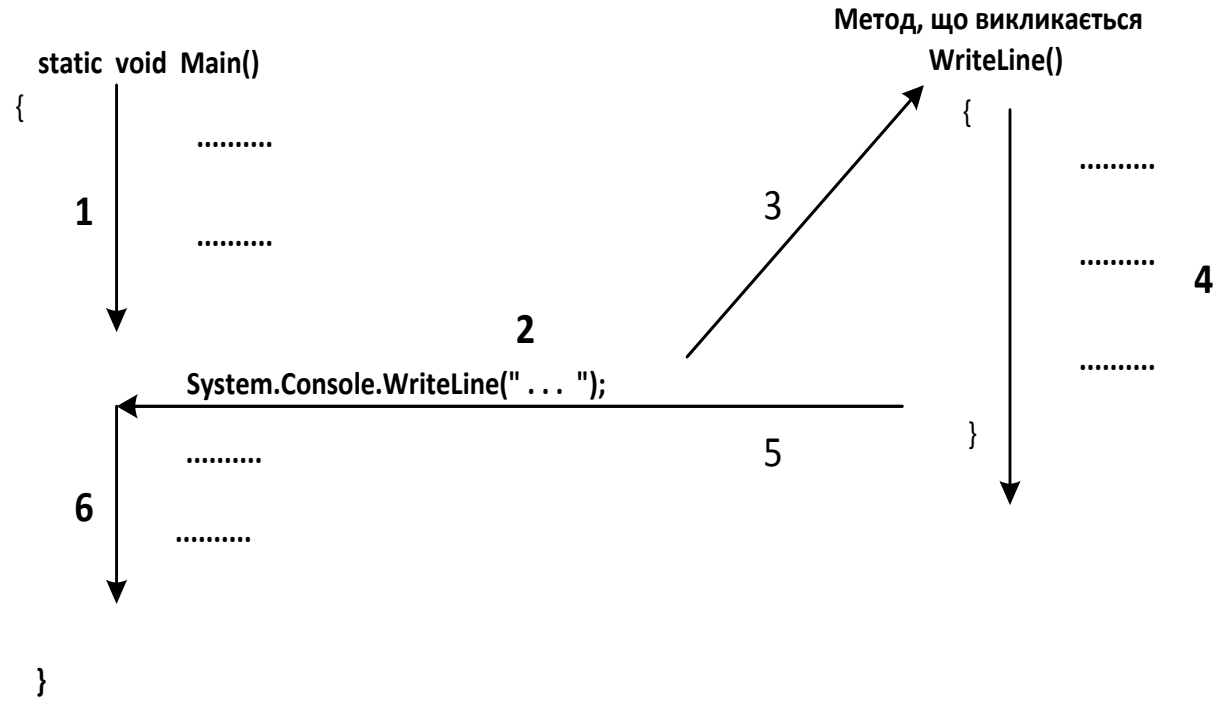


Рис. 30. **Рух потоку виконання програми під час виклику метода**

У рядку 9 знову повторно використовується клас System.Console. Цього разу застосовується інший з його статичних методів – ReadLine, що призупиняє виконання програми, очікуючи введення від користувача. Відповіддю може бути введений текст, який завершується натисканням клавіші Enter. Як виходить із назви, метод ReadLine читає введення:

9: answer = System.Console.ReadLine();

Під час натискання Enter текст, введений користувачем, зберігається в змінній answer.

Механізм задання нового значення змінній answer називається присвоюванням. Говорять, що введений текст присвоюється змінній answer.

Загальне вираження в рядку називають оператором присвоювання, а сам знак рівності (=) називають операцією присвоювання (в даному контексті). Якщо знак "рівність" використовується в інших контекстах, він має інші назви.

Завершення методу Main() і класу Program.

Дужка } в рядку 11 завершує блок методу Main(), початий в рядку 6.

```
11:    }
```

У рядку 12 дужка } завершує блок класу Program.

```
12: }
```

Формат вихідного коду C#.

Порожні рядки, символи пробілу, табуляції та повернення каретки називають єдиним терміном "порожній символ". Компілятор C# ігнорує їх. Тому всі ці символи можна використовувати еквівалентно.

Неподільні елементи в рядку вихідного коду називаються лексемами. Вони повинні відділятися один від одного порожніми символами, комами або крапками з комою. Самі лексеми розділяти порожніми або іншими символами не можна. Лексема (token) – це слово, що має певне значення. Цей термін часто використовується в логіці та лінгвістиці. Спроба розриву лексем приводить до некоректного коду.

Хоч C# надає певну свободу у форматуванні коду, гарний стиль може значно поліпшити його читабельність. Стиль, наведений у лістингу на рис. 25, прийнятий більшістю програмістів.

Поняття типу даних

Концепція типу даних.

Кожний конкретний тип даних визначається двома факторами: множиною значень, які можуть приймати об'єкти даного типу; набором операцій, що можна застосовувати до даного типу.

В описі даних повинна міститися (для компілятора) така інформація, що задається типом даних:

ім'я змінної або константи;

розмір пам'яті, необхідної для зберігання значень;

які дії можна виконувати зі змінною або константою;

вид та спосіб виділення пам'яті;

початкове значення змінної або значення константи.

C# є жорстко типізованою мовою. Під час його використання програміст повинен оголошувати тип кожного об'єкта, який він створює (наприклад, цілі числа, числа із плаваючою точкою, рядки, вікна, кнопки і т. д.).

Класифікація типів даних.

C# підрозділяє типи на два види: вбудовані типи (або прості типи), які визначені в мові, і типи, визначені користувачем (типи, які вибирає програміст).

C# також підрозділяє типи на дві інші категорії: типи-значення (або розмірні) та посилальні типи.

Основна відмінність між ними – це спосіб, яким їхні значення зберігаються в пам'яті.

Змінна типу-значення містить значення, збережене безпосередньо в ній (тобто у відповідних комірках пам'яті комп'ютера). Прикладом може слугувати тип int.

Змінна посилального типу містить в пам'яті посилання на об'єкт, а не сам об'єкт безпосередньо.

Посилання становить позицію (адресу) об'єкта в пам'яті. Для ілюстрації слід звернутися до вже вивченого типу string (який, як з'ясується надалі, є посилальним). Змінна типу string не містить рядок із текстом, а оголошується для зберігання посилання на рядок де знаходиться текст. Сам рядок розміщується за визначеною адресою у пам'яті.

Важливо зазначити, що фактична адреса під час використання посилань у вихідному коді програми ніколи не застосовується.

Усі класи є посилальними типами.

Адреса місця в комп'ютерній пам'яті, де зберігається об'єкт, називається також покажчиком на цей об'єкт.

У більшості випадків відмінності в роботі з типами-значеннями та посилальними типами незначні. Приклад цього – можливість застосування рядків у попередніх програмах без використання поняття посилання.

Особливості застосування простих типів.

Прості типи належать до групи вбудованих типів C#. Прикладами їхніх значень є окремі числа (тип int) і окремі символи.

Під час вибору змінної певного типу програміст фактично задає вид величини, що змінна може зберігати, і набір операцій, в яких вона може брати участь. Кожний простий тип характеризується такими властивостями:

Форма подання змінної. Приклади – цілі числа, числа з плаваючою точкою та одиночні символи.

Діапазон значень змінної. Наприклад, діапазон типу int: від
-2 147 483 648 до 2 147 483 647.

Обсяг використовуваної внутрішньої пам'яті. Для представлення однієї змінної залежно від її типу використовується від 8 до 64 бітів. Наприклад, змінна типу int займає 32 біти пам'яті.

Типи операцій, які можна виконувати зі змінної. Попередні приклади показують, що тип int підходить для додавання, а строкові значення – для конкатенації.

У C# визначено 13 простих типів, які перераховані в табл. 1.

Таблиця 1

Прості типи в мові C#

Ключове слово мови C#	Tun .NET CTS	Вид значення	Використовувана пам'ять	Діапазон і точність
sbyte	System.SByte	Ціле число	8 бітів	Від -128 до 127
byte	System.Byte	Ціле число	8 бітів	Від 0 до 255
short	System.Int16	Ціле число	16 бітів	Від -32 768 до 32 767
ushort	System.Ulnt16	Ціле число	16 бітів	Від 0 до 65 535
int	System.Int32	Ціле число	32 біти	Від -2 147 483 648 до 2 147 483 647
uint	System.Ulnt32	Ціле число	32 біти	Від 0 до 4 294 967 295
long	System.Int64	Ціле число	64 біти	Від -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807
ulong	System.Ulnt64	Ціле число	64 біти	Від 0 до 18 446 744 073 709 551 615
char	System.Char	Ціле число (один символ)	16 бітів	Всі символи Unicode
float	System.Single	Число з плаваючою точкою	32 біти	Від (+/-)1.5 * 1045 до (+/-)3.4 * 1038 Приблизно 7 значущих цифр
double	System.Double	Число з плаваючою точкою	64 біти	Від (+/-) 5.0 * 10-324 до (+/-)3.4 * 1030 15-16 значущих цифр
decimal	System.Decimal	Десяткове число (високої точності)	128 бітів	Від (+/-) 1.0 * 10-28 до (+/-)7.9 * 1028
bool	System.Boolean	true або false	1 біт	Немає

Хоча тип bool (останній рядок табл. 1) розглядається тут як простий тип, він пов'язаний з керуванням потоком виконання програм. Ключове слово відноситься до символу, що використовується у вихідному коді C# під час оголошенні змінної.

Простір імен System .NET Framework містить всі прості типи. Кожне ключове слово, показане в першому стовпці, – це псевдонім типу, визначеного в CTS. Наприклад, ключове слово int позначає System.Int32 в CTS. Таким чином, у вихідному коді можна використовувати як короткий псевдонім типу, так і його довге повне ім'я.

Перетворення вбудованих типів.

Об'єкти одного типу можуть бути перетворені в об'єкти іншого типу неявно або явно.

Неявні перетворення відбуваються автоматично, компілятор робить це замість програміста (рис. 31).

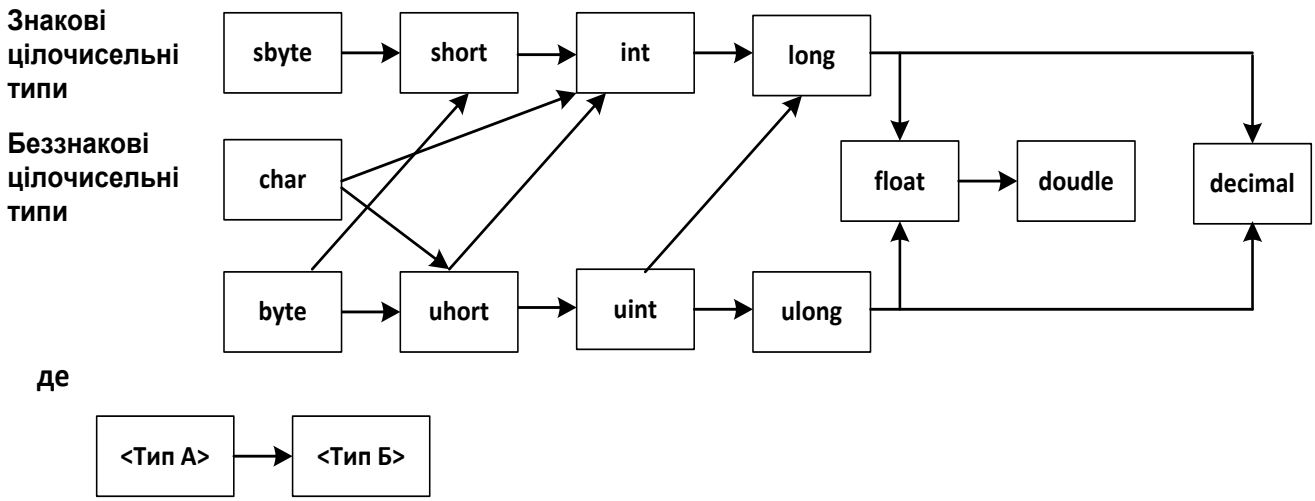


Рис. 31. **Шляхи неявного перетворення числових типів**

Неявні перетворення гарантують також, що дані не будуть загублені. Наприклад, можна неявно приводити від short (2 байти) до int (4 байти). Незалежно від того, яке значення находиться в short, воно не втратиться під час перетворення до int:

```
short x = 1;
int y = x;          //      неявне перетворення
```

Якщо робиться зворотне перетворення, то, звичайно ж, можна втратити інформацію. Якщо значення в int більше, ніж 32.767, воно буде усіче-

не під час перетворення. Компілятор не стане виконувати неявне перетворення від int до short:

```
short   x;
int y  =  5;
x  =  y;    //        не  компілюється
```

Явні перетворення здійснюються, коли програміст "приводить" значення до іншого типу.

Програміст повинен виконати явне перетворення, використовуючи оператор приведення:

```
short   x;
int   y  =  5;
x  =  (short)   y; //        OK
```

Константні величини.

Часто у вихідній програмі використовуються фіксовані числа, наприклад: 3.141592 або інші величини, які не змінюються протягом виконання програми. Мова C# дозволяє оголошувати імена для літералів або інших виражень і використовувати їх замість запису фактичного значення.

Наприклад, замість

```
distance = secondsTraveled * 186000;
```

можна дати значенню 186000 ім'я SpeedOfLight і привести вираження до вигдялу:

```
distance = secondsTraveled * SpeedOfLight;
```

Константа SpeedOfLight схожа на змінну тим, що вона має ім'я і певне значення. Фактично її можна було б оголосити як змінну:

```
int SpeedOfLight = 186000;
```

Це дозволяє застосовувати SpeedOfLight під час обчислення відстані. Але не слід забувати, що значення SpeedOfLight може бути випадково змінене в програмі. Тому, щоб залишити величину SpeedOfLight незмінною, у процесі її визначення потрібно вказати ключове слово const:

```
const   int   SpeedOfLight = 186000;
```

Ім'я, що становить постійну величину, в C# називається константою.

Константи поділяються на такі групи:

- Числові
  - Цілі
  - Речовинні
- Перелічувані
- Символьні (літерні)

Клавіатурні

Кодові (керуючі або розділові символи)

Кодові числові

Строкові (рядки або літерні рядки)

Іменовані (символічні)

Для виконання подальших лабораторних робіт становить інтерес особливості застосування символьних (літерних) констант.

Розрізняють такі символьні константи.

Клавіатурні: '1', 't', 'y' – клавіатурний символ задається в апострофах;

Наприклад, char t = 'd'; //

Кодові (табл. 2) – для завдання деяких керуючих і розділових символів, наприклад, '\n', '\t';

В англійській термінології для керуючих послідовностей застосовується термін "escape" – "бігти", "уникати", що підкреслює їхню здатність уникати стандартної конкатенації.

Таблиця 2

Керуючі послідовності

Керуюча послідовність	Призначення	Подання Unicode
\'	Одинарні лапки	\u0027
\”	Подвійні лапки	\u0022
\\	Зворотна коса риска	\u005C
\0	Символ Null	\u0000
\a	Дзвінок	\u0007
\b	Символ забою	\u0008
\f	Подача сторінки	\u000C
\n	Переведення рядка	\u000A
\r	Повернення каретки	\u000D
\t	Горизонтальна табуляція	\u0009
\v	Вертикальна табуляція	\u0008

Кодові числові – для задання будь-яких кодів символів, наприклад,

char t = '\x1A' ; // код символу "←"

Програмування лінійних обчислювальних процесів

Основні операції мови C#.

Будь-яка програма може бути складена за допомогою чотирьох основних структур:

послідовність (або структура проходження) – це група команд, виконуваних одна за другою. Програми, що складаються тільки зі структури проходження, називаються *лінійними програмами*;

рішення (або структура вибору) – це конструкція, що дозволяє даним впливати на хід виконання програми (організовувати розгалуження в програмах);

повторення (цикл або структура повторення) – дозволяє багаторазово виконувати команду або групу команд;

метод (процедура, функція) – дає можливість замінити групу команд однією командою.

Структура проходження вбудована в C#. Поки не зазначено інше, комп'ютер виконує інструкції C# одну за другою в тій послідовності, в якій вони записані.

У C# доступна велика кількість операцій. За винятком тих, які призначені для спеціальних цілей, операції можна розділити на чотири основних категорії: арифметичні, відношення, логічні та побітові.

Арифметичні операції застосовуються до значень простих числових типів і становлять основу для обчислень у C#.

Операції відношення дозволяють порівнювати значення, наприклад sum < 100.

Операції відношення дають вираз логічного типу, що, за визначенням, завжди має значення або true, або false.

Логічні операції дозволяють об'єднати два або більше логічних виразів (будь-які значення типу bool). Вони тісно пов'язані з операціями відношення і дають можливість за допомогою оператора if та операторів циклів керувати потоком виконання програми.

Побітові операції дозволяють працювати з окремими бітами відповідних операндів.

Арифметичні операції (додавання (+), віднімання (-), множення (*) і ділення (/)) в сполученні з числами (так званими операндами) формують арифметичні вирази.

Арифметичні вирази на C# досить легко інтерпретувати, тому що вони (більш-менш) відповідають усім відомим арифметичним правилам. Усім чотирьом базовим операціям, згаданим у попередньому розділі, потрібен один операнд ліворуч і один праворуч. Арифметична операція, що поєднує у виразі два операнда, наприклад:

distance1 + distance2,

називається бінарною. Операнд у такому контексті може набувати різних форм. Це може бути просто число (наприклад, 345.23), змінна, константа, що становить числове значення, або ж числовий вираз, наприклад:

distance1 * 100 + distance2 * 300.

Операндом може бути й виклик методу. В цьому випадку, природно, метод повинен повертати значення.

Пріоритети операцій.

Будь-який вираз, де використовується більше двох операндів, завжди можна розбити на підвираження, кожне з яких складається тільки з двох операндів і однієї бінарної операції. Після обчислення результатів підвиражень вони використовуються на наступному рівні.

Слід розглянути вираз:

$$4 \times 5 + 40 \times 10 - 20 \times 40/10 + 70$$

Легко побачити, що воно складається з декількох підвиразів. На рис. 32 показано, як, користуючись визначенням числового виразу, можна виконати обчислення.

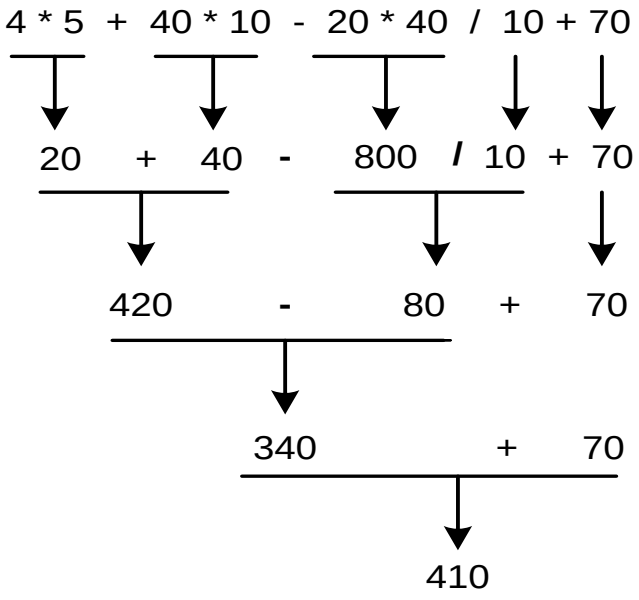


Рис. 32. **Черговість виконання операцій під час обчислення виразу**

Відповідно до відомих правил, множення відбувається до додавання, як показано на рис. 33.

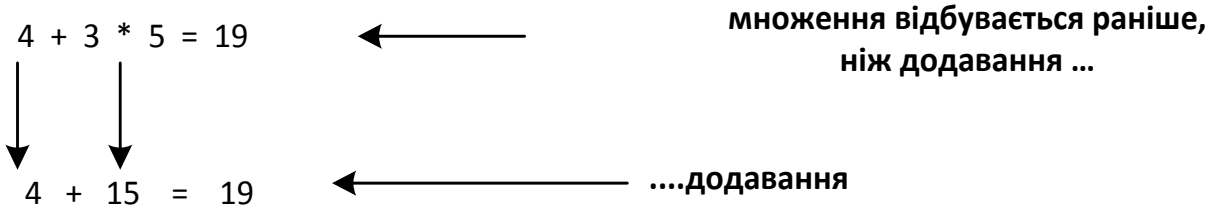


Рис. 33. **Множення відбувається до додавання**

Отже, коли операнд (наприклад, число 3 на рис. 3.3) може бути оброблений більш ніж однією операцією (+ або * в даному випадку), мова C# діє за правилами старшинства (пріоритетів) операцій.

Для простоти в числових виразах на рисунках використовувалися числа в явній формі. Очевидно, що замість них можна було б застосувати комбінацію будь-яких визначень операндів.

Крім чотирьох бінарних операцій, C# містить і інші арифметичні операції. Порядок виконання кожної з них стосовно інших чітко визначений у таблиці пріоритетів.

Огляд операцій та їхніх пріоритетів наведено у табл. 3.

Форматування числових значень

До тепер числа виводилися з використанням простого вбудованого формату, наприклад,

```
Console.WriteLine("Distance  traveled:" + 10000000.432);
```

При цьому на консоль відображається таке:

```
Distance traveled: 10000000.432
```

Однак, змінюючи зовнішній вигляд числа за допомогою ком (у закордонній нотації комою прийнято відокремлювати розряди, кратні тисячам), наукової нотації та обмеженої кількості десяткових цифр, можна поліпшити його читабельність і зробити більше компактним під час виведення на екран. У табл. 4 наведені відповідні приклади.

Стандартне форматування.

Кожний числовий тип в .NET Framework поданий структурою struct, що дозволяє йому містити корисну вбудовану функціональність. Одним із її прикладів є метод ToString. Він дозволяє перетворити будь-який із простих типів у рядок і, крім того, вказати необхідний формат.

Таблиця 3

Пріоритети операцій

Категорії	Операції	Асоціативність	Значення
1	2	3	4
Угрупування	(<Вираз>)	Зліва направо	Круглі дужки для угрупування
Первинні	<Ім'я_об'єкта>.<Ім'я_елемента>	Доступ до елемента	
	<Ім'я_методу>(<Аргументи>)	Зліва направо	Виклик методу
	<Ім'я_масиву>[Індекс]		Доступ до масиву
	<Змінна>++	Справа наліво	Постфіксна операція інкремента
	<Змінна> --	Справа наліво	Постфіксна операція декремента
	new <Ім'я_класу> (<Аргументи>)		Створення екземпляра класу
	typeof(<Тип>)		Визначення типу
	sizeof(<Тип>)		Визначення розміру структури
Унарні	+<Вираз>	Справа наліво	Унарний плюс
	-< Вираз >	Справа наліво	Унарний мінус
	!<Логічний_ Вираз >	Справа наліво	Логічне заперечення
	++<Змінна>	Справа наліво	Префіксна операція інкремента
	--<Змінна>	Справа наліво	Префіксна операція декремента
	(<Тип>) < Вираз >	Справа наліво	Приведення типу
Мультиплікативні	< Вираз1 > * < Вираз2>	Зліва направо	Множення
	< Вираз1> / < Вираз2>	Зліва направо	Ділення
	< Вираз1> % < Вираз2>	Зліва направо	Ділення за модулем

Закінчення табл. 3

1	2	3	4
Адитивні	< Вираз1> + < Вираз2>	Зліва направо	Додавання
	<< Вираз1> - < Вираз2>	Зліва направо	Віднімання
Відношення	< Вираз1> < < Вираз2>	Зліва направо	Менше
	< Вираз1> <= < Вираз2>	Зліва направо	Менше або дорівнює
	< Вираз1> > < Вираз2>	Зліва направо	Більше
	< Вираз1> >= < Вираз2>	Зліва направо	Більше або дорівнює
	<Об'єкт> is <Тип>	Зліва направо	Приналежність типу
Рівність	< Вираз1> == < Вираз2>	Зліва направо	Дорівнює
	< Вираз1> != < Вираз2>	Зліва направо	Не дорівнює
Логічні побітові	< Логічний_вираз1> & < Логічний_вираз2>	Зліва направо	Побітове І
	< Логічний_вираз1> ^ < Логічний_вираз2>	Зліва направо	Побітове АБО, що виключає
	< Логічний_вираз1> | < Логічний_вираз2>	Зліва направо	Побітове АБО
Логічні	< Логічний_вираз1> && < Логічний_вираз2>	Зліва направо	Логічне І
	< Логічний_вираз1> || < Логічний_вираз2>	Зліва направо	Логічне АБО
	< Логічний_вираз> ? < Вираз1> : < Вираз2>	Зліва направо	Умовна операція
Присвоювання	<Змінна> = < Вираз>	Справа наліво	Просте присвоювання
	<Змінна> *= < Вираз>	Справа наліво	Множення і присвоювання
	<Змінна> /= < Вираз>	Справа наліво	Ділення і присвоювання
	<Змінна> % = < Вираз >	Справа наліво	Ділення за модулем і присвоювання
	<Змінна> += < Вираз >	Справа наліво	Додавання і присвоювання
	<Змінна> - = < Вираз >	Справа наліво	Вирахування і присвоювання

Таблиця 4

Приклади форматування чисел

Ім'я змінної	Число	Відформатоване число
distance	7 000 000 000 000 000	7.00E+015
Mass	3.8783902983789877362	3.8784
Length	20 000 000	20,000,000

На рис. 34 показано використання методу ToString для перетворення числа 20 000 000.45965981m типу decimal у рядок типу string з комами (для розділення тисяч) і лише двома десятковими розрядами.

Метод ToString має один аргумент типу string, що задає формат.

Аргумент складається з символу, званого символом формату (в даному випадку N), що задає формат, і необов'язкового числа специфікатора точності (в даному випадку 2), яке має різний сенс для різних символів формату.

Використовувані символи й відповідні їм формати наведені в табл. 5.

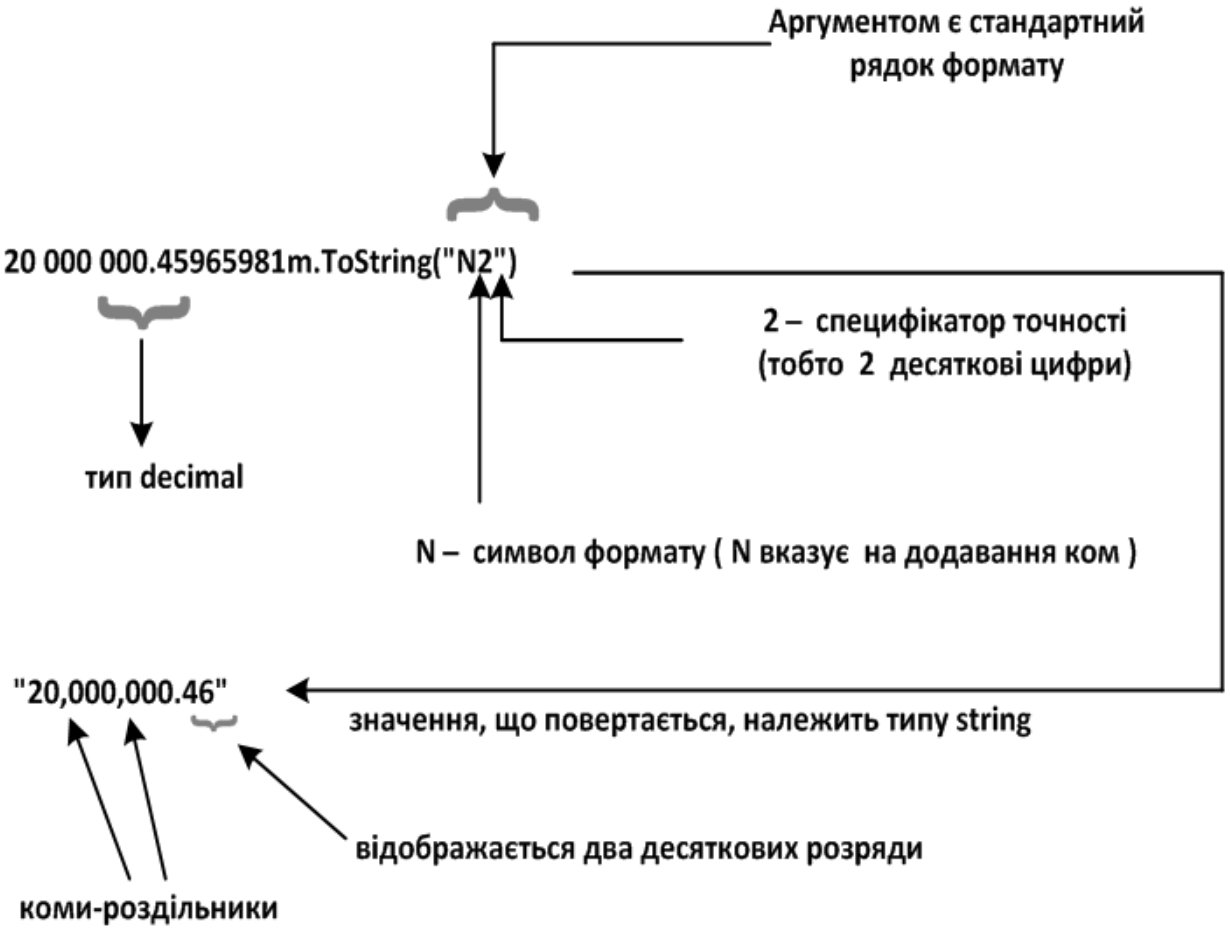


Рис. 34. **Форматування літерала типу decimal**

Таблиця 5

C# символи формату

Символи	Опис	Приклади
C, c	Валюта. Форматування, специфічне для настроювань локалізації. Вони містять інформацію про тип грошової одиниці й інших параметрів, які можуть змінюватися залежно від країни	2000000.456m.ToString("C") Повертає "$2,000,000,46" (Якщо операційна система налаштована відповідно до американських стандартів)
D, d	Ціле число. Специфікатор точності встановлює мінімальне число цифр. Виведення доповнюється ведучими нулями, якщо кількість цифр фактичного числа менше, чим специфікатор точності. Примітка: цей символ формату застосовується тільки для цілочисельних типів)	45687.ToString("D8") Повертає: "00045678"
E, e	Експонентна (наукова) нотація. Специфікатор точності визначає кількість десяткових цифр, за замовчуванням рівне 6	345678900000.ToString("E3") Повертає "3.457E+011" 345678912000.ToString("e") Повертає "3.456789e011"
F, f	Фіксована точка. Специфікатор точності вказує кількість десяткових цифр	3.7667892.ToString("F3") Повертає "3.767"
G, g	Загальний. Найбільш компактний формат під час вибору E або F. Специфікатор точності встановлює максимальну кількість цифр у поданні числа	65432.98765.ToString("G") Повертає "65432.98765" 65432.98765.ToString("G7") Повертає "65432.99" 65432.98765.ToString("G4") Повертає "6.543E4"
N, n	Число. Число з комами-роздільниками. Специфікатор точності встановлює кількість десяткових цифр	1000000.123m.ToString("N2") Повертає "1,000,000.12"
X, x	Шістнадцятиричне число. Специфікатор точності встановлює мінімальну кількість цифр, що подаються в рядку. Для досягнення певної ширини додаються ведучі нулі	950.ToString("x") Повертає "3b6" 950.ToString("X6") Повертає "0003B6"

Метод ToString є потужним засобом для здійснення процесу форматування числових величин. Але його застосування виявляється незручним, якщо в рядок типу string вставляється декілька по-різному відформатованих чисел.

Наступний фрагмент ілюструє, яким заплутаним стає виклик WriteLine і як важко зрозуміти, яка частина є текстом, а яка – форматованим числом:

Console.WriteLine(“The length is: " + 10000000.4324ToString("N2") + "The widlh is: " + 65476356278.098746.ToString("N2") + "The height is: " + 4532554432.45684.ToString("N2"));

На екран виводиться таке:

The length is: 10,000,000.43 The width is: 65,476,356,278.10 The height is: 4,532,554,432.46

Вираз був би більш чітким, якби можна було розділити статичний текст і числа та вказати (за допомогою невеликого числа специфікаторів) позицію і формат кожного числа.

Мова C# надає рішення цієї проблеми.

Якщо на час відкинути форматування і використовувати специфікатор {<N>}, де <N> задає позицію числа в списку чисел, що йдуть після статичного тексту у виклику WriteLine, попередні рядки коду можна записати в такому вигляді:

Console.WriteLine(“The length is: {0} The width is: {1} The height is: {2}", 10000000.4324, 65476356278.098746, 4532554432.45684);

де {0} відноситься до першої величини (100000000.4324) в списку чисел після рядка тексту, {1} – до другої (65476356278.098746), а {2} – до третьої.

На специфікатор формату вказують фігурні дужки { }. Наприклад, виклик методу Console.WriteLine (рис. 35), приводить до наступного виведення на консоль:

Mass: 100 Distance 50 Energy 35

Специфікатор формату дозволяє також вказати ширину простору, що відводиться під виведення <Значення>. Для цього після індексу через кому задається ширина в символах.

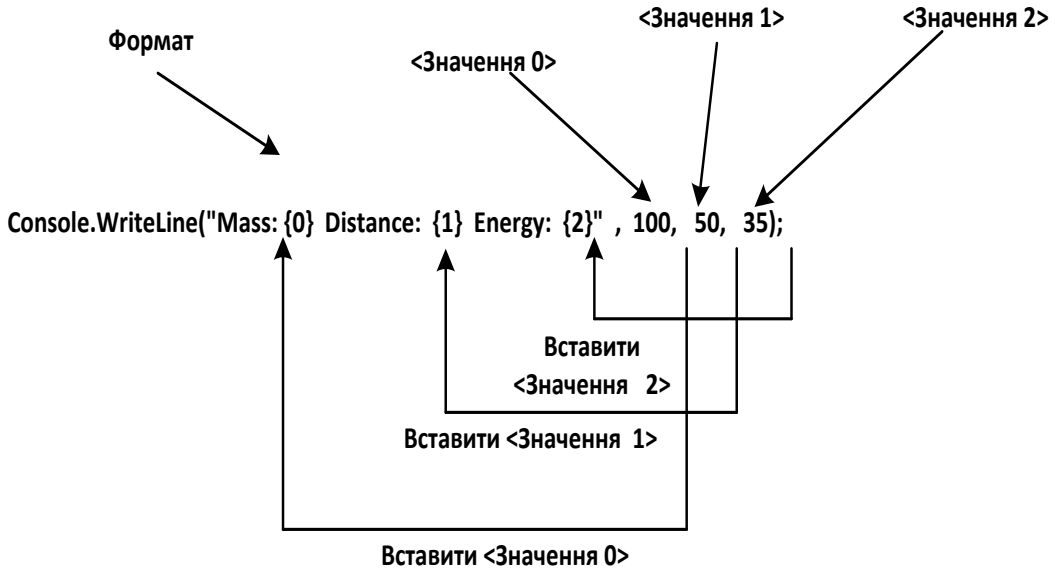


Рис. 35. **Приклад виклику методу Console.WriteLine**

Позитивне число означає вирівнювання праворуч, а негативне – ліворуч.

Наприклад:

Console.WriteLine(“Distance: {0,10} miles”, 100);

виводить

Distance: 100 miles

ширина дорівнює 10 символів і число вирівняне праворуч,

тоді, як код

Console.WriteLine(“Distance: {0,-10} miles”, 100);

негативне число вказує на вирівнювання ліворуч.

Виводить:

Distance: 100 miles

ширина дорівнює 10 символів і число вирівняне ліворуч.

Порядок виконання лабораторної роботи

Загальна частина

Набрати, відкомпілювати і запустити на виконання програми з конспекту лекцій "Основи програмування", які наведені в розділі 2 "Основні типи даних" [2, с. 21, 46] і в розділі 3 "Програмування лінійних обчислювальних процесів" [2, с. 50., 71, 73].

Опрацювати лекційний матеріал і переконатися, що робота програм відповідає їх опису.

Слід проекспериментувати з програмами:

• змінити вихідні дані;

• дослідити, як впливають синтаксичні помилки на результат компіляції програми. Які при цьому виникають помилки компіляції?

Індивідуальна частина

1. Отримати розрахункові формули у викладача (перелік можливих варіантів додається в роздатковому матеріалі до лабораторної роботи).

2. Проаналізувати отримані вирази: визначити допустимі діапазони зміни вхідних величин, їх розмірність і тип.

3. Підготувати контрольні приклади (використовуючи, наприклад калькулятор), які повною мірою характеризують аналізовані вирази (наприклад, особливі точки, в яких результат обчислення дорівнює нулю).

4. Розробити алгоритм обчислення і намалювати його графічну схему (блок-схему).

5. Відповідно до алгоритму набрати і відкомпілювати текст програми, усуваючи у разі необхідності помилки.

6. Дослідити роботу програми, аналізуючи виконання контрольних прикладів.

Зміст звіту

1. Титульний лист.

2. Цілі лабораторного заняття і вказівка, які навички та вміння передбачається отримати в результаті його виконання.

3. Тексти налагоджених програм загальної частини лабораторного заняття з необхідними коментарями і результатом виконання.

4. Аналіз вихідного виразу індивідуального завдання з обґрунтуванням контрольних прикладів, вибору типів даних і найбільш доцільною послідовністю операторів обчислень у вигляді відповідної графічної схеми.

5. Текст налагодженої програми з результатом виконання всіх контрольних прикладів індивідуального завдання.

6. Висновки.

Контрольні запитання

1. Опишіть відомі вам алгоритмічні структури.

2. Дайте огляд основних операцій C#.

3. Навіщо потрібні логічні операції? Наведіть приклади.

4. Що таке пріоритети операцій? Де і коли вони використовуються?

5. Що таке асоціативність операцій? Де і коли вони використовуються?

6. Розкрийте суть понять: простір імен, область видимості змінних, область видимості і час існування змінних.

7. Напишіть програму обчислення суми, добутку, максимуму і мінімуму трьох чисел.

8. Як перетворити значення типу string в тип int?

9. Опишіть загальну схему створення і виклику призначених для користувача методів.

10. Охарактеризуйте клас Math. Наведіть приклад програми, де використовуються вбудовані математичні методи.

Лабораторна робота № 3
Програмування обчислювальних процесів, що розгалужуються

Мета роботи – набуття практичних навичок з підготовки, налагодження і виконання програм, що розгалужуються.

Дана лабораторна робота сприяє напрацюванню таких **компетентностей** відповідно до Національної рамки кваліфікацій:

знання:

методики розробки програми із загальною лінійною частиною і кількома гілками;

алгоритмів виконання та синтаксис операторів if, if / else, switch і умовного виразу (?);

уміння:

складати програми з розгалуженнями;

виконувати налагодження та покрокове тестування програми з розгалуженнями в середовищі системи Visual C# .NET;

комунікації:

рекомендації команді учасників проекту щодо доцільності застосування поточного сценарію у вигляді алгоритмічних структур із розгалуженнями;

робота в команді над окремими частинами складного коду, який складається із структур вибору;

автономність і відповідальність:

прийняття рішення щодо розподілу початкового коду складної програми, в яку входять структури з розгалуженнями;

самостійний обґрунтування можливих варіантів C# ─ реалізацій структур вибору.

Основні положення

У лінійних програмах всі оператори виконувалися послідовно і, як наслідок, вони не здатні реагувати на поточні умови.

Однак часто в процесі реалізації поточного сценарію потрібно змінювати потік керування, реагуючи на якісь зовнішні події.

Потік керування становить порядок, в якому виконуються оператори програми. Крім того, часто використовуються терміни "порядок виконання" і "керуючий потік".

Гілкою називають сегмент програми, що містить один оператор або їх групу. Оператор розгалуження дозволяє запускати потрібний блок операторів. Вибір здійснюється за умовою. Оператори розгалуження часто називають операторами вибору.

C# забезпечує три типи структур вибору альтернатив:

єдиний вибір – структура if (ЯКЩО);

подвійний вибір – структура if / else (ЯКЩО / ІНАКШЕ);

множинний вибір – структура switch.

Структура вибору if.

Синтаксис оператора if відображений у такому синтаксичному блоці:

Оператор_if ::=

if(<Умова>)

<Оператор>;

Обрана тут форма запису – це спрощена версія часто використовуваної для опису синтаксису комп'ютерної мови нотації, яку називають формою Бекуса – Наура (Backus – Naur), або BNF. Вона була розроблена Дж. Бекусом і П. Науром.

Форма запису синтаксису складається з таких елементів:

Символ :: =, що означає "визначається як".

Метазмінні (розміщені в кутових дужках) в формі <Слово>.

Символ, що складається із двох квадратних дужок [ ] та позначає необов'язкові елементи [<це необов'язково>].

Три крапки ... які вказують на необмежену кількість елементів.

Вертикальна риса |, що вказує на можливі альтернативи.

Вираження логічного типу *(<Умова>)* завжди дає одне з двох значень: true (істина) або false (неправда).

<Оператор>, що слідує за логічним виразом, виконується лише в тому випадку, якщо останнє істинно.

Графічна схема оператора наведена на рис. 36.

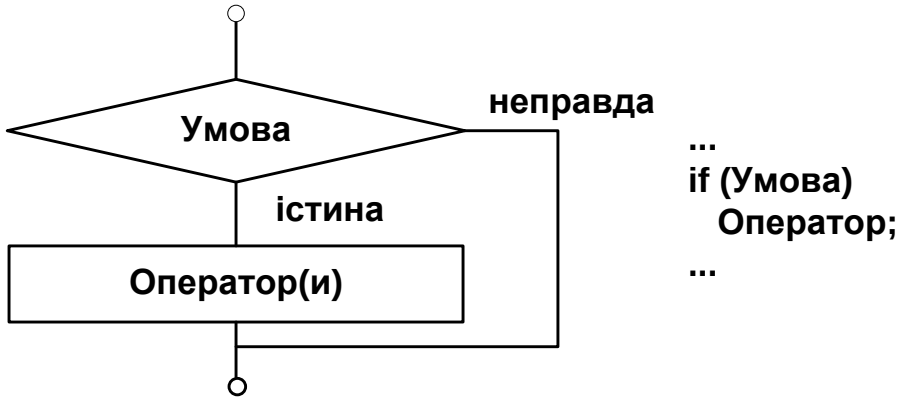


Рис. 36. **Графічна схема оператора if**

Приклад 1. Перевірка правильності введення змінної, яка може містити числа від 1 до 31.

```
using System;
        class Class1
        {
                static void Main( )
                {
                        int valor;
                        Console.WriteLine("Введіть число місяця");
                        valor = Convert.ToInt32(Console.ReadLine());
                        if (valor < 1  ||  valor >31)
                                Console.WriteLine("Помилка введення!");
                        Console.WriteLine("Ви ввели число, рівне {0}", valor);
                }
        }
```

Як оператори не можна використовувати оголошення і визначення. Однак тут можуть бути складені оператори і блоки:

```
Складений_оператор ::=
                        {
                        <Оператори>
                        }
```

Структура вибору if / else.

Синтаксичний блок оператора if / else:

Оператор if / else ::=

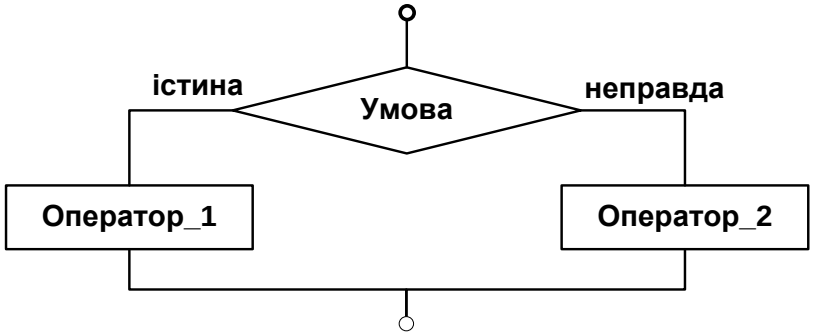
if (<Умова>)
<Оператор_1>; | <Складений_оператор_1>
else
<Оператор_2>; | <Складений_оператор_2>

<Оператор_1>; | <Складений_оператор_1> виконується лише в тому випадку, коли логічний вираз (<Умова>) дорівнює true.

<Оператор_2>; | <Складений_оператор_2> виконується лише тоді, коли (<Умова>) дорівнює false.

Перед else обов'язково ставиться крапка з комою.

Графічна схема оператора наведена на рис. 37.




```
...
if (Умова)
  Оператор_1;
else
  Оператор_2;
...
```

Рис. 37. **Графічна схема оператора if / else**

Приклад 2. Знайти мінімум із двох чисел.

```
using System;
    class Class1
    {
        static void Main()
        {
            int x, y, min;
            Console.WriteLine("Послідовно введіть два цілих числа");
            x = Convert.ToInt32(Console.ReadLine());
            y = Convert.ToInt32(Console.ReadLine());
            if  (x<y)
                min = x;
            else
                min = y;
            Console.WriteLine("Мінімальне число дорівнює {0}", min);
        }
    }
```

Оператори if можуть бути вкладені один в один.

Приклад 3. Знайти максимальне число із трьох чисел a, b, c.

```
using System;
        class Class1
        {
                static void Main()
                {

                        int a, b, c, max;
                        Console.WriteLine("Послідовно введіть три цілих
числа");
                        a = Convert.ToInt32(Console.ReadLine());
                        b = Convert.ToInt32(Console.ReadLine());
                        c = Convert.ToInt32(Console.ReadLine());
                        if  (a > b  &&  a > c)
                                max = a;
                        else
                                if  (b > c)
                                max = b;
                        else
                                max = c;
                        Console.WriteLine("Максимальне число дорівнює {0}",
max);
                }
        }
```

Обидві гілки повного умовного оператора можуть бути складеними.

Множинний вибір – структура switch

Оператор switch дозволяє програмі обрати одну з кількох дій на основі значення заданого виразу. Логіка, яка реалізована switch, подібна до логіки оператора if / else.

Синтаксичний блок оператора switch:

```
switch (<Вираз_switch >)
{
      case <Константний_вираз>:
            [ <Оператор>;
            <Оператор>; <Оператор>;
            <Оператор_break>; | <Оператор_goto> ]
```

case <Константний_вираз>:
[<Оператор>;
<Оператор>; <Оператор>;
<Оператор_break>; | <Оператор_goto>]
<Будь-яка кількість блоків case>
[default:
[<Оператор>;
<Оператор>; <Оператор>;
<Оператор_break>; | <Оператор_goto>]
}

Графічна схема оператора switch наведена на рис. 38.

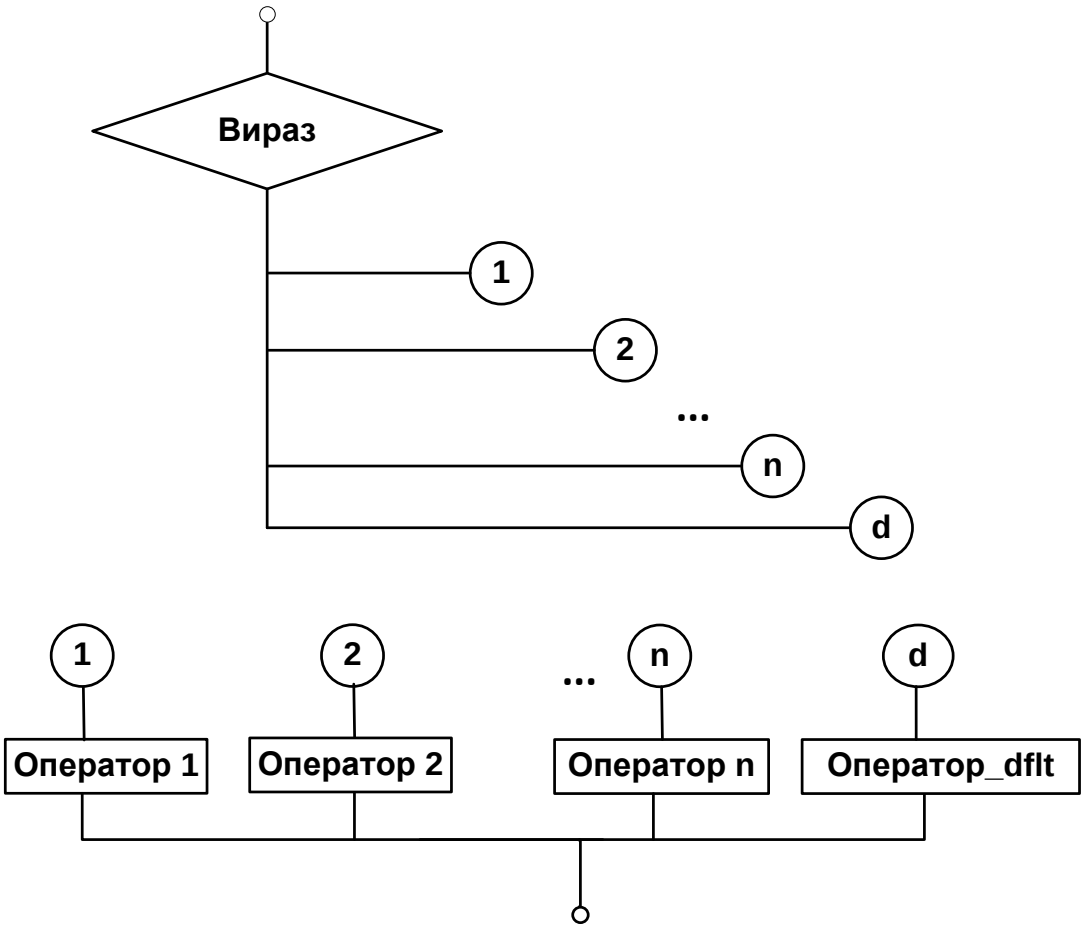


Рис. 38. **Графічна схема оператора switch**

Може бути один або не бути жодного блоку default.

Термін (<Вираз_switch>), використовуваний разом із ключовим словом switch, – це загальноприйнята назва керуючого виразу.

Розділи case і default називають розділами вибору.

<Константний_вираз>, що йде услід за ключовим словом case, називають значенням або case-міткою. Кожна з них повинна бути унікальною.

Найпоширеніший спосіб завершення розділу вибору – застосування <Оператор_break> або <Оператор_goto>.

Хоча розділи case і default можна розміщувати в будь-якій послідовності, гарний стиль програмування передбачає, що розділ default розміщується наприкінці оператора switch.

Коли потік керування переходить від одного розділу switch до іншого, таке виконання називається провалом. Для його запобігання застосовується оператор break або goto.

Приклад 4. Необхідно проаналізувати значення змінної nota, що є виставленою оцінкою.

```
using System;
class Nota
{
public static void Main()
{
    int nota;

    Console.WriteLine("Ваша оцінка (від 2 до 5)? ");
    nota = Convert.ToInt32(Console.ReadLine());

    switch(nota)
    {
            case 2:
                Console.WriteLine("Ви вибрали 2 ");
                Console.WriteLine("Оцінка - незадовільно");
                break;
            case 3:
                Console.WriteLine("Ви вибрали 3 ");
                Console.WriteLine("Оцінка - задовільно");
                break;
            case 4:
                Console.WriteLine("Ви вибрали 4 ");
                Console.WriteLine("Оцінка - добре");
                break;
```

```
                case 5:
                        Console.WriteLine("Ви вибрали 5 ");
                        Console.WriteLine("Оцінка - відмінно");
                        break;
                default:
                        Console.WriteLine("Помилка вибору. Ви повинні ви-
брати число " +
                        "між 2 and 5");
                        break;
            }
        }
        }
```

Приклад 5. Створення простого меню.

```
using System;
        class Class1
        {
                static void Main()
                {
                        char vibor;
                        Console.Write("МЕНЮ:\t A(dd) D(elete) S(ort) Q(uit)
\n");
                        Console.Write("Ваш вибір? ");
                        vibor = Convert.ToChar(Console.Read());
                        switch (Char.ToUpper(vibor))
                        {
                                case 'A':
                                        Console.Write("Обрано Add\n");
                                        break;
                                case 'D':
                                        Console.Write("Обрано Delete\n");
                                        break;
                                case 'S':
                                        Console.Write("Обрано Sort\n");
                                        break;
                                case 'Q':
                                        Console.Write("Обрано Quit\n");
                                        break;
```

```
                default:
                    Console.Write("Введений помилковий символ\n");
                    break;
                }
            Console.Write("\nДля завершення програми натисніть <Enter>");
            Console.ReadLine();  // для паузи
            Console.ReadLine();  // для паузи
            }
    }
```

Умовний вираз.

У C# є ще одна скорочена форма умовного оператора – так званий умовний вираз. Його синтаксис:

<Логічний_вираз_1> ? <Вираз_2> : <Вираз_3>;

Сенс цієї конструкції полягає в такому:

обчислюється <Логічний_вираз_1>; якщо він істинний (true), то результатом буде <Вираз_2>, у протилежному випадку результатом буде <Вираз_3>.

По суті, умовний вираз еквівалентний такому умовному операторові:

```
if  ( Логічний_вираз_1 )
        Вираз_2;
else
        Вираз_3;
```

Приклад. Знайти максимум із двох чисел.

```
int x = 13, y = 7, max;
max  =  (x > y) ? x : y;
Console.WriteLine("max = {0}", max);
```

Порядок виконання лабораторної роботи

Загальна частина

1. Набрати, відкомпілювати і запустити на виконання приклади програм, які були наведені в розділі "Основні положення" даної лабораторної роботи.

2. Проекспериментувати з програмами:

змінити вихідні дані;

дослідити, як впливають синтаксичні помилки на результат компіляції програми. Які при цьому виникають помилки компіляції?

Індивідуальна частина

1. Формули для обчислення у вигляді F = f (a, b, c, x) і опис змінних взяти з відповідного варіанта індивідуального завдання у викладача (див. додатковий файл з індивідуальними завданнями).

Завдання відповідає такому сценарію роботи:

Введіть чотири дійсні числа:

a =? <число_1> <Введення>

b =? < число _1> <Введення>

c =? < число _1> <Введення>

x =? < число _1> <Введення>

Обрана гілка № <виводиться номер гілки 1,2 або 3>

F = <виводиться результат обчислення>

Результат повинен містити три варіанти відповіді (по одному для кожної з гілок обчислювального процесу) і відповідні значення функції F = f (a, b, c, x), обчислені (для контролю) на калькуляторі.

2. Проаналізувати отримані вирази: визначити допустимі діапазони зміни вхідних величин, їх розмірність і тип.

3. Підготувати контрольні приклади (використовуючи, наприклад калькулятор), які повною мірою характеризують аналізовані вирази.

4. Розробити алгоритм обчислення і намалювати його графічну схему (блок-схему).

5. Відповідно до алгоритму набрати і відкомпілювати текст програми, усуваючи у разі необхідності помилки.

6. Дослідити роботу програми, аналізуючи виконання контрольних прикладів.

Зміст звіту

1. Титульний лист.

2. Цілі лабораторного заняття і вказівка, які навички та вміння передбачається отримати в результаті його виконання.

3. Тексти налагоджених програм загальної частини лабораторного заняття з необхідними коментарями і результатом виконання.

4. Аналіз вихідного виразу індивідуального завдання з обґрунтуванням контрольних прикладів, вибору типів даних і найбільш доцільною послідовністю операторів обчислень у вигляді відповідної графічної схеми.

Чисельні приклади виконання завдання для усіх гілок вихідного виразу.

5. Текст налагодженої програми з результатом виконання всіх контрольних прикладів індивідуального завдання.

6. Висновки.

Контрольні запитання

1. Дайте поняття потоку управління програмою.

2. Що становить структура вибору if, коли її треба використовувати?

3. Опишіть структуру вибору if / else.

4. У чому полягає специфіка множинного вибору І як цей вибір доцільно реалізувати за допомогою структури "switch"?

5. Умовний вираз. Наведіть його синтаксис і відповідний приклад застосування.

Лабораторна робота № 4
Програмування циклічних обчислювальних процесів

Мета роботи – набуття практичних навичок з підготовки, налагодження і виконання програм, що містять цикли.

Дана лабораторна робота сприяє напрацюванню таких **компетентностей** відповідно до Національної рамки кваліфікацій:

знання:

методики розробки програми, які мають цикли;

алгоритмів виконання та синтаксис стандартних операторів циклу: while, do / while, for;

уміння:

складати програми, які мають цикли;

виконувати налагодження та покрокове тестування типових циклічних програм у середовищі системи Visual C# .NET;

комунікації:

рекомендації команді учасників проекту щодо доцільності застосування поточного сценарію у вигляді циклічних алгоритмічних структур;

робота в команді з окремими частинами складного коду, який містить структури повторення;

автономність і відповідальність:

прийняття рішення щодо розподілу початкового коду складної програми, в яку входять циклічні структури;

самостійний обґрунтування можливих варіантів C# ─ реалізацій структур повторення.

Основні положення

Оператори циклу використовуються для організації багаторазово повторюваних обчислень.

Будь-який цикл складається з:

тіла циклу, тобто тих операторів, які виконуються кілька разів;

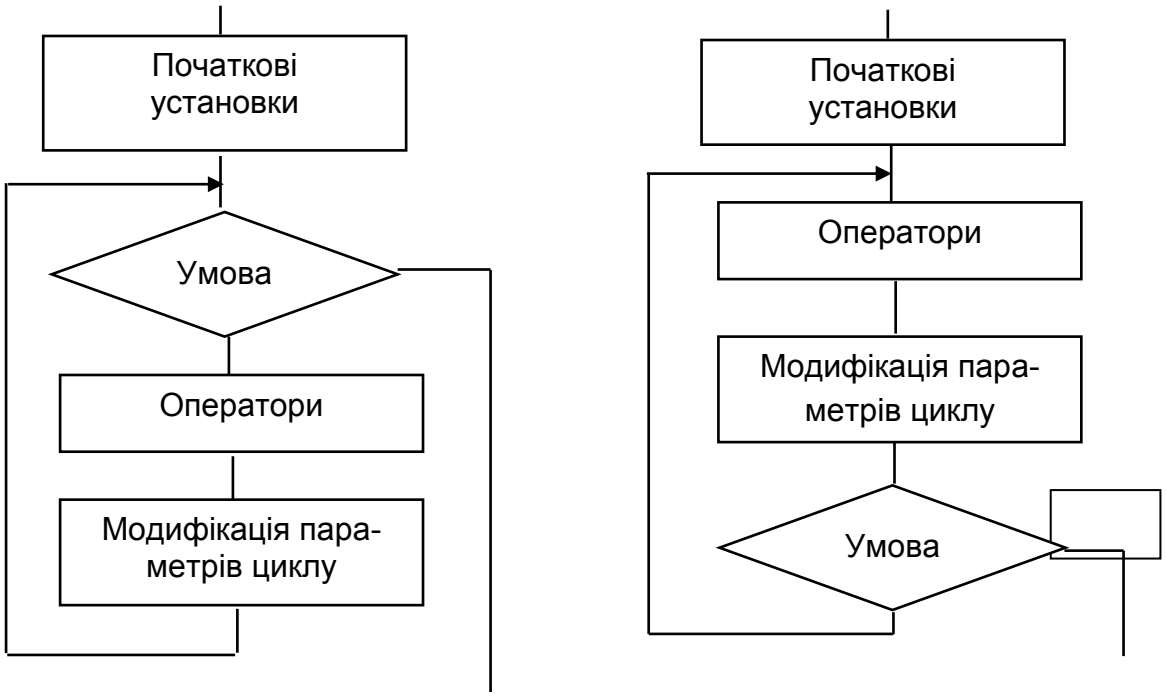
початкових установок;

модифікації параметра циклу;

перевірки умови продовження виконання циклу.

Один прохід циклу називається ітерацією. Перевірка умови виконується на кожній ітерації або до тіла циклу (цикл із передумовою), або після тіла циклу (цикл із постумовою).

Графічні схеми, що демонструють роботу циклу з передумовою (а) та постумовою (б) показані на рис. 39.



а) цикл із передумовою б) цикл із постумовою

Рис. 39. **Графічні схеми, що демонструють роботу циклів**

Різниця між ними полягає в тому, що тіло циклу з постумовою завжди виконується хоча б один раз, після чого перевіряється, чи треба його виконувати ще раз. Перевірка необхідності виконання циклу з передумовою робиться до тіла циклу, тому можливо, що він не виконається жодного разу.

Змінні, які змінюються в тілі циклу та використовуються у ході перевірки умови продовження, називаються параметрами циклу.

Цілочисельні параметри циклу, що змінюються з постійним кроком на кожній ітерації, називаються лічильниками циклу.

Початкові установки можуть явно не бути присутніми в програмі, їх сенс полягає в тому, щоб до входу в цикл задати значення змінним, які в ньому використовуються.

Цикл завершується, якщо умова його продовження не виконується.

Можливо примусове завершення як поточної ітерації, так і циклу в цілому. Для цього слугують оператори break, continue, return, goto. Передавати керування ззовні всередину циклу не рекомендується.

У C# є чотири різних оператори циклу – while, do while, for, foreach.

Цикл із передумовою (while).

Цикл із передумовою реалізує графічну схему (без блоку початкових установок), наведену на рис. 39 а, і має такий синтаксис:

Оператор_while ::=
while (<Умова_циклу>)
<Тіло_циклу>

де:
<Умова_циклу> ::= <Логічний_вираз>
<Тіло_цикпу> ::= <Оператор>;
::= <Складений_оператор>

Тіло циклу повторюється доти, доки <Умова_циклу> істинне (true).

Виконання оператора починається з обчислення логічного виразу (<Умова_циклу>). Якщо воно істинне (не дорівнює false), виконується <Тіло_циклу>.

Якщо під час першої перевірки вираз дорівнює false, цикл не виконується жодного разу.

Логічний вираз обчислюється перед кожною ітерацією циклу.

Приклад 1. Виведення алфавіту.

```
using System;
    class Class1
    {
        static void Main()
        {
```

```
            char c;
            Console.WriteLine("Виведення алфавіту:\n");
            c = 'A';
            while ( c <= 'Z' )
            {
                Console.Write("{0} ", c );
                c++;
            }
            Console.WriteLine(); // Переведення на новий рядок
        }
    }
```

Результат виконання програми:

Виведення алфавіту:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Розповсюджений прийом програмування – організація безкінечного циклу з заголовком while (true) і примусовим виходом з тіла циклу після виконання якої-небудь умови.

У круглих дужках після ключового слова while можна вводити опис змінної. Областю її дії є цикл, наприклад:

```
while (int x = 0)   { ... /* область дії x */ }
```

Приклад 2. Обчислення виразу f(x) = 1 + 1/2 + 1/3 + ... + 1/n.

```
using System;
    class Class1
    {
        static void Main()
        {
            int n;
            double f;
            Console.WriteLine("Введіть n (ціле)");
            n = Convert.ToInt32(Console.ReadLine());
            f = 0;
        while (n > 0)
            {
                f +=1.0 / n;
                n--;
            }
```

```
            Console.WriteLine("Значення функції дорівнює {0:N3}", f);
            }
    }
```

Результат виконання програми;

Введіть n (ціле)

10

Значення функції дорівнює 2,929

Цикл із постумовою (do / while).

Цикл із постумовою реалізує графічну схему (без блоку початкових установок), наведену на рис. 39 б, і має вигляд:

Оператор_do_while ::=

```
                        do
                          <Тіло_цикпу>
                        while (<Умова_циклу>);
```

де:

```
<Тіло_циклу>    ::=  <Оператор>;
                ::= <Складений_оператор>
<Умова_циклу>::= <Логічний_вираз>
```

<Тіло_циклу> повторюється до тих пір, поки <Умова_циклу> дорівнює true.

Спочатку виконується простий або складений оператор, що складають тіло циклу, а потім обчислюється логічний вираз (<Умова_циклу>). Якщо воно істинно (не дорівнює false), тіло циклу виконується ще раз.

Цикл завершується, коли (<Умова_циклу>) стане рівною false або в тілі циклу буде виконаний який-небудь оператор передачі керування.

Приклад 3. Виведення алфавіту.

```
using System;
class Class1
{
    static void Main()
    {
        char c = 'A';
        Console.WriteLine(" Виведення алфавіту:\n");
        c = Convert.ToChar(Convert.ToInt32('A')-1);
        do
        {
```

```
                c++;
                Console.Write("{0} ", c );
            } while ( c < 'Z' );
            Console.WriteLine(); // Переведення на новий рядок
        }
    }
```

Результат виконання програми:

Виведення алфавіту:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Оператор циклу for.

Цей оператор використовується в тих випадках, коли кількість повторень тіла циклу заздалегідь відома або може бути визначена програмним шляхом.

Його формат:

for([<Оператори_ініціалізації>];
    [<Умова_циклу>]; [<Оператори_оновлення>])
<Тіло_циклу>

де:

<Тіло_циклу> ::= <Оператор>;
::= <Складений_оператор>

<Оператор_ініціалізації> ::= <Оператор_ініціалізації1>,
<Оператор_ініціалізації2>...

<Умова_циклу> :: = <Логічний_вираз>

<Оператори_оновлення> ::= <Оператор_оновлення1>,
<Оператор_оновлення2>...

<Оператор_ініціалізації> використовується для оголошення та присвоєння початкових значень величинам, що використовуються в циклі. В цій частині можна записати декілька операторів, розділених комами.

Областю дії змінних, оголошених у частині ініціалізації циклу, є цикл.

Ініціалізація виконується один раз на початку виконання циклу.

<Умова_циклу> визначає умову виконання циклу: якщо його результат дорівнює true, цикл виконується.

<Оператори_оновлення> виконуються після кожної ітерації циклу й слугують зазвичай для зміни параметрів циклу. В частині модифікацій можна записати декілька операторів через кому.

Простий або складений оператор становить тіло циклу. Будь-яка з частин оператора for може бути опущена (але крапки з комою треба залишити на своїх місцях!).

Приклад 4. Розрахунок суми перших десяти цілих чисел.

```
using System;
class Program
   {
     static void Main(string[] args)
      {
         int s = 0;
         for(int i=0; i<=10; i++)
         {
            s = s + i;
         }
         Console.WriteLine("s = {0}", s);
      }
   }
```

Результат виконання програми:

s = 55

Вкладені цикли.

Вкладеним циклом називають конструкцію, в якій один цикл виконується всередині іншого. Внутрішній цикл виконується повністю під час кожної з ітерацій зовнішнього циклу.

Приклад 5. Потрібно заповнити заданий прямокутник символами "*".

```
using System;
class StarsTwoDimensions
{
        public static void Main()
        {
                for (int i = 0; i < 5; i++)
                {
                        for (int j = 0; j < 7; j++)
                        {
                                Console.Write("*");
                        }
```

```
                    Console.WriteLine();
                }
            }
        }
```

Результат виконання програми:

```
*******
*******
*******
*******
*******
```

У програмі можна використовувати будь-які комбінації вкладених циклів усіх типів: for, do / while, while, якщо цього вимагає логіка побудови програми.

Рекомендації з вибору циклів.

Оператори циклу взаємозамінні. Далі наведено деякі рекомендації з вибору найкращого оператора циклу в кожному конкретному випадку:

оператор do / while зазвичай використовують, коли цикл потрібно обов'язково виконати хоча б раз (наприклад, якщо в циклі здійснюється введення даних);

оператор for – переважне в більшості інших випадків (однозначно – для організації циклів із лічильниками);

оператором while зручніше користуватися у випадках, коли число ітерацій заздалегідь не відомо, очевидних параметрів циклу немає, або модифікацію параметрів зручніше записувати не наприкінці тіла циклу.

Оператор foreach буде розглянуто під час обробки масивів.

Порядок виконання лабораторної роботи

Загальна частина

1. Набрати, відкомпілювати і запустити на виконання прикладі програм, які були наведені в розділі "Основні положення" даної лабораторної роботи (всього 5 програм).

2. Проекспериментуйте з програмами:

замінити вихідні дані;

дослідити, як впливають синтаксичні помилки на результат компіляції програми. Які при цьому виникають помилки компіляції?

Індивідуальна частина

1. Формули для обчислення у вигляді F = f (a, b, c, x) і опис змінних взяти з відповідного варіанта індивідуального завдання у викладача (див. додатковий файл з індивідуальними завданнями).

Завдання відповідає такому сценарію роботи:

Введіть вихідні дані для розрахунку:

a =? <число_1> *<Введення>*

b =? <число_1> *<Введення>*

c =? <число_1> *<Введення>*

x_nach =? <число_1> *<Введення>*

x_kon =? <число_1> *<Введення>*

dx =? <число_1> *<Введення>*

Обчислити, і вивести на екран у вигляді таблиці значення функції F на інтервалі від Хнач. до Хкон. з кроком dX.

Результат в таблиці повинен охоплювати три варіанти відповіді (по одному для кожної з гілок обчислювального процесу) і відповідні значення функції F = f (a, b, c, x).

2. Проаналізувати отримані вирази: визначити допустимі діапазони зміни вхідних величин, їх розмірність і тип.

3. Підготувати контрольні приклади (використовуючи, наприклад калькулятор), які повною мірою характеризують аналізовані вирази.

4. Розробити алгоритм обчислення і намалювати його графічну схему (блок-схему).

5. Відповідно до алгоритму набрати і відкомпілювати текст програми, усуваючи у разі необхідності помилки.

6. Дослідити роботу програми, аналізуючи виконання контрольних прикладів.

Зміст звіту

1. Титульний лист.

2. Цілі лабораторного заняття і вказівка, які навички та вміння передбачається отримати в результаті його виконання.

3. Тексти налагоджених програм загальної частини лабораторного заняття з необхідними коментарями і результатом виконання.

4. Аналіз вихідного виразу індивідуального завдання з обґрунтуванням контрольних прикладів, вибору типів даних і найбільш доцільною послідовністю операторів обчислень у вигляді відповідної графічної схеми.

По одному варіанту чисельних прикладів виконання завдання для кожної з гілок вихідного виразу.

5. Текст налагодженої програми з результатом виконання всіх контрольних прикладів індивідуального завдання.

6. Висновки.

Контрольні запитання

1. Дайте загальний огляд структур повторення.

2. Опишіть особливості застосування циклу з передумовою (while). Наведіть приклад.

3. Коли доцільно застосовувати цикл з постумовою (do / while)? Наведіть приклад.

4. Дайте синтаксис оператора циклу for.

5. Наведіть загальні рекомендації відносно вибору циклів.

Змістовий модуль 2. Організація даних

Лабораторна робота № 5

Обробка одновимірних масивів і матриць

Мета роботи – набуття практичних навичок щодо обробки одновимірних масивів та освоєння методики опрацювання таблиць.

Дана лабораторна робота сприяє напрацюванню таких **компетентностей** відповідно до Національної рамки кваліфікацій:

знання:

призначення, оголошення й визначення масиву;

способів доступу до елементів масиву;

способів ініціалізації елементів масиву;

типових алгоритмів перетворення масивів;

алгоритмів сортування елементів масиву;

уміння:

складати програми, дані в яких надані у вигляді відповідних масивів;

виконувати налагодження та покрокове тестування типових програм перетворення масивів у середовищі системи Visual C# .NET;

розробляти програми формування багаторядкових табличних документів;

комунікації:

обґрунтування рекомендацій команді учасників проекту щодо доцільності застосування поданих даних у вигляді відповідних масивів;

робота в команді окремими частинами складного коду, який містить обробку масивів;

автономність і відповідальність:

прийняття рішення щодо доцільності застосування відповідних алгоритмів перетворення масивів;

самостійне обґрунтування можливих варіантів обробки таблиць.

Основні положення

Масив – це набір елементів одного і того ж типу, об'єднаних загальним ім'ям.

Масиви в C# можна використовувати за аналогією з тим, як вони використовуються в інших мовах програмування. Однак C# – масиви мають суттєві відмінності: вони відносяться до посилальних типів даних, більш того ─ реалізовані як об'єкти.

Виділення пам'яті під елементи відбувається на етапі ініціалізації масиву. А за звільненням пам'яті стежить система збору сміття ─ невикористовувані масиви автоматично утилізуються даною системою.

Одновимірний масив ─ це фіксована кількість елементів одного і того ж типу, об'єднаних загальним ім'ям, де кожен елемент має свій номер.

Нумерація елементів масиву в C# починається з нуля, тобто якщо масив складається з 10 елементів, то його елементи матимуть такі номери: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Одновимірний масив у C# реалізується як об'єкт, тому його створення є двоступеневим процесом. Спочатку оголошується посилальна змінна на масив, потім виділяється пам'ять під необхідну кількість елементів базового типу, і посилальної змінної присвоюється адреса нульового елемента в масиві. Базовий тип визначає тип даних кожного елемента масиву. Кількість елементів, які будуть зберігатися в масиві, визначається розмір масиву.

Загальний синтаксис для оголошення й створення масиву наведений в наступному синтаксичному блоці.

Оголошення масиву й присвоювання йому значення:

Оголошення_масиву ::=

<Базовий_тип> [ ] <Ідентифікатор масиву>;

Створення_масиву ::= new <Базовий_тип> [<Довжина_масиву>];

Присвоювання_посилання_на_масив ::= <Ідентифікатор_масиву> =

new <Базовий_тип> [<Довжина_масиву>];

Присвоювання_посилання_на_масив ::= <Оголошення_масиву>
new <Базовий_тип> [<Довжина_масиву>];

<Базовий_тип> в оголошенні повинен бути ідентичним <Базовий_тип> в операторі породження об'єкта.

<Довжина_масиву> повинна бути позитивною й належати типу, що неявно перетворюється до int. Це може бути літерал, константа або змінна.

Квадратні дужки [ ] в даному випадку є частиною синтаксису; вони не вказують на необов'язковість заключних у них елементів синтаксису.

Коли масив оголошений і присвоєне посилання на його об'єкт, можна звертатися і до його окремих елементів.

У загальному випадку процес оголошення змінної типу масив, і виділення необхідного обсягу пам'яті може бути розділене. Крім того, на етапі оголошення масиву можна провести його ініціалізацію. Тому для оголошення одновимірного масиву може використовуватися одна з таких форм запису:

Форма 1.

базовий_тіп [ ] ім'я__масива;

Наприклад:

int [ ] a.

Описано посилання на одновимірний масив, яке в подальшому може бути використане:

для адресації на вже існуючий масив;

передачі масиву в метод як параметр відстроченого виділення пам'яті під елементи масиву.

Форма 2.

базовий_тіп [ ] ім'я__масива = new базовий_тип [розмір];

Наприклад:

int [ ] a = new int [10];

Оголошено одновимірний масив заданого типу і виділена пам'ять під одновимірний масив зазначеного розміру. Адреса даної області пам'яті записана в посилальну змінну. Елементи масиву дорівнюють нулю.

Треба зазначити, що в C # елементам масиву присвоюються початкові значення за замовчуванням залежно від базового типу.

Для арифметичних типів — нулі, для посилальних типів — null, для символів — пробіл.

Форма 3.

базовий_тип [ ] ім'я__масива = {список ініціалізації};

Наприклад:

int [ ] a = {0, 1, 2, 3};

Виділена пам'ять під одновимірний масив, розмірність якого відповідає кількості елементів у списку ініціалізації. Адреса цієї області пам'яті записана в посилальну змінну. Значення елементів масиву відповідає списку ініціалізації.

Звернення до елементів масиву відбувається за допомогою індексу, для цього потрібно вказати ім'я масиву та в квадратних дужках його номер. Наприклад, a [0], b [10], c [i].

Оскільки масив є набором елементів, об'єднаних загальним ім'ям, то обробка масиву зазвичай проводиться в циклі.

Слід розглянути кілька простих прикладів роботи з одновимірними масивами.

Приклад 1. Програма виводить на екран монітору заданий масив у вигляді рядка

```
using System;
   class Program
   {
      static void Main(string[] args)
      {
                  int[ ] myArray = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
         int i;
         for (i = 0; i < 10; ++i)
            Console.Write(" {0}",myArray[i]);
         Console.WriteLine();
      }
   }
```

Завдання. Змініть програму так, щоб числа виводилися в стовпчик.

Приклад 2.

```
static void Main ()
{
  int [ ] myArray = new int [10];
```

```
  int i;
  for (i = 0; i <10; i ++)
       myArray [i] = i * i;
  for (i = 0; i <10; i ++)
       Console.WriteLine (myArray [i]);
}
```

Завдання. Змініть програму так, щоб оброблявся масив із n чисел.

Під час ініціалізації масиву немає необхідності використовувати операцію new, все ж масив можна задати такий спосіб:

```
int [ ] myArray = new int [ ] {99, 10, 100, 18, 78, 23, 163, 9, 87, 49};
```

Незважаючи на надмірність, дана форма ініціалізації масиву може виявитися корисною в тому випадку, коли вже існуючої посиланню на одновимірний масив присвоюється посилання на новий масив. наприклад:

```
static void Main ()
{
  int [ ] myArray = {0, 1, 2, 3, 4, 5};
  int i;
  for (i = 0; i <10; i ++)
       Console.Write ("" + myArray [i]);
  Console.WriteLine ("\ nНовий масив:");
  myArray = new int [ ] {99, 10, 100, 18, 78, 23, 163, 9, 87, 49}; // 1
  for (i = 0; i <10; i ++)
       Console.Write ("" + myArray [i]);
}
```

Слід зазначити, що спочатку змінна myArray посилалася на 6-ти елементний масив. У рядку 1 змінної myArray була привласнена посилання на новий 10-елементний масив, у результаті чого вихідний масив надалі не застосовується, оскільки на нього тепер не посилається жоден об'єкт. Тому він автоматично буде видалений складальником сміття.

Приклад 3. Обчислення функції y = a*x^2 + sin(x)

```
using System;
class Class1
{
static void Main()
       {
       const double a = 10.5;
```

```
            double [ ] mas = new double[7];
        //  double [ ] mas = {-1, -0.93, -0.49, 0, 1.13, 0.96, 1.75};
        //  double [ ] mas = new double[7]  {-1, -0.93, -0.49, 0, 1.13, 0.96,
1.75};
            double y;
            // Введення масиву
            Console.WriteLine("Введіть значення елементів масиву");
            for ( int i=0; i < mas.Length; i++)
                {
                    Console.Write("mas[{0}] = ",i);
                    mas[i] = Convert.ToDouble(Console.ReadLine());
                }
            // Контрольне виведення масиву
            Console.WriteLine("Вихідний масив:");
            for ( int i=0; i < mas.Length; i++)
                {
                    Console.Write("{0} ",mas[i]);
                }
            Console.WriteLine(); // переведення рядка
            //  Обчислення функції
            for ( int i=0; i < mas.Length; i++)
                {
                y = a*mas [ i ] * mas [ i ] - Math.Sin ( mas [ i ] );
                Console.WriteLine("При значенні x={0}, y={1:N4}",mas[i], y);
                }
            }
        }
```

Завдання. Змінити програму для обчислення такої функції: y = a + 1/x + 1/x^2 + 1/x^3 +1/x^4 + … +1/x^n, де параметр n задається з клавіатури.

Приклад 4. Пузиркове сортування.

```
using System;
class Class1
{
    static void Main()
    {
    const double a = 10.5;
```

```
        double [ ] mas = new double[7];
        // double [ ] mas = {-1, -0.93, -0.49, 0, 1.13, 0.96, 1.75};
        // double [ ] mas = new double[7] {-1, -0.93, -0.49, 0, 1.13, 0.96, 1.75};
        double y;
        // Введення масиву
        Console.WriteLine("Введіть значення елементів масиву");
        for ( int i=0; i < mas.Length; i++)
                {
                        Console.Write("mas[{0}] = ",i);
                        mas[i] = Convert.ToDouble(Console.ReadLine());
                }
        // Контрольне виведення масиву
        Console.WriteLine("Вихідний масив:");
        for ( int i=0; i < mas.Length; i++)
                {
                        Console.Write("{0}  ",mas[i]);
                }
        Console.WriteLine(); // переведення рядка
        //  Обчислення функції
        double t;
                for ( int i=0; i < mas.Length-1; i++)
                        for ( int j=0; j < mas.Length-1; j++)
                if(mas[j] < mas[j+1])      // варіант if(mas[j] > mas[j+1])
                        {
                        t=mas[j];
                        mas[j] = mas[j+1];
                        mas[j+1]=t;
                        }
        // Контрольне виведення масиву
                Console.WriteLine("Масив після сортування:");
                for ( int i=0; i < mas.Length; i++)
                {
                        Console.Write("{0}  ",mas[i]);
                }
                Console.WriteLine(); // переведення рядка
        }
}
```

Завдання. Змінити програму, щоб була можливість введення з клавіатури розмірності масиву.

Перебір елементів масиву за допомогою оператора foreach.

Крім циклу for, для проходження по всьому масиву можна скористатися оператором foreach.

Наприклад, для виведення значення кожного елемента масиву childbirths, оголошеного й визначеного як

```
uint [ ] childbirths =  {1340, 3240, 1003, 4987, 3877};
```

може використовуватися оператор foreach:

```
foreach (uint  temp  in  childbirths )
        {
          Console.WriteLine(temp);
        }
```

який дає виведення: 1340 3240 1003 4987 3877

Оператор foreach складається із заголовка й тіла (рис. 40). Тіло циклу може бути одиночним або складеним оператором. Перші два слова в круглих дужках заголовка є, відповідно, типом і ідентифікатором. Разом вони оголошують ітераційну змінну оператора foreach. У даному випадку вона називається temp (від слова temporary – тимчасовий).

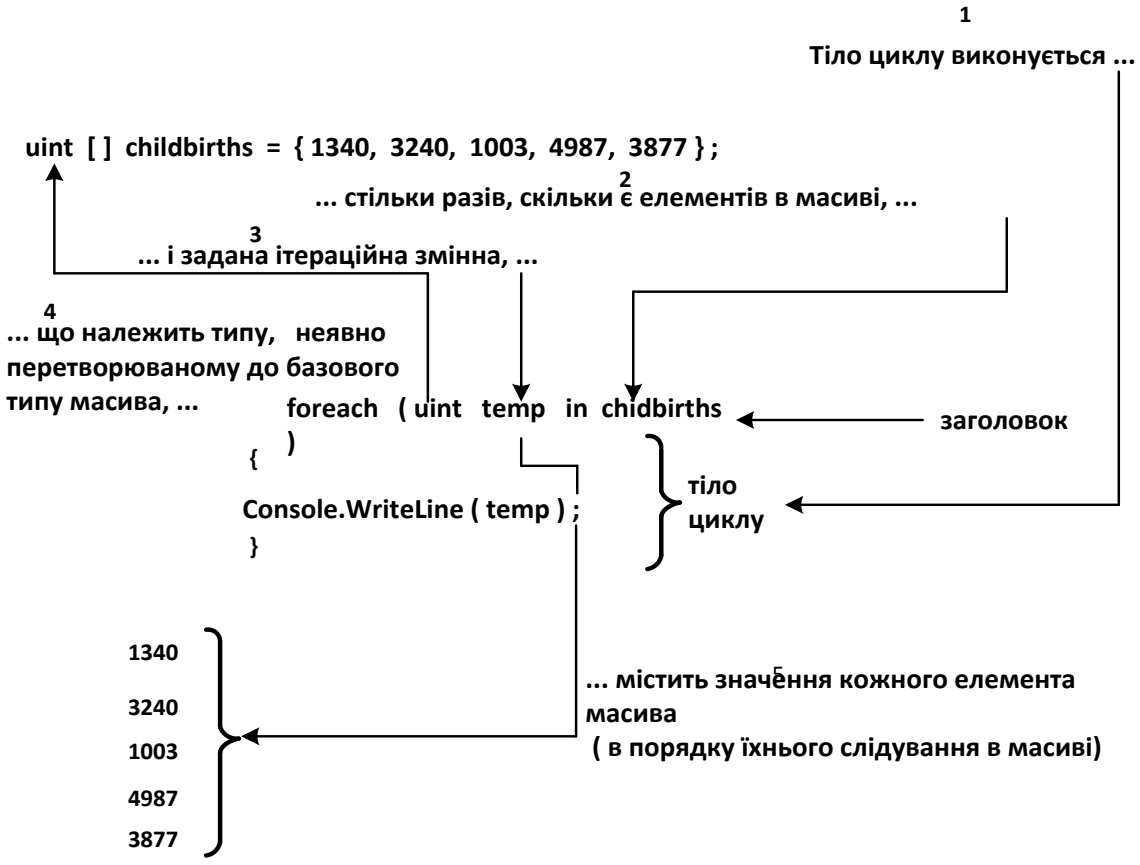


Рис. 40. **Оператор foreach**

Праворуч від ітераційної змінної знаходиться ключове слово in, за яким іде масив (у даному випадку childbirths). Важливо, щоб тип ітераційної змінної міг бути неявно перетворений в базовий тип масиву.

Тіло циклу виконується один раз для кожного елемента. Ітераційну змінну можна використовувати й у тілі циклу. Під час першого проходу (виконання) вона дорівнює першому елементу масиву й т.д.

Синтаксичний блок оператора foreach:

Оператор foreach ::=

foreach (<Тип> <Ідентифікатор_ітераційної_змінної>

in <Ідентифікатор_масиву>)

[< Оператор > | <Складений_оператор>]

Виконуючи оператор foreach, C# автоматично визначає кількість ітерацій. При цьому кожний елемент масиву присвоюється ітераційній змінній без необхідності явної індексації.

Лічильник, умова й оновлення циклу (необхідні в стандартному циклі for), в даному випадку непотрібні, що забезпечує простоту та ясність коду.

Багатовимірні масиви.

Оголошення й визначення двовимірного масиву.

Мова C# дозволяє визначати двовимірні масиви, де для ідентифікації елемента потрібні два індекси (фактично в C# можна визначати масиви будь-якої розмірності).

На рис. 41 наведено приклад оголошення й визначення двовимірного масиву.

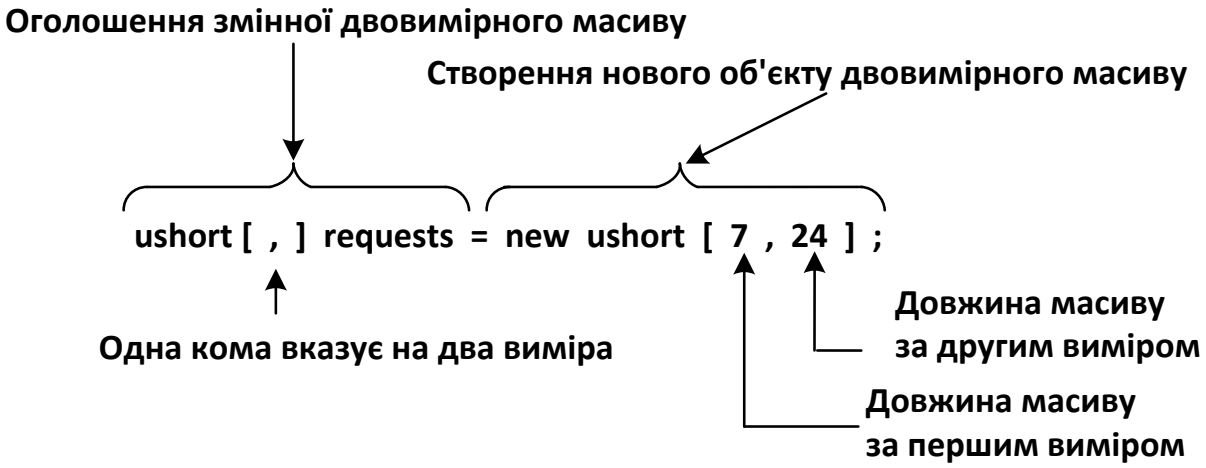
Оголошення змінної масиву, створення нового об'єкта і присвоювання змінній посилання на об'єкт (перший рядок на рис. 41) можна (як і у випадку одновимірного масиву) розбити на два оператори. Результат показано в нижній частині рис. 41.

Індекс першого виміру змінюється від 0 до 6, а другого – від 0 до 23.

Доступ до елементів двовимірного масиву.

Після оголошення змінної масиву і присвоювання їй посилання на двовимірний об'єкт можна звертатися до його окремих елементів.

За винятком того, що для звертання до елементів двовимірного масиву потрібен додатковий індекс, вони використовуються аналогічно елементам одномірного масиву. Отже, будь-який з них можна використовувати так само, як і окрему змінну базового типу.



Попередній оператор, як у випадку одновимірних масивів, можна розбити на два рядки:

```
ushort [ , ] requests ;
requsts = new ushort [ 7 , 24 ] ;
```

Рис. 41. **Приклад оголошення й визначення двовимірного масиву**

Наприклад:

requests[0,0] = (ushort) 89;

Тут застосовується операція приведення до типу (ushort), оскільки він є базовим типом requests. Вона необхідно, тому що літерал 89 належить до типу int, що не може бути неявно перетворений в ushort.

Подання двовимірного масиву як масиву масивів.

Два виміри масиву requests можна розглядати як масив масивів. Якщо звернутися до першого виміру requests (що наводить, наприклад, дні тижня), сім днів можна вважати одновимірним масивом (рис. 42). Якщо тепер включити в розгляд години, то кожний елемент "день" можна вважати складеним з одновимірного масиву "години".

Приклад 5. Обробка матриці зарплати.

```
using System;
class Class1
{
        static void Main()
        {
        int kol_rab;        // кількість робітників у бригаді
        int kol_brigad;     // кількість бригад
```

```
char vibor;          // для організації діалогу
do
{
//      Введення матриці із клавіатури
        Console.WriteLine("Введіть кількість робітників у бригаді...");
        kol_rab=Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Введіть кількість бригад...");
        kol_brigad=Convert.ToInt32(Console.ReadLine());
```

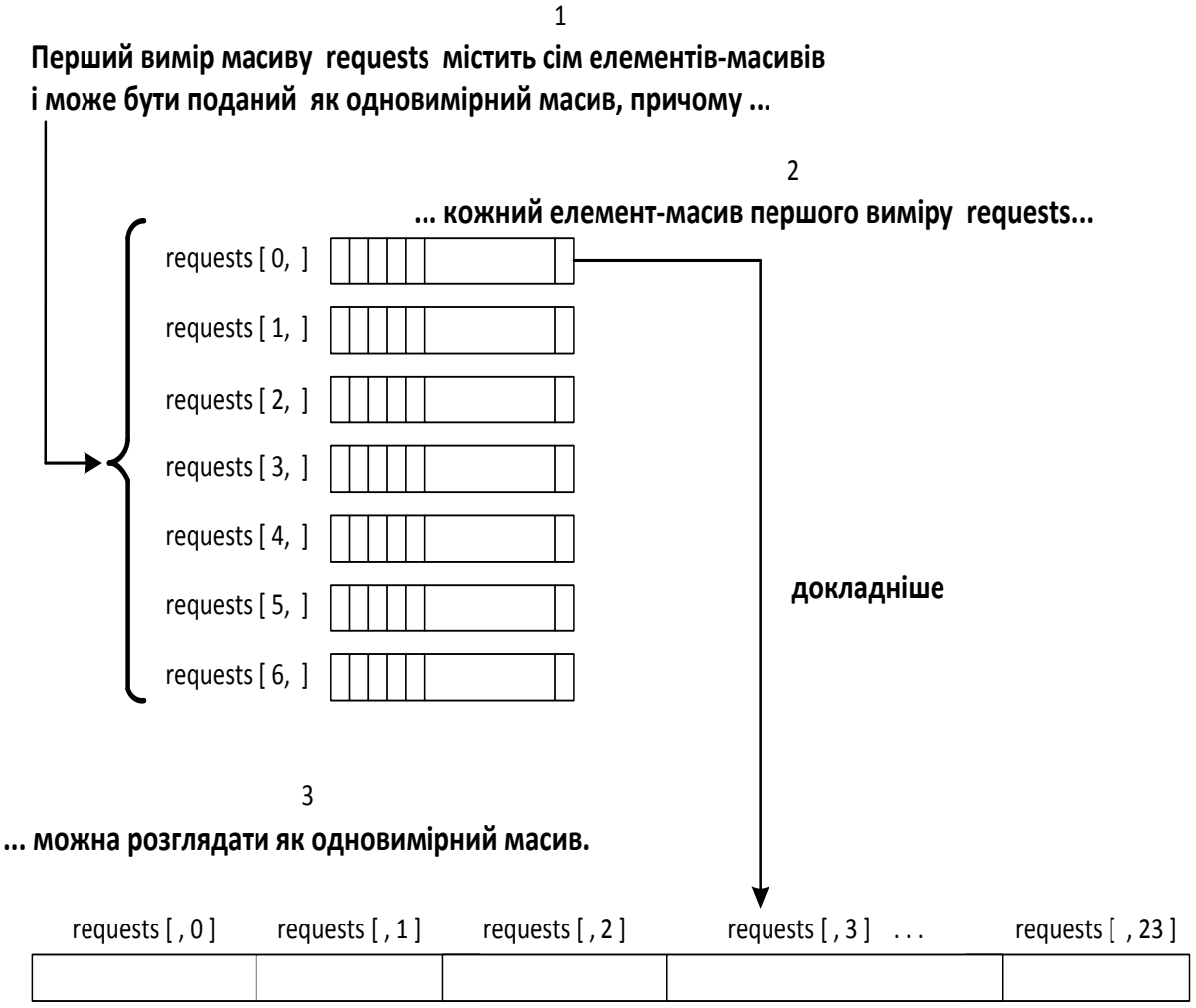


Рис. 42. **Двовимірний масив можна подати як два одновимірних масиви**

```
double  [,] Matr = new double[kol_rab,kol_brigad];
for( int i=0; i < kol_rab;  i++ )
{
        for (int j=0; j < kol_brigad;  j++)
```

```
                    {
                    Console.WriteLine("Введіть зарплату {0} робітника з
{1} бригади", i+1,j+1);
                    Console.WriteLine("Matr[{0},{1}]=?",i,j);
                    Matr[i,j]=Convert.ToDouble(Console.ReadLine());
                    }
                }
            // Контрольне виведення матриці
                Console.WriteLine("Матриця зарплат:");
                for( int i=0; i < kol_rab;  i++ )
                {
                    for (int j=0; j < kol_brigad;  j++)
                    {
                        Console.Write("{0}  ",Matr[i,j]);
                    }
                Console.WriteLine();  // переведення рядка
                }
                // Обробка матриці
                Console.WriteLine("Розрахунок максимальної зарплати
по бригадах:\n");
                Console.WriteLine("Номер бригади      Номер робітника
Зарплата");
                int  n_rab = 0,       // номер робітника
                n_brigad = 0;         // номер бригади
                double max_zarplata = 0;
                for( int j=0; j < kol_brigad;  j++ )
                {
                    for (int i=0; i < kol_rab ;  i++)
                    {
                    if(Matr[i,j]>max_zarplata)
                    {
                        max_zarplata = Matr[i,j];
                        n_rab = i;
                        n_brigad = j;
                    }
                    }
```

```
                Console.WriteLine("{0,7}{1,15}{2,15}",
n_brigad+1,n_rab+1,max_zarplata );
                max_zarplata = 0;  // підготовка до наступної ітерації
                }
            // Діалог для продовження роботи
            Console.WriteLine(" Будете продовжувати роботу? Так - <Y>,
Ні - <N> " );
            vibor=Convert.ToChar(Console.ReadLine());
            } while( (vibor == 'y') || (vibor == 'Y') ); // закінчення циклу do /
        while
            Console.WriteLine("Роботу завершено!" );
            }
        }
```

Масив як об'єкт.

Масиви в C# реалізовані як об'єкти. Якщо говорити більш точно, то вони реалізовані на основі базового класу Array, визначеного в просторі імен System. Даний клас містить різні властивості і методи. Наприклад, властивість Length дозволяє визначати кількість елементів у масиві. Інші властивості і методи класу Array наведені в табл. 6:

Таблиця 6

Властивості і методи класу Array

Елементи	Види	Опис
1	2	3
Length	Властивість	Кількість елементів масиву (за всіма вимірюванням)
BinarySearch	Статичний метод	Двійковий пошук у відсортованому масиві
Clear	Статичний метод	Присвоєння елементам масиву значень за замовчуванням
Copy	Статичний метод	Копіювання заданого діапазону елементів одного масиву в інший
CopyTo	Екземпляр методу	Копіювання всіх елементів поточного одновимірного масиву в інший масив
GetValue	Екземпляр методу	Отримання значення елемента масиву

Закінчення табл. 6.

1	2	3
IndexOf	Статичний метод	Пошук першого входження елемента в одновимірний масив
LastIndexOf	Статичний метод	Пошук останнього входження елемента в одновимірний масив
Reverse	Статичний метод	Зміна порядку проходження елементів на зворотний
SetValue	Екземпляр методу	Установка значення елемента масиву
Sort	Статичний метод	Упорядкування елементів одновимірного масиву

Виклик статичних методів відбувається через звернення до імені класу, наприклад, Array.Sort (myArray). В даному випадку відбудеться звернення до статичного методу Sort класу Array і передача даному методу як параметр об'єкт myArray – екземпляр класу Array.

Звернення до властивості або виклик екземпляра методу робиться через звернення до примірника класу, наприклад, myArray.Length або myArray.GetValue (i).

Приклад 6.

```
class Program
{
  static void Main ()
  {
    try
    {
      int [ ] MyArray;
      Console.Write ("Введіть розмірність масиву:");
      int n = int.Parse (Console.ReadLine ());
      MyArray = new int [n];
      for (int i = 0; i <MyArray.Length; ++ i)
      {
        Console.Write ("a [{0}] =", i);
        MyArray [i] = int.Parse (Console.ReadLine ());
      }
      PrintArray ("Вихідний масив:", MyArray);
      Array.Sort (MyArray);
```

```
            PrintArray ("Масив відсортований за зростанням", MyArray);
            Array.Reverse (MyArray);
            PrintArray ("Масив відсортований за спаданням", MyArray);
          }
          catch (FormatException)
          {
            Console.WriteLine ("Неправильний формат вводу даних");
          }
          catch (OverflowException)
          {
            Console.WriteLine ("Переповнення");
          }
          catch (OutOfMemoryException)
          {
            Console.WriteLine ("Недостатньо пам'яті для створення нового об'єкта");
          }
        }
        static void PrintArray (string a, int [ ] mas)
        {
          Console.WriteLine (a);
          for (int i = 0; i <mas.Length; i ++) Console.Write ("{0}", mas [i]);
          Console.WriteLine ();
        }
      }
      }
```

Завдання. Додати в програму метод InputArray, який призначено для введення з клавіатури елементів масиву. Продемонструвати роботу даного методу.

Порядок виконання лабораторної роботи

Загальна частина

1. Набрати, відкомпілювати і запустити на виконання приклади програм, які були наведені в розділі "Основні положення" даної лабораторної роботи. Звернути увагу на додаткові завдання, які наведені після лістингів програм.

2. Проекспериментувати з програмами:

змінити вихідні дані;

дослідити, як впливають синтаксичні помилки на результат компіляції програми. Які при цьому виникають помилки компіляції?

Індивідуальна частина

1. Написати програму, яка здійснює обробку одновимірного масиву, що складається з n дійсних елементів. Варіанти алгоритмів обробки взяти з додаткового файла з індивідуальними завданнями.

2. Написати програму, яка здійснює обробку квадратної матриці з n на n дійсних елементів. Варіанти алгоритмів обробки взяти з додаткового файла з індивідуальними завданнями.

3. Написати програму, яка формує в результаті діалогу табличний документ і здійснює його обробку. Варіанти табличних документів взяти з додаткового файла з індивідуальними завданнями.

Для пунктів 2 і 3 завдання підготувати чисельні контрольні приклади початкових масивів та результати їх обробки згідно з індивідуальним варіантом.

Для кожного з пунктів розробити графічну схему (блок-схему) відповідного алгоритму.

4. Відповідно до алгоритму набрати і відкомпілювати тексти програми, усуваючи у разі необхідності помилки.

6. Дослідити роботу програми, аналізуючи виконання контрольних чисельних прикладів.

Зміст звіту

1. Титульний лист.

2. Цілі лабораторного заняття і вказівка, які навички та вміння передбачається отримати в результаті його виконання.

3. Тексти налагоджених програм загальної частини лабораторного заняття з необхідними коментарями і результатом виконання.

4. Постановка задачі індивідуального завдання.

5. Для кожного з алгоритмів надати:

словесний опис його виконання, який супроводжується чисельним прикладом;

графічну схему (блок-схему) відповідного алгоритму.

6. Тексти налагоджених програм із результатом виконання всіх контрольних чисельних прикладів індивідуального завдання.

7. Висновки.

Контрольні запитання

1. У чому полягають особливості призначення, оголошення й визначення масиву?

2. Як відтворюється доступ до окремих елементів масиву?

3. Наведіть приклади варіантів ініціалізації масиву.

4. Опишіть загальну схему перебору елементів масиву за допомогою оператора foreach.

5. Наведіть приклади алгоритмів пошуку заданих елементів масиву.

6. Наведіть приклади алгоритмів перетворення масиву.

7. У чому суть алгоритму сортування елементів масиву методом "Пузирку"?

8. Як реалізується доступ до елементів двовимірного масиву?

9. У чому суть подання двовимірного масиву як масиву масивів?

10. Наведіть приклади обробки матриць.

Лабораторна робота № 6
Обробка структур

Мета роботи – набуття практичних навичок щодо обробки структур та освоєння на їх основі методики опрацювання таблиць.

Дана лабораторна робота сприяє напрацюванню таких **компетентностей** відповідно до Національної рамки кваліфікацій:

знання:

призначення, оголошення й визначення структур;

способів доступу до елементів структур;

способів ініціалізації елементів структур;

типових алгоритмів застосування структур;

уміння:

складати програми, дані в яких надані у вигляді відповідних структур;

виконувати налагодження та покрокове тестування типових програм обробки масивів структур у середовищі системи Visual C# .NET;

розробляти програми формування багаторядкових табличних документів;

комунікації:

обґрунтування рекомендацій команді учасників проекту щодо доцільності застосування поданих даних у вигляді відповідних структур та їх масивів;

робота в команді над окремими частинами складного коду, який містить обробку структур;

автономність і відповідальність:

прийняття рішення щодо доцільності застосування відповідних алгоритмів обробки структур та їх масивів;

самостійне обґрунтування можливих варіантів обробки таблиць на базі структур.

Основні положення

Структури – це складені типи даних, побудовані з використанням інших типів. Вони становлять об'єднаний загальним ім'ям набір даних різних типів.

Окремі дані структури називаються елементами або полями. Елементи однієї й тієї ж структури повинні мати унікальні імена, але дві різні структури можуть містити неконфліктуючі елементи з однаковими іменами.

Синтаксичний блок визначення структури має такий вид:

```
[ <Специфікатор_доступності> ]  struct  <Ідентифікатор_структури>
                                              [ <Список_інтерфейсів> ]
        {
                    <Елементи_структури>
        };
```

Відповідно до синтаксису мови, опис структури починається зі службового слова struct, услід за яким міститься обране користувачем ім'я типу. Елементи, що входять у структуру, розміщуються в фігурних дужках, після яких ставиться крапка з комою. Елементи структури можуть мати вбудований або похідний тип.

Опис структури не резервує ніякого простору в пам'яті, він тільки створює новий тип даних, що може використовуватися для визначення змінних. У структурі обов'язково повинен бути вказаний хоча б один компонент.

Нехай необхідно створити тип для опису характеристики викладача університету. Цей тип повинен містити ім'я викладача, його кваліфікацію (гарна, задовільна тощо), стаж роботи і поточну якість викладання (за 12-бальною оцінкою). Далі наведено опис структури, що задовольняє ці вимоги:

```
struct Profesor
{
        public string Nombre;            // ім'я
        public string Calificacion;      // кваліфікація
        public int  Aprendizaje;         // стаж
        public double Calidad;           // якість
} ;
```

Ключове слово struct указує на те, що код визначає тип структури. Ідентифікатор Profesor – назва для цього типу. Таким чином, тепер можна створювати змінні типу Profesor так само, як змінні будь-якого базового типу, наприклад int або char.

Між фігурними дужками знаходиться список полів структури. Кожний елемент списку – це оператор визначення. Тут можна використовувати будь-який з типів даних C#, включаючи масиви та інші стуктури. В цьому прикладі використовуються два масиви типу string, зручні для збереження рядків з атрибутами “Ім'я” і “Кваліфікація”, а також змінні int і double – для зберігання відповідних числових значень.

Тепер, коли структура оголошена, її можна використовувати. Для цього спочатку потрібно створити (визначити) екземпляр структури.

Екземпляр структури створюється за допомогою ключового слова new:

```
Profesor  P_Econom_Inform = new Profesor( );
```

але, на відміну від класу, екземпляр структури можна створити і без new. Це виглядає в такий спосіб:

```
Profesor  P_Econom_Inform;
```

Під час створення структури без ключового слова new її конструктори не викликаються. При цьому значення всім її елементам слід присвоїти явно, звернувшись до них через ім'я структури, як показано далі:

```
P_Econom_Inform.Nombre = "Браткевич В'ячеслав";
        P_Econom_Inform.Calificacion = "задовільна";
        P_Econom_Inform.Aprendizaje = 32;
        P_Econom_Inform.Calidad = 7.59;
```

Ініціалізацію не можна виконати через методи або властивості, оскільки жоден з елементів-функцій не може бути викликаний, поки не будуть ініціалізовані елементи-дані. Тому останні потрібно оголошувати як public**.**

Приклад 1. Оголошення, визначення (без new), ініціалізація та виведення на екран структури Profesor.

```
using System;
// Опис структури Profesor
struct Profesor
{
        public string Nombre;           // ім'я
        public string Calificacion;     // кваліфікація
        public int  Aprendizaje;        // стаж
        public double Calidad;          // якість
} ;
class Class1
{
        static void Main()
        {
                Profesor    P_Econom_Inform; // Оголошення екземпляра структури
                //      Роздільна ініціалізація полів структури
                P_Econom_Inform.Nombre = "Браткевич В'ячеслав";
                P_Econom_Inform.Calificacion = "задовільна";
                P_Econom_Inform.Aprendizaje = 32;
                P_Econom_Inform.Calidad = 7.59;
                // Контрольне виведення
                Console.WriteLine("Викладач {0}: \nКваліфікація - {1};"+
                "\nСтаж - {2};\nЯкість - {3}", P_Econom_Inform.Nombre,
                P_Econom_Inform.Calificacion,
P_Econom_Inform.Aprendizaje,
                P_Econom_Inform.Calidad);
        }
}
```

Результат роботи програми:

Викладач Браткевич В'ячеслав:

Кваліфікація – задовільна;

Стаж – 32;

Якість – 7,59.

Приклад 2. Оголошення, визначення (з new), ініціалізація та виведення на екран структури Profesor.

```
using System;
// Опис структури Profesor
struct Profesor
{
        public string Nombre;           // ім'я
        public string Calificacion;     // кваліфікація
        public int  Aprendizaje;        // стаж
        public double Calidad;          // якість
} ;
        class Class1
{
        static void Main()
        {
                // Ініціалізації елементів структури за замовчуванням
                Profesor  P_Econom_Inform = new Profesor();
                // Контрольне виведення
                Console.WriteLine("Викладач  {0}:  \nКваліфікація  -  {1};"+
"\nСтаж - {2};\nЯкість - {3}\n", P_Econom_Inform.Nombre,
        P_Econom_Inform.Calificacion, P_Econom_Inform.Aprendizaje,
        P_Econom_Inform.Calidad);
                // Роздільна ініціалізація полів структури
                P_Econom_Inform.Nombre = "Браткевич В'ячеслав";
                P_Econom_Inform.Calificacion = "задовільна";
                P_Econom_Inform.Aprendizaje = 32;
                P_Econom_Inform.Calidad = 7.59;
                // Контрольне виведення
                Console.WriteLine("Викладач  {0}:  \nКваліфікація  -  {1};"+
"\nСтаж - {2};\nЯкість - {3}\n", P_Econom_Inform.Nombre,
        P_Econom_Inform.Calificacion, P_Econom_Inform.Aprendizaje,
                                                P_Econom_Inform.Calidad);
                Console.Read();  // для паузи
        }
}
```

Результат роботи програми:

Викладач :

Кваліфікація ─

Стаж ─ 0;

Якість ─ 0

Викладач Браткевич В'ячеслав:
Кваліфікація – задовільна;
Стаж – 32;
Якість – 7,59
Масиви структур.

Приклад 3. Обробка масиву структур.

```
using System;
struct Stroka
{
        public string name;                 // Автор книги
        public double stoimost;             // Вартість виданої книги
        public int kolich;                  // Кількість виданих книг
        public double sum_stoimost;         // Вартість виданих книг
};
class Class1
{
        static void Main()
        {
                //      Введення вихідних даних
                Console.WriteLine("Введіть кількість рядків у документі");
                int kol = Convert.ToInt32(Console.ReadLine());
                Stroka[ ] Tabl = new Stroka[kol];
                for( int i=0; i < Tabl.Length; i++)
                {
                        Console.WriteLine("Автор книги?");
                        Tabl[i].name = Console.ReadLine();
                        Console.WriteLine("Вартість книги?");
                Tabl[i].stoimost = Convert.ToDouble(Console.ReadLine());
                        Console.WriteLine("Кількість книг?");
                        Tabl[i].kolich = Convert.ToInt32(Console.ReadLine());
                }
                //  Виконання розрахунків:
                double s1=0, s2=0, s3=0;
                for( int i=0; i < Tabl.Length; i++)
                {
                        Tabl[i].sum_stoimost = Tabl[i].stoimost * Tabl[i].kolich;
```

```
                    s1 += Tabl[i].stoimost;
                    s2 += Tabl[i].kolich;
                    s3 += Tabl[i].sum_stoimost;
                }
            //  Побудова "шапки" таблиці
            Console.WriteLine("\nВідомості про вартість виданих книг\n");
            Console.WriteLine("|-----------------------------------------------------------
--|");
            Console.WriteLine("|  n/n |  Автор  |  Вартість  | Видано |  Витра-
ти  |");
            Console.WriteLine("|-----------------------------------------------------------
--|");
                // Заповнення таблиці даними:
                for( int i=0; i < Tabl.Length; i++)
                {
                    Console.WriteLine("|{0,5}{1,20}{2,14}{3,9}{4,10:N2}  |",
                        i+1, Tabl[i].name, Tabl[i].stoimost, Tabl[i].kolich,
                        Tabl[i].sum_stoimost);
                }
            Console.WriteLine("|-----------------------------------------------------------
--|");
            Console.WriteLine("| Разом: {0,31} {1,8} {2,9:N2}  |",s1, s2, s3);
            Console.WriteLine("|-----------------------------------------------------------
--|");
            }
        }
```

Результат роботи програми:

```
Введіть кількість рядків у документі
3
Автор книги?
Рубіна
Вартість книги?
34,45
Кількість книг?
3
Автор книги?
Улицька
```

Вартість книги?
45,78
Кількість книг?
10
Автор книги?
Пушкін
Вартість книги?
75,23
Кількість книг?
3

Відомості про вартість виданих книг

n/n	Автор	Вартість	Видано	Витрати
1	Рубіна	34,45	3	103,35
2	Улицька	45,78	10	457,80
3	Пушкін	75,23	3	225,69
Разом:		155,46	16	786,84

Порядок виконання лабораторної роботи

Загальна частина

1. Набрати, відкомпілювати і запустити на виконання приклади програм, які були наведені в розділі "Основні положення" даної лабораторної роботи.

2. Проекспериментувати з програмами:

змінити вихідні дані;

дослідити, як впливають синтаксичні помилки на результат компіляції програми. Які при цьому виникають помилки компіляції?

Індивідуальна частина

Написати програму, яка на базі відповідної структури формує в результаті діалогу табличний документ і здійснює його обробку. Варіанти табличних документів взяти з додаткового файла з індивідуальними завданнями.

Розробити графічну схему (блок-схему) відповідного алгоритму.

Набрати і відкомпілювати тексти програми, усуваючи у разі необхідності помилки.

Дослідити роботу програми, аналізуючи виконання контрольних чисельних прикладів.

Зміст звіту

1. Титульний лист.

2. Цілі лабораторного заняття і вказівка, які навички та вміння передбачається отримати в результаті його виконання.

3. Тексти налагоджених програм загальної частини лабораторного заняття з необхідними коментарями і результатом виконання.

4. Постановка задачі індивідуального завдання і словесний опис його виконання, який супроводжується чисельним прикладом; графічну схему (блок-схему) відповідного алгоритму.

5. Текст налагодженої програми з результатом виконання контрольних чисельних прикладів індивідуального завдання.

6. Висновки.

Контрольні запитання

1. Коли доцільно використовувати структури?

2. Як реалізується призначення, оголошення й визначення структур?

3. Наведіть приклади оголошення, визначення, ініціалізації та виведення на екран елементів заданої структури (без застосування операції **new)**.

4. Наведіть приклади оголошення, визначення, ініціалізації та виведення на екран елементів заданої структури з застосуванням операції **new**.

5. Які особливості обробки елементів структур?

Лабораторна робота № 7
Використання функцій

Мета роботи – набуття практичних навичок щодо опису та застосування функцій.

Дана лабораторна робота сприяє напрацюванню таких **компетентностей** відповідно до Національної рамки кваліфікацій:

знання:

призначення функції;

використання функції;

значення, що повертаються;

параметри функцій та їх відповідність;

обмін інформацією з функцією;

функції і масиви;

уміння:

складати програми, обробка даних в яких здійснюється за допомогою відповідних функцій;

виконувати налагодження та покрокове тестування програм, структура яких містить функції;

комунікації:

обґрунтування рекомендацій команді учасників проекту щодо доцільності застосування відповідних функцій згідно з певним сценарієм;

робота в команді над окремими частинами складного коду, який містить перелік функцій;

автономність і відповідальність:

прийняття рішення щодо доцільності розподілу вихідного алгоритму на певні частини у вигляді функцій;

самостійне обґрунтування можливих варіантів реалізацій відповідних функцій.

Основні положення

Досить часто виникають ситуації, коли виконання певних задач – наприклад, пошук максимального елемента масиву – необхідно здійснювати в різних місцях програми. Рішенням такого роду проблем є застосування функцій**.**

Функції в C# – це засіб, що дозволяє виконувати деякі ділянки коду в довільному місці додатка.

Як уже зазначалося, мова C# є об'єктно-орієнтованою і тому всі програми на її основі містять класи. Класи необхідні для опису об'єктів, на базі яких створюються їх певні екземплярів. Реалізація сценарію (алгоритму) здійснюється як взаємодія об'єктів. Взаємодія відбувається за допомогою виклику методів і подальшому обміну відповідною інформацією між ними.

Метод — це функціональний елемент класу, який реалізує обчислення або інші дії, що виконуються класом або його екземпляром (об'єктом).

Таким чином, в першому наближенні (в рамках процедурного стилю програмування) метод – це функція в класі. Тому надалі поняття методу і функції в поточній лабораторній роботі розглядаються як синоніми.

Детальний огляд методів буде надано під час вивчення дисципліни "Основи об'єктно-орієнтованого програмування" у наступному семестрі.

Метод є закінченим фрагментом коду, до якого можна звернутися по імені. Він описується один раз, а викликатися може багаторазово. Сукупність методів класу визначає, що конкретно може робити клас. Наприклад, стандартний клас Math містить методи, які дозволяють обчислювати значення математичних функцій.

Синтаксис методу:

[атрибути] [специфікатори] тип_що_повертається ім'я_метода ([список_параметрів])

```
{
    тіло_метода;
    return значення;
}
```

де:

атрибути і специфікатори є необов'язковими елементами синтаксису опису методу. На даному етапі атрибути використовуватися не будуть, а з усіх специфікаторів в обов'язковому порядку буде використано специфікатор static, який дозволить звертатися до методу класу без створення його екземпляра. Решту специфікаторів буде розглянуто в відповідному розділі дисципліни "Основи об'єктно-орієнтованого програмування";

"тип_що_повертається" визначає тип значення, що повертається методом. Це може бути будь-який тип, включаючи типи класів, створювані програмістом. Якщо метод не повертає ніякого значення, необхідно вказати тип void (в цьому випадку в тілі методу оператор return відсутній);

"ім'я_метода" ― ідентифікатор, заданий програмістом з урахуванням вимог, що накладаються на ідентифікатори в C#, відмінний від тих, які вже використані для інших елементів програми в межах поточної області видимості;

"список_параметрів" є послідовністю пар, що складаються з типу даних та ідентифікатора, розділених комами. Параметри ― це змінні або константи, які отримують значення, передані методу під час виклику.

Якщо метод не має параметрів, то "список_параметрів" залишається порожнім;

"значення" визначає значення, що повертається методом. Тип значення повинен відповідати типу "тип_що повертається" або приводиться до нього.

Приклад 1. Виклик методу без параметрів.

```
using System;
class Program
{
   static void Func() // додатковий метод
   {
      Console.Write("x =");
      double x = double.Parse(Console.ReadLine());
      double y = 1 / x;
      Console.WriteLine("y ({0}) = {1}", x, y);
   }
   static void Main() // точка входу в програму
   {
      Func(); // перший виклик методу Func
      Func(); // другий виклик методу Func
   }
}
```

Результат виконання програми:

```
x =5
y (5) = 0,2
x =7
y (7) = 0,142857142857143
```

У даному прикладі в метод Func не передаються ніякі значення, тому список параметрів порожній. Крім того, метод нічого не повертає, тому тип значення void. В основному методі Main метод Func викликається два рази. Якщо буде необхідно, то даний метод можна буде викликати ще стільки раз, скільки буде потрібно для вирішення задачі.

Завдання.

1. Додати в метод Main третій виклик методу Func.
2. Перетворити програму так, щоб метод Func викликався n раз.

Приклад 2. Буде змінено приклад 1 так, щоб в нього передавалося значення x, а сам метод повертав значення y.

```
using System;
class Program
{
   static double Func(double x) // додатковий метод
   {
      return 1 / x; // Значення, що повертається
   }

   static void Main() // точка входу в програму
    {
      Console.Write ("a =");
      double a = double.Parse (Console.ReadLine ());
      Console.Write ("b =");
      double b = double.Parse (Console.ReadLine ());
      for (double x = a; x <= b; x += 0.5)
       {
         double y = Func (x); // виклик методу Func
         Console.WriteLine ("y ({0}) = {1}", x, y);
       }
   }
}
```

Результат виконання програми:

```
a =3
b =5
y (3) = 0,333333333333333
y (3,5) = 0,285714285714286
y (4) = 0,25
y (4,5) = 0,222222222222222
y (5) = 0,2
```

У даному прикладі метод Func містить параметр x, тип якого double. Для того, щоб метод Func повертав у метод Main, який викликає його, значення виразу 1/x (тип якого double), перед ім'ям методу вказується тип значення — double, а в тілі методу використовується оператор передачі управління — return. Оператор return завершує виконання методу і передає управління в точку його виклику.

Завдання. Перетворити програму так, щоб метод Func повертав значення виразу: x^2 при x >= 0, або 1/x при x < 0.

Приклад 3. Вкладення одного виклику в інший.

```
class Program
{
   static int Func(int x, int y) // рядок 1
   {
      return (x > y) ? x : y;
   }

   static void Main()
   {
      Console.Write("a =");
      int a = int.Parse(Console.ReadLine());
      Console.Write("b =");
      int b = int.Parse(Console.ReadLine());
      Console.Write("c =");
      int c = int.Parse(Console.ReadLine());
      int max = Func(Func(a, b), c); // рядок 2 - виклики методу Func
      Console.WriteLine("max ({0}, {1}, {2}) = {3}", a, b, c, max);
   }
}
```

Результат виконання програм:

```
a =3
b =5
c =7
max (3, 5, 7) = 7
```

У даному прикладі метод Func має два цілочислових параметри — x, y, а в якості результату метод повертає найбільше з них.

На етапі опису методу (рядок 1) вказуються формальні параметри, на етапі виклику (рядок 2) у метод передаються фактичні параметри, які за кількістю і за типом збігаються з формальними параметрами.

Якщо кількість фактичних і формальних параметрів буде різною, то компілятор видає відповідне повідомлення про помилку. Якщо параметри будуть відрізнятися типами, то компілятор спробує виконати неявне перетворення типів. Якщо неявне перетворення неможливе, то також буде згенеровано помилку.

Під час виклику методу Func використовувалося вкладення одного виклику в інший.

Завдання. Перетворити програму так, щоб за допомогою методу Func можна було знайти найбільше значення з чотирьох чисел: a, b, c, d. Метод Func при цьому не змінювати.

У загальному випадку параметри використовуються для обміну інформацією між методами. В C# для обміну передбачено чотири типи параметрів: параметри-значення, параметри-посилання, вихідні параметри, параметри-масиви.

У ході передачі параметра за значенням метод отримує копії параметрів, і оператори методу працюють із цими копіями. Доступу до початкових значень параметрів у методу немає, а, отже, немає і можливості їх змінити. Всі приклади, розглянуті раніше, використовували передачу даних за значенням.

Приклад 4.

```
using System;
class Program
{
   static void Func(int x)
    {
       x += 10; // Змінили значення параметра
       Console.WriteLine ("In Func:" + x);
     }
   static void Main()
   {
      int a = 10;
      Console.WriteLine("In Main:" + a);
      Func(a);
      Console.WriteLine("In Main:" + a);
   }
}
```

Результат виконання програми:

```
In Main: 10
In Func: 20
In Main: 10
```

У даному прикладі значення формального параметра *x* було змінено в методі Func, але ці зміни не позначилися на фактичному параметрі *a* методу Main.

Під час передачі параметрів за посиланням метод отримує копії адрес параметрів, що дозволяє здійснювати доступ до елементів пам'яті за цими адресами і змінювати вихідні значення параметрів.

Для того щоб параметр передавався по посиланню, необхідно під час опису методу перед формальним параметром і під час виклику методу перед відповідним фактичним параметром поставити службове слово ref.

Приклад 5. Передача параметрів по посиланню за допомогою специфікатора ref.

```
using System;
class Program
{
   static void Func(int x, ref int y)
    {
      x += 10; y += 10; // зміна параметрів
      Console.WriteLine ("In Func: {0}, {1}", x, y);
     }
   static void Main()
   {
      int a = 10, b = 10; // рядок 1
      Console.WriteLine("In Main: {0}, {1}", a, b);
      Func(a, ref b);
      Console.WriteLine("In Main: {0}, {1}", a, b);
   }
}
```

Результат виконання програми:

```
In Main: 10, 10
In Func: 20, 20
In Main: 10, 20
```

У даному прикладі в методі Func були змінені значення формальних параметрів x і y. Ці зміни не позначилися на фактичному параметрі *a,* оскільки він передавався за значенням, але значення *b* було змінено, оскільки він передавався по посиланню.

Передача параметра за посиланням вимагає, щоб аргумент було ініційовано до виклику методу (див. рядок 1). Якщо в цьому рядку не проводити ініціалізацію змінних, то компілятор видає повідомлення про помилку.

Проте в деяких випадках буває неможливо ініціювати параметр до виклику методу. Тоді параметр слід передавати як вихідний, використовуючи специфікатор out.

Приклад 6. Передача параметрів по посиланню за допомогою специфікатора out.

```
using System;
class Program
{
   static void Func(int x, out int y)
    {
      x += 10; y =10; // Визначення значення вихідного параметра y
      Console.WriteLine ("In Func: {0}, {1}", x, y);
     }
   static void Main()
   {
      int a = 10, b;
      Console.WriteLine("In Main: {0}", a);
      Func(a, out b);
      Console.WriteLine("In Main: {0}, {1}", a, b);
   }
}
```

Результат виконання програми:

```
In Main: 10
In Func: 20, 10
In Main: 10, 10
```

У даному прикладі в методі Func формальний параметр *y* і відповідний йому фактичний параметр *b* методу Main були помічені специфікатором out. Тому значення *b* до виклику методу Func можна було не визначати, але зміна параметра *y* образилися на зміні значення параметра *b*.

Приклад 7. Пошук максимального елемента заданого в програмі масиву.

```
using System;
   class Class1
   {
      static int MaxValue(int[] intArray)
      {
```

```
            int maxVal = intArray[0];
            for (int i = 1; i < intArray.Length; i++)
            {
                if (intArray[i] > maxVal)
                    maxVal = intArray[i];
            }
            return maxVal;
        }
        static void Main(string[] args)
        {
            int[] myArray = { 1, 8, 3, 6, 2, 5, 9, 3, 0, 2 };
            int maxVal = MaxValue(myArray);
            Console.WriteLine("Максимальний елемент масиву myArray =
{0}", maxVal);
        }
    }
```

Результат виконання програми:

Максимальний елемент масиву myArray = 9

Розглянутий код містить функцію, що приймає як параметр масив цілих чисел і повертає найбільше з них. Її опис має такий вигляд:

```
    static int MaxValue(int[ ] intArray)
    {
        int maxVal = intArray[0];
        for (int i = 1; i < intArray.Length; i++)
        {
            if (intArray[i] > maxVal)
                maxVal = intArray[i];
        }
        return maxVal;
    }
```

Функція MaxValue ( ) має один параметр, описаний як масив типу int з ім'ям intArray. Значення, що повертається, також має тип int.

Словесний опис алгоритму пошуку максимального значення вихідного масиву має може бути зроблений таким чином.

Локальній цілій змінній з ім'ям maxVal як початкове значення присвоюється перший елемент масиву, а потім здійснюється порівняння цього значення послідовно з усіма іншими елементами.

Якщо поточний елемент більше, ніж значення змінної maxVal, то поточне значення maxVal заміняється на це значення. Коли виконання циклу завершене, змінна maxVal містить найбільше значення даного масиву, що й вертається оператором return.

Код, розташований в Main (), оголошує та ініціалізує простий цілий масив, що буде використовуватися разом із функцією MaxValue( ):

int[ ] myArray = {1, 8, 3, 6, 2, 5, 9, 3, 0, 2};

Під час виклику функції функцією MaxValue( ) значення присвоюється змінній maxVal типу int:

int maxVal = MaxValue(myArray);

Потім це значення виводиться на екран за допомогою оператора

Console.WriteLine("Максимальний елемент масиву myArray = {0}", maxVal);

Завдання. Модифікуйте програму таким чином, щоб програма як результат знаходила середнє значення усіх елементів масиву.

Приклад 8. Обробка одновимірного масиву з використанням функцій.

Приклад обробки одновимірного масиву, що приводиться далі, не має потреби в коментарях. Функціональність програми добре видна з назви відповідних змінних і функцій.

```
using System;
    class Class1
    {
        static void pech_mas(int[ ] m)
        {
            for (int i = 0; i < m.Length;  i++)
                Console.Write("{0} ",m[i]);
            Console.WriteLine(); Console.WriteLine();
        }
        static void sum_otr(int[ ] m)
        {
            double sum_otr = 0.0;
            for(int i=0; i<m.Length; i++)
                if(m[i]<0)
                    sum_otr += m[i];
            Console.WriteLine( "sum_otr = {0}", sum_otr);
        }
```

```
        static void proisv_maxMod_minMod(int[ ] m)
        {
            int proisv_maxMod_minMod = 1;

            int maxMod,minMod;
            maxMod = minMod = Math.Abs(m[0]);

            int index_maxMod, index_minMod;
            index_maxMod = index_minMod = 0;

            for(int i=0; i<m.Length; i++)
            {
                if( Math.Abs(m[i]) < minMod )
                {
                    minMod = Math.Abs(m[i]);
                    index_minMod=i;
                }

                if( Math.Abs(m[i])> maxMod )
                {
                    maxMod = Math.Abs(m[i]);
                    index_maxMod=i;
                }
            }
            for(int i=0; i<m.Length; i++)
                if( i>index_minMod && i<index_maxMod || i<index_minMod && i>index_maxMod)
                    proisv_maxMod_minMod *= m[i];
        Console.WriteLine("index_minMod = {0}, index_maxMod = {1}", index_minMod, index_maxMod);
        Console.WriteLine("minMod = {0}, maxMod = {1}", minMod, maxMod);
        Console.WriteLine("index_minMod = {0}, index_maxMod = {1}", index_minMod, index_maxMod);

        Console.WriteLine("proisv_maxMod_minMod = {0}", proisv_maxMod_minMod);
        }
```

```
        static void Main(string[ ] args)
        {
            int[ ] mas = {-1,5,2,3,5,0,3,9,2,0,1,6,0,-3,2};
            pech_mas(mas);
            sum_otr(mas);
            proisv_maxMod_minMod(mas);
            Console.WriteLine();
        }
    }
```

Результат виконання програми:

-1 5 2 3 5 0 3 9 2 0 1 6 0 -3 2

sum_otr = -4

index_minMod = 5, index_maxMod = 7

minMod = 0, maxMod = 9

index_minMod = 5, index_maxMod = 7

proisv_maxMod_minMod = 3

Завдання. Модифікуйте програму таким чином, щоб вихідний масив вводився з клавіатури.

Порядок виконання лабораторної роботи

Загальна частина

1. Набрати, відкомпілювати і запустити на виконання приклади програм, які були наведені в розділі "Основні положення" даної лабораторної роботи. Звернути увагу на додаткові завдання, які наведені після лістингів програм.

2. Проекспериментувати з програмами:

змінити вихідні дані;

дослідити, як впливають синтаксичні помилки на результат компіляції програми. Які при цьому виникають помилки компіляції?

Індивідуальна частина

1. Написати програму, яка здійснює обробку масиву згідно з усіма пунктами індивідуального завдання. Варіанти алгоритмів обробки взяти з додаткового файла з індивідуальними завданнями.

2. Для кожного з пунктів завдання підготувати чисельні контрольні приклади початкових масивів та результати їх обробки згідно з індивідуальним варіантом.

3. Для кожного з пунктів розробити графічну схему (блок-схему).

4. Відповідно до алгоритму набрати і відкомпілювати тексти програми, усуваючи у разі необхідності помилки.

6. Дослідити роботу програми, аналізуючи виконання контрольних чисельних прикладів.

Зміст звіту

1. Титульний лист.

2. Цілі лабораторного заняття і вказівка, які навички та вміння передбачається отримати в результаті його виконання.

3. Тексти налагоджених програм загальної частини лабораторного заняття з необхідними коментарями і результатом виконання.

4. Постановку задачі індивідуального завдання.

5. Для кожного з пунктів індивідуального завдання надати словесний опис його виконання, який супроводжується чисельним прикладом та графічну схему (блок-схему) відповідного алгоритму.

6. Текст налагодженої програми, яка реалізує всі пункти індивідуального завдання з результатом виконання контрольних чисельних прикладів.

7. Висновки.

Контрольні запитання

1. Дайте визначення функції. Чому функція може слугувати прикладом коду, який повторно виконується?

2. Перерахуйте основні етапи виконання функції.

3. Який синтаксис запису значень, що повертаються з функції?

4. Охарактеризуйте параметри функцій. У чому полягає відповідність параметрів?

5. У чому полягають основні ідеї реалізації процесу обміну інформацією з функцією.

6. Що таке область дії змінних? Охарактеризуйте параметри і значення, що повертаються, за порівнянням із глобальними даними.

7. У чому полягають особливості передачі параметрів за посиланням і за значенням.

10. Які особливості використання функції і масиву?

Рекомендована література

1. Ватсон К. Программист – программисту. C# / К. Ватсон ; пер. с англ. – Изд. "Лори", 2010. – 862 с.

2. Гаврилов В. П. Основи програмування : конспект лекцій для студентів напряму підготовки 0927 "Видавничо-поліграфічна справа" всіх форм навчання / укл. В. П. Гаврилов, В. В. Браткевич, І. О. Бондар. – Х. : Вид. ХНЕУ, 2007. – 172 с.

3. Лабор В. В. Си Шарп. Создание приложений для Windows / В. В. Лабор – Мн. : Харвест, 2011. – 384 с.

4. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах / Ч. Петцольд ; пер. с англ. – М. : Издательско-торговый дом "Русская Редакция", 2009. – 576 с.: ил.

5. Просиз Д. Программирование для Microsoft NET / Д. Просиз ; пер. с англ. – М. : Изд. "Русская Редакция", 2011. – 704 с.

6. Робинсон У. C# без лишних слов / У. Робинсон ; пер. с англ. – М. : ДМК Пресс, 2010. – 352 с.

7. C# для профессионалов. В 2-х томах / С. Робинсон, О. Корнес, Д. Глинн и др. ; пер. с англ. – М. : Изд. "Лори", 2012. – 734 с.

Зміст

НАВЧАЛЬНЕ ВИДАННЯ

Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни "ОСНОВИ ПРОГРАМУВАННЯ"

для студентів напряму підготовки 6.051501 "Видавничо-поліграфічна справа" всіх форм навчання

Укладач **Браткевич** Вячеслав Вячеславович

Відповідальний за випуск *Пушкар О. І.*

Редактор *Бутенко В. О.*

Коректор *Маркова Т. А.*

План 2015 р. Поз. № 77.
Підп. до друку 03.07.2015 р. Формат 60x90 1/16. Папір офсетний. Друк цифровий.
Ум. друк. арк. 7,0. Обл.-вид. арк. 8,75. Тираж 30 пр. Зам. № 85.

Видавець і виготівник – ХНЕУ ім. С. Кузнеця, 61166, м. Харків, просп. Леніна, 9-А

Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру ***ДК № 4853 від 20.02.2015 р.***