Міністерство освіти і науки України
Харківський національний економічний університет
імені Семена Кузнеця

Програмування засобів мультимедіа: методичні рекомендації до виконання лабораторних робіт для студентів спеціальності 186 "Видавництво та поліграфія" першого (бакалаврського) рівня

Укладач Браткевич В. В.

Відповідальний за випуск Пушкар О. І.

Харків. Вид ХНЕУ ім. С. Кузнеця, 2020

УДК 301а БК

Затверджено на засіданні кафедри комп'ютерних систем і технологій.

Протокол № 8 від 11 квітня 2020 р.

Програмування засобів мультимедіа: методичні рекомендації до виконання лабораторних робіт для студентів спеціальності 186 "Видавництво та поліграфія" першого (бакалаврського) рівня / укл. В. В. Браткевич. – Х. : Вид. ХНЕУ ім. С. Кузнеця, 2020. – 63 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 в звіт. В якості середовища розробки обрано Microsoft Visual Studio .NET.

Рекомендовано для студентів спеціальності 186 "Видавництво та поліграфія" першого (бакалаврського) рівня

Вступ

Навчальна дисципліна «Програмування засобів мультимедіа» вивчається студентами другого курсу спеціальності 186 "Видавництво та поліграфія" першого (бакалаврського) рівня. В методичних рекомендаціях розглянуто перші сім лабораторних робіт, що відносяться к змістовим модулям «Організація процедурно-орієнтованих програм» та «Організація і обробка складених типів даних» розділу «Основи програмування», який вивчається протягом третього семестру.

Метою викладання навчальної дисципліни «Програмування засобів мультимедіа» 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) серед розробки мультимедіа і Web-дизайну.

Основними завданнями вивчення дисципліни «Програмування засобів мультимедіа» 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 та 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 процедуро орієнтованих 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#.

Мови програмування та компілятори.

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.

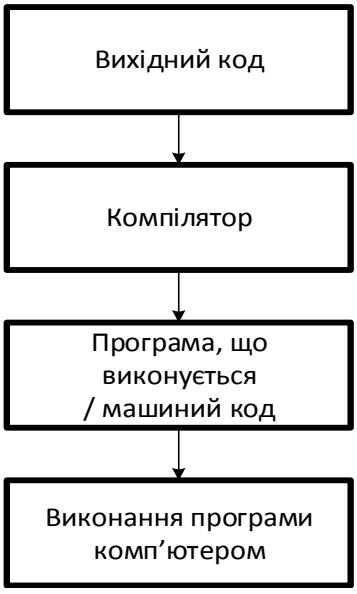


Рис. 1. **Традиційний процес компіляції**

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.

Елементи інтерфейсу Visual Studio .NET.

Основні елементи інтерфейсу містять величезний набір інструментів, покликаних спростити життя програмісту. У даній роботі зроблено огляд деяких можливостей середовища Visual Studio .NET.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. 2.

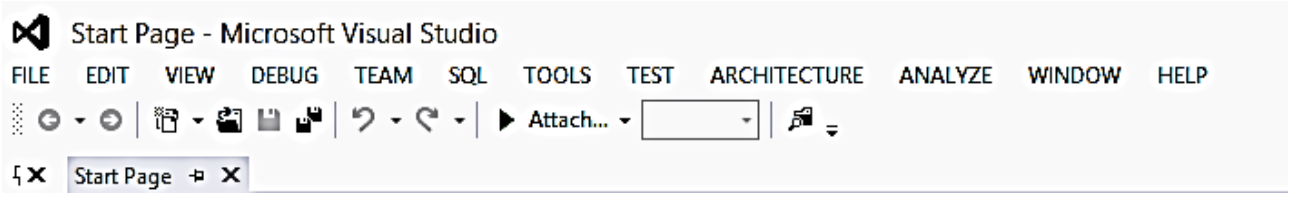


Рис. 2. **Фрагмент типової стартової сторінка Visual Studio .NET**

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. 3.

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.

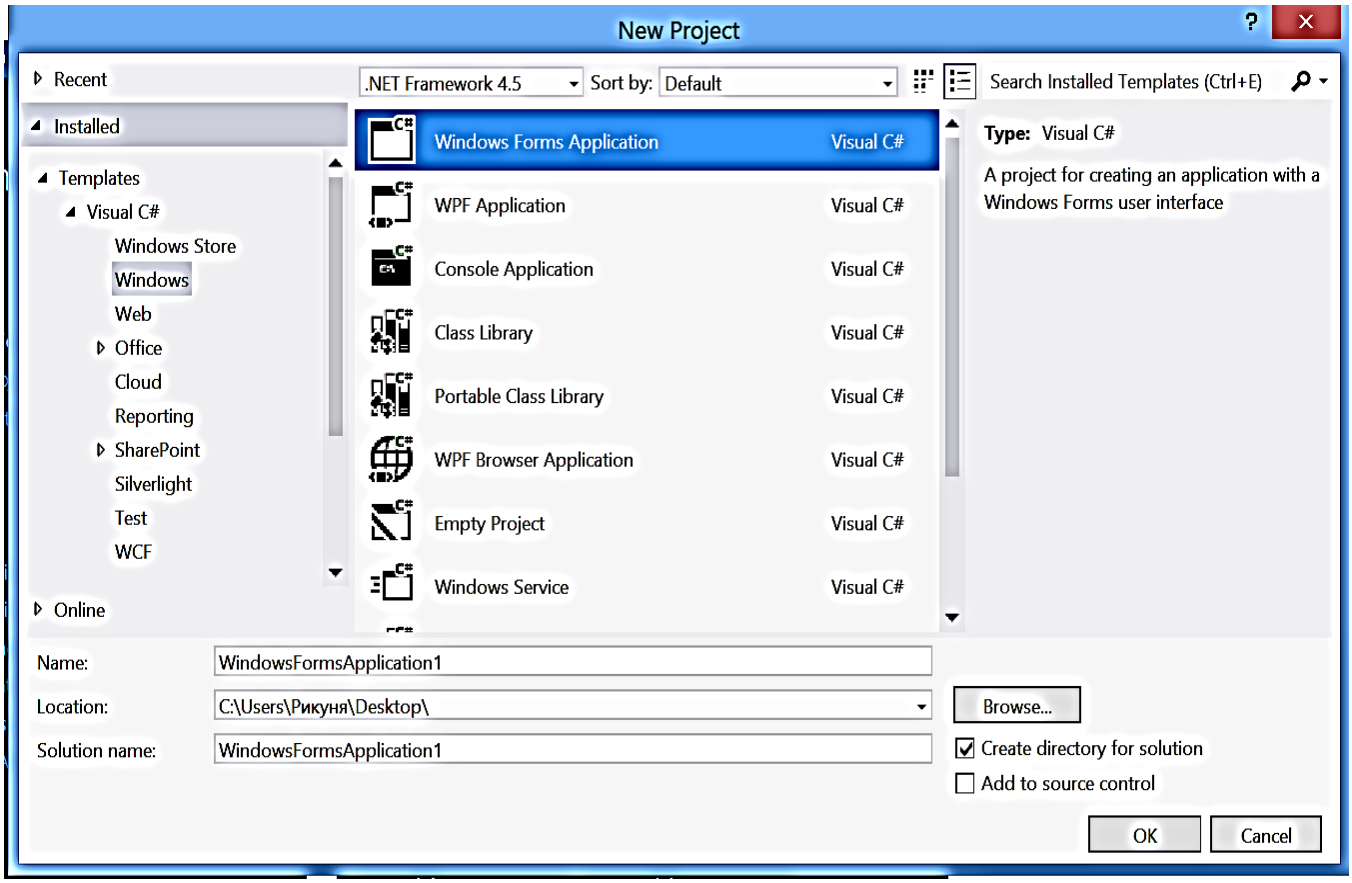


Рис. 3. **Вікно вибору типу проекту**

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 візуальної середовища розробки проекту. Вони зображені на рис. 4.

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 дозволяє управляти компонентами, включеними в проект.

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.

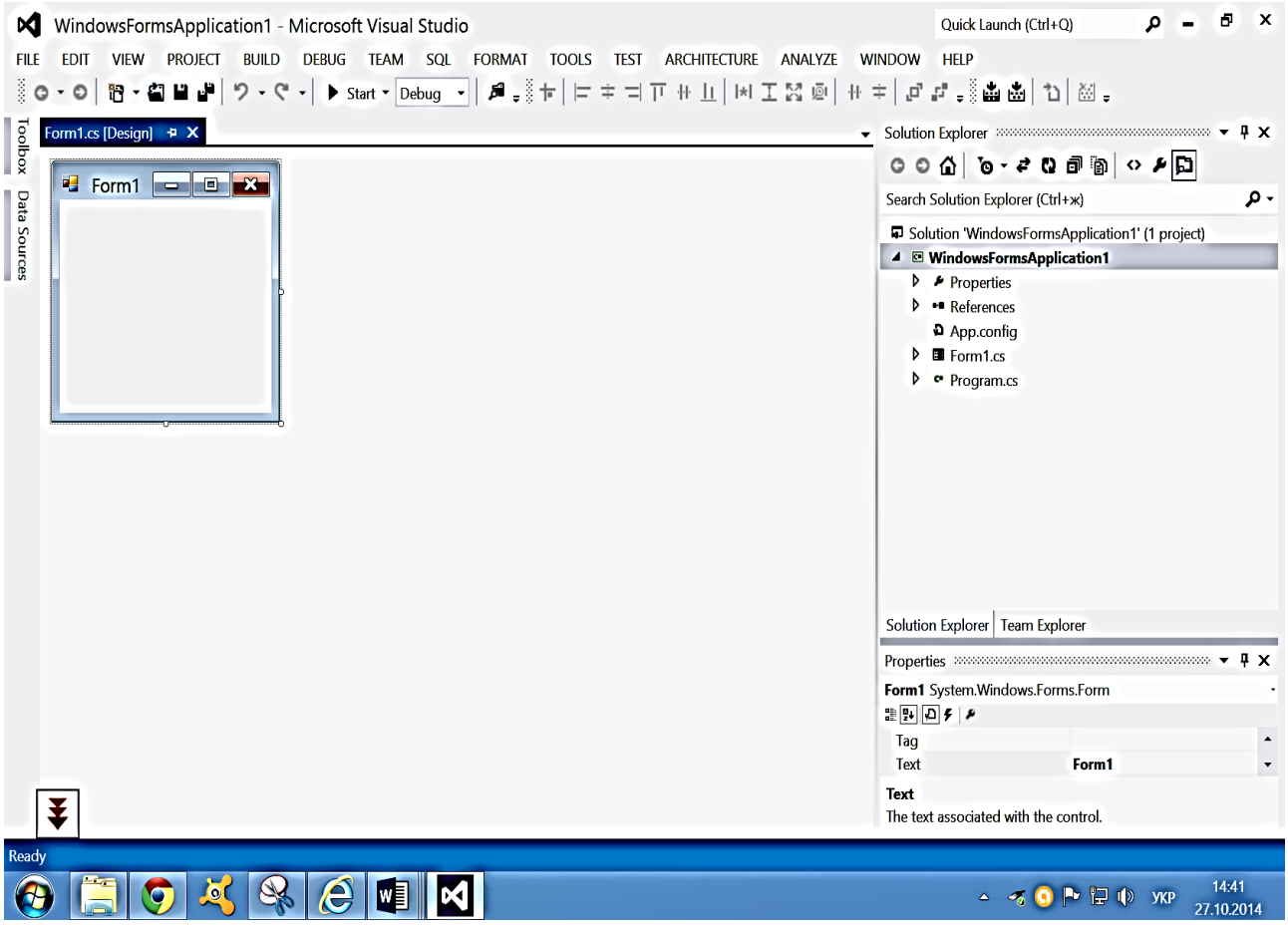


Рис. 4. **Основні частини візуальної середовища розробки**

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

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.

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.

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.

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 під 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.

Порядок виконання лабораторної роботи

Створення простого консольного додатку.

Буде використано 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. 5) настройки: Visual C# / Windpws; тип проекту Console Application.

3. У поле 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. 5).

4. Клацніть по кнопці ОК.

5. Після того, як шаблон проекту буде створено, додати в файл, виведений в основному вікні (рис. 6), 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. 7).

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.

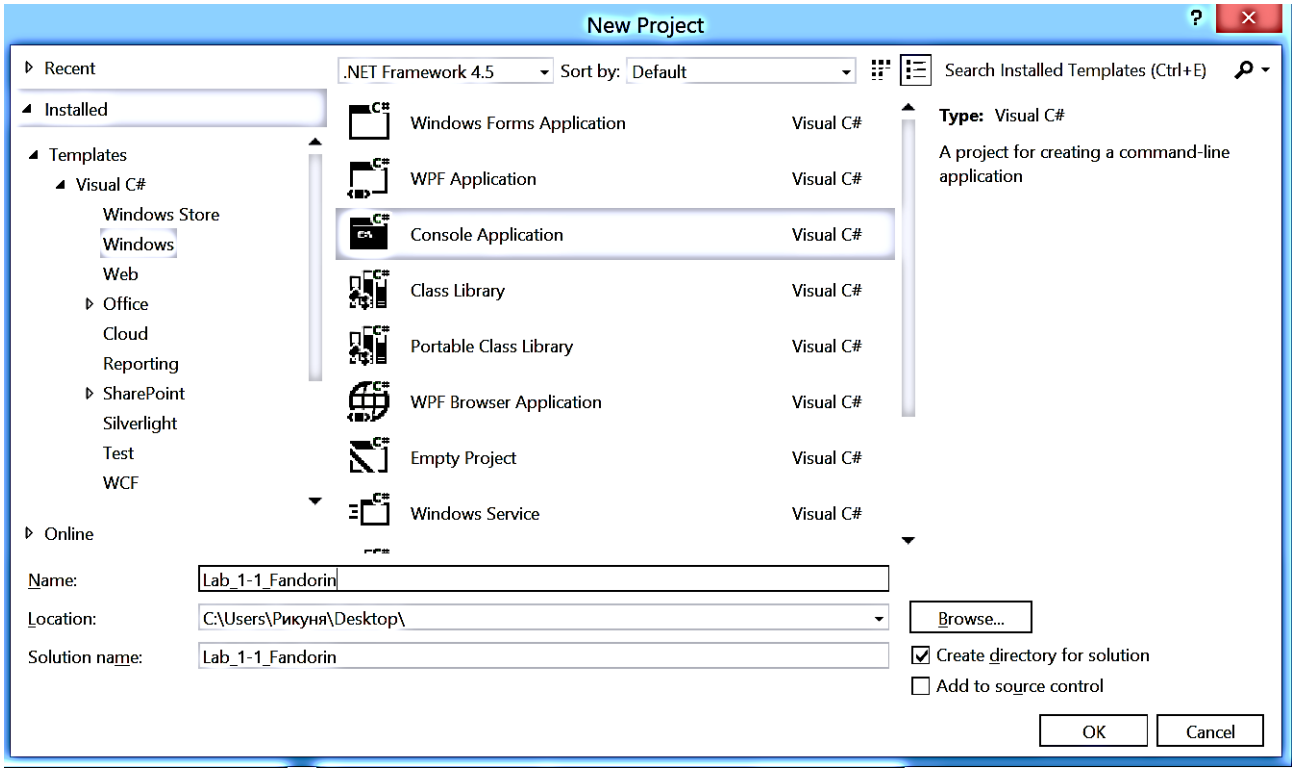


Рис. 5. **Вікно New Project для створення простого консольного додатка**

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. 8.

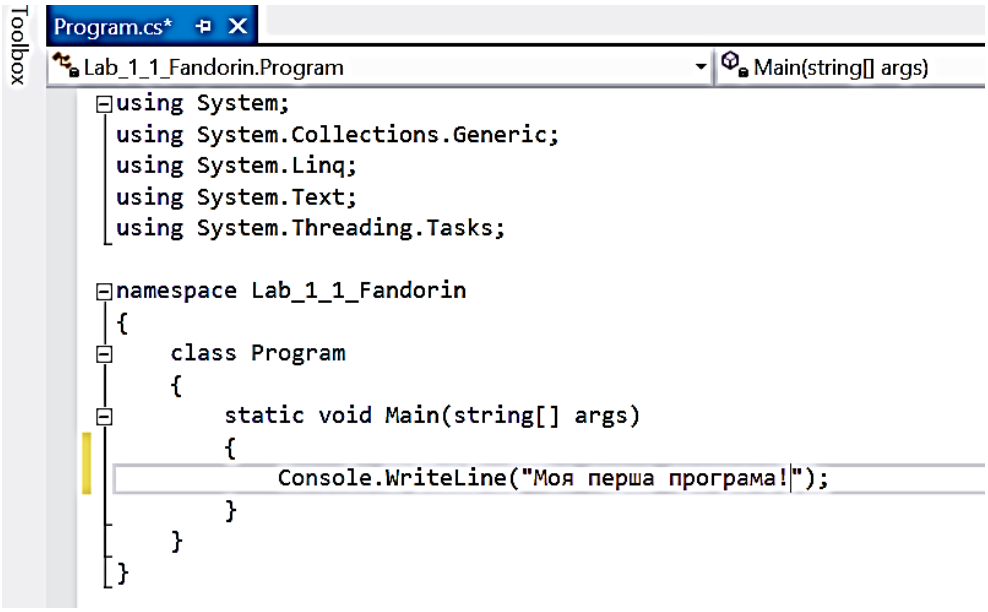


Рис. 6. **Перша консольна програма**

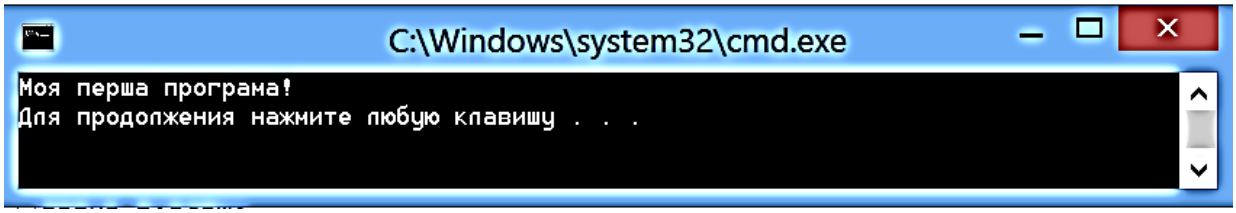


Рис. 7. **Результат виконання першої консольної програми**

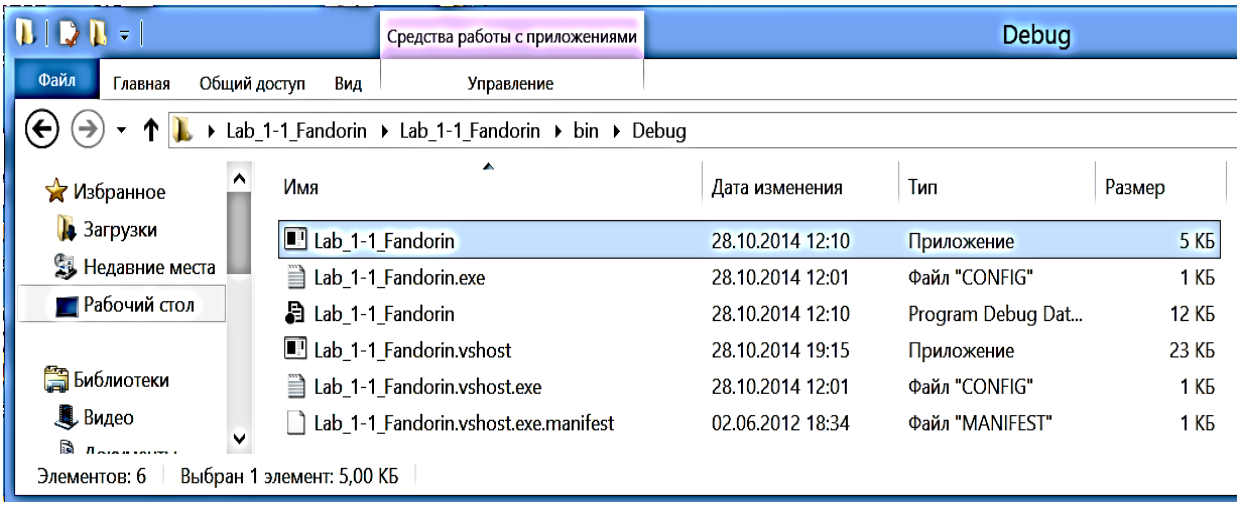


Рис. 8**. Директорія з файлами, які отримано в результаті компіляції проекту**

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 з віконним середовищем.

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

Лабораторна робота № 2

Програмування лінійних обчислювальних процесів

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

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

знання:

класифікацій 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 в 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 да-

ній 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. Українськи 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# наведено у вигляді програми (рис. 9) та результата її виконання (рис. 10).

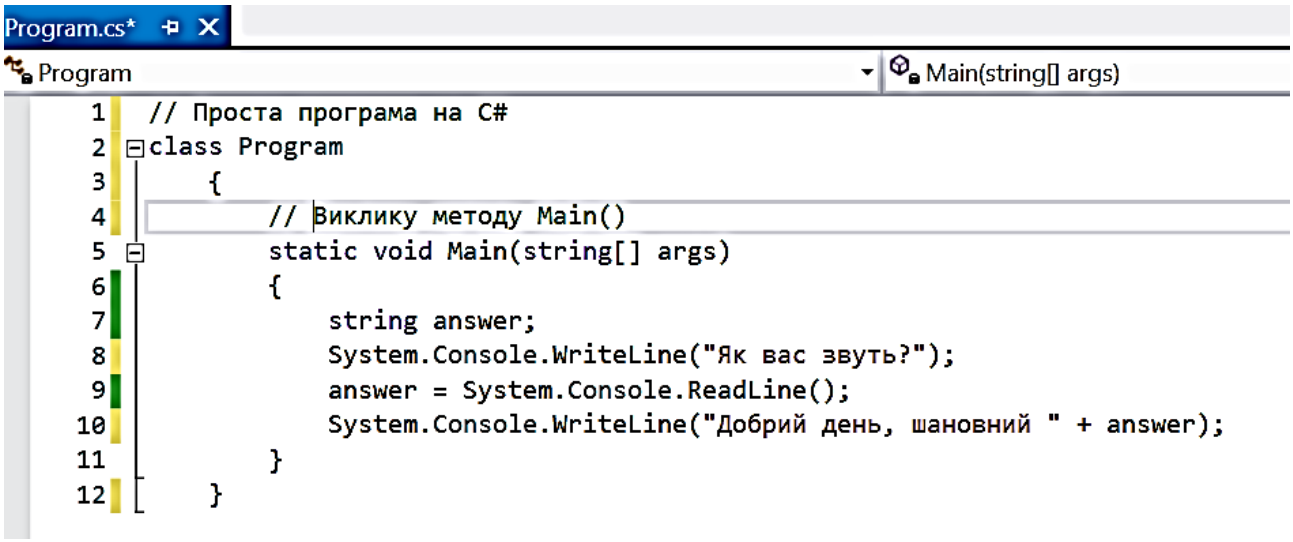


Рис. 9. **Базова структура C# програми**

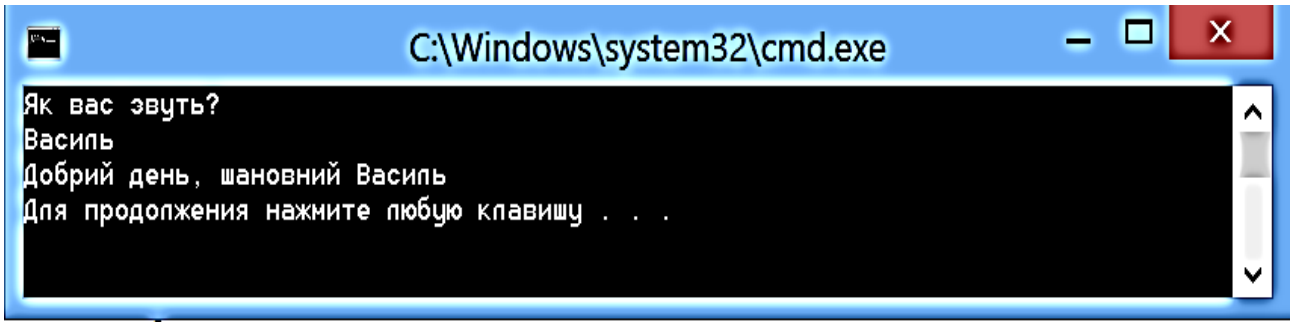


Рис. 10. **Результат виконання базової структури програми**

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.

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.

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.

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. 1.

Порядок виконання лабораторної роботи

Загальна частина.

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?

Таблиця 1

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

Продовження табл. 3

1	2	3	4
Адитив-ні	< Вираз1> + < Вираз2> << Вираз1> - < Вираз2>	Зліва направо Зліва направо	Додавання Віднімання
Відно-шення	< Вираз1> < < Вираз2> < Вираз1> <= < Вираз2> < Вираз1> > < Вираз2> < Вираз1> >= < Вираз2> <Об'єкт> is <Тип>	Зліва направо Зліва направо Зліва направо Зліва направо Зліва направо	Менше Менше або дорівнює Більше Більше або дорівнює Приналежність типу
Рівність	< Вираз1> == < Вираз2> < Вираз1> != < Вираз2>	Зліва направо Зліва направо	Дорівнює Не дорівнює
Логічні побітові	< Логічний_вираз1> & < Логічний_вираз2> < Логічний_вираз1> ^ < Логічний_вираз2> < Логічний_вираз1> | < Логічний_вираз2>	Зліва направо Зліва направо Зліва направо	Побітове І Побітове АБО, що ви-ключає Побітове АБО
Логічні	< Логічний_вираз1> && < Логічний_вираз2> < Логічний_вираз1> || < Логічний_вираз2> < Логічний_вираз> ? < Вираз1> : < Вираз2>	Зліва направо Зліва направо Зліва направо	Логічне І Логічне АБО Умовна операція

Закінчення табл. 3

1	2	3	4
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

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.

Графічна схема оператора наведена на рис. 11.

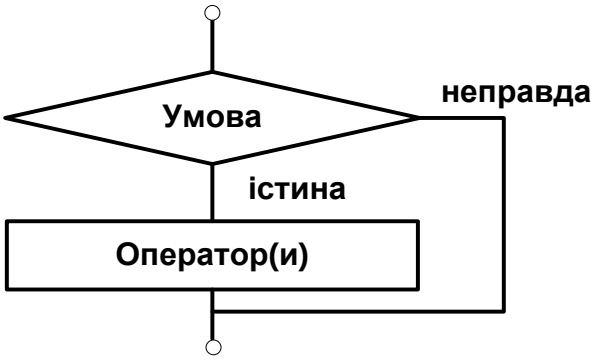


Рис. 11. **Графічна схема оператора if**

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.

Структура вибору if / else.

Графічна схема оператора наведена на рис. 12.

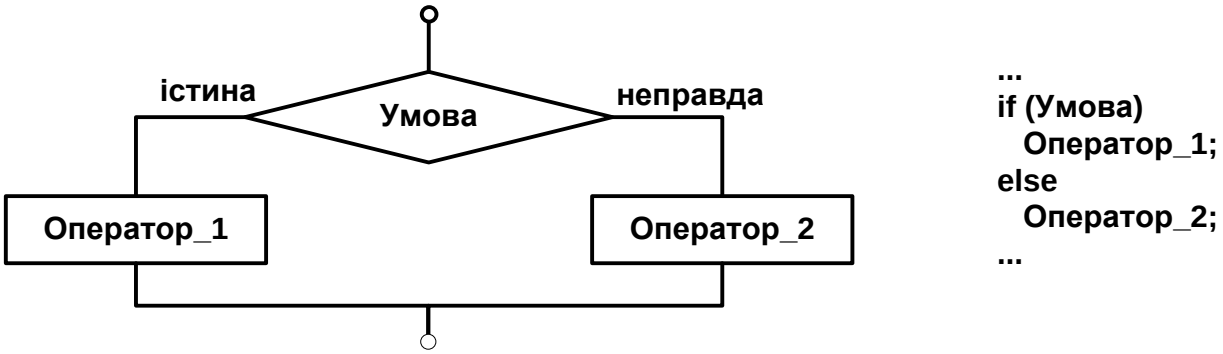


Рис. 12. **Графічна схема оператора if / else**

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.

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. Графічна схема оператора switch наведена на рис. 13.

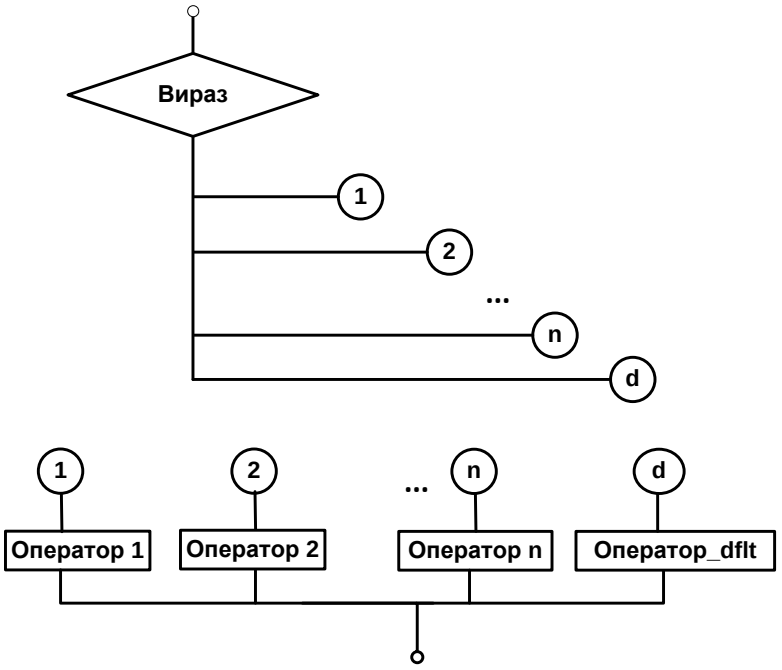


Рис. 13. **Графічна схема оператора switch**

Може бути один або не бути жодного блоку default.

Приклад 3. 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 вибрати чи-
сло " +
                    "між 2 and 5");
                    break;
        }
    }
    }
```

Умовний вираз.

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

<Логічний_вираз_1> ? <Вираз_2> : <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;

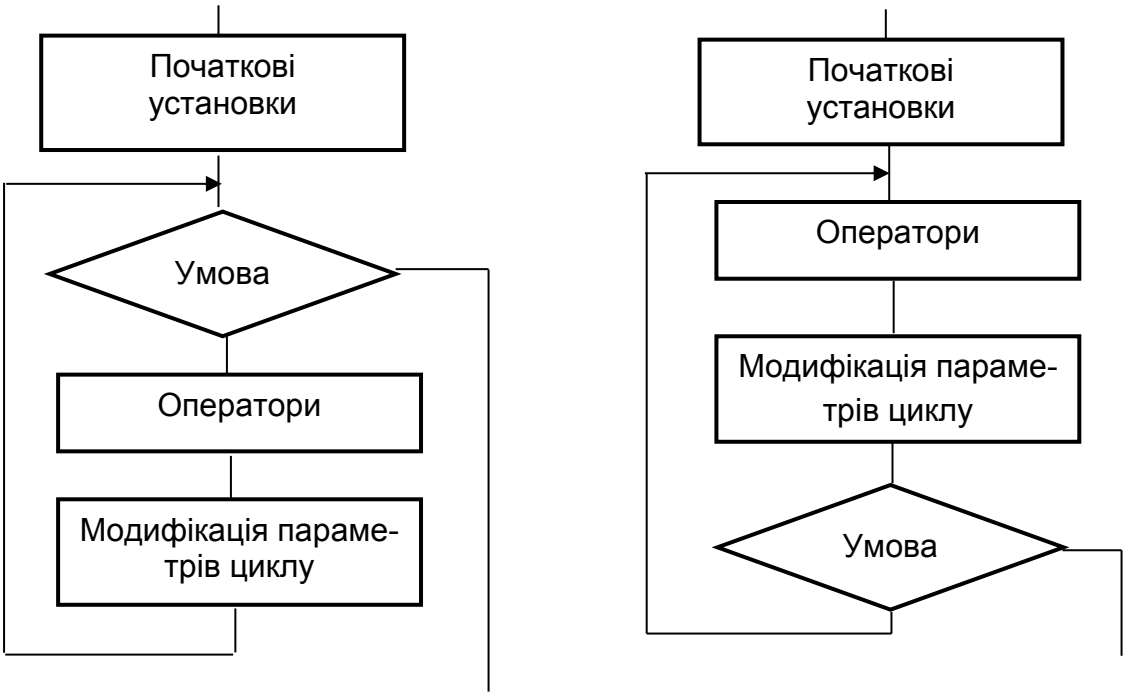
модифікації параметра циклу;

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. 14.

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.



а) цикл із передумовою б) цикл із постумовою

Рис. 14. **Графічні схеми, що демонструють роботу циклів**

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. 14 а.

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.

Цикл із постумовою (do / while).

Цикл із постумовою реалізує графічну схему (без блоку початкових установок), наведену на рис. 14 б.

Приклад 2. 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.

Приклад 3. 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 4. 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.

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 і оброблення кладених типів 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.

Оголошення масиву й присвоювання йому значення:

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 в залежності 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};
```

Приклад 3. 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

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.

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).

Приклад 4. 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 в бригаді...");
                kol_rab=Convert.ToInt32(Console.ReadLine());
                Console.WriteLine("Введіть кількість бригад...");
                kol_brigad=Convert.ToInt32(Console.ReadLine());
                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} бри-
гади", 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 брига-
дах:\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!" );
        }
}
```

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 файлу 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.

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 (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

Приклад 2. 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");
        Console.WriteLine("|-----------------------------------------------------------|");
        Console.WriteLine("|  n/n |  Автор  |  Вартість  | Видано |  Витрати  |");
        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("|-----------------------------------------------------------|");
        }
}
```

Один з можливих результатів роботи програми:

```
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 розділу, який присвячено об’єктно-орієнтованому програмуванняю.

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 іншій.

```
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. 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 питання

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# / Ватсон; – пер. с англ. Издательство «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. : — Издательство «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 томах / С. Робинсон, О. Корнес, Д. Глинн и др. ; пер. с англ. – М. : Изд. «Лори», 2012. – 734 с.

ЗМІСТ

НАВЧАЛЬНЕ ВИДАННЯ

Програмування засобів мультимедіа: методичні рекомендації до виконання лабораторних робіт для студентів спеціальності 186 "Видавництво та поліграфія" першого (бакалаврського) рівня

Укладач Браткевич В. В.

Відповідальний за випуск Пушкар О. І.

Редактор

Коректор

План 2020 р. Поз. №

Підп. до друку Формат 60 × 90 1/16. Бумага TATRA. Друк офсетний.

Ум.-друк. арк. Обл.-вид. арк. Тираж экз. Зам. № Безкошто-вно.

Свідоцтво про внесення до Державного реєстру суб'єктів видавничої справи ***Дк № 481 від 06.2001***

Видавець і виготівник – видавництво ХНЕУ ім. С. Кузнеця, 61001, м. Харків, пр. Науки, 9а