**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»**

**Факультет електроенерготехніки та автоматики**

# МЕТОДИЧНІ ВКАЗІВКИ ТА ЗАВДАННЯ
**до виконання практичних робіт з дисципліни**

# «ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ. ЧАСТИНА 1»
**для студентів спеціальності**
**141 Електроенергетика, електротехніка та електромеханіка**

**(навчальне електронне видання)**

**НТУУ “КПІ”**
**2016**

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»**

**Факультет електроенерготехніки та автоматики**

# МЕТОДИЧНІ ВКАЗІВКИ ТА ЗАВДАННЯ
**до виконання практичних робіт з дисципліни**

# «ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ. ЧАСТИНА 1»
**для студентів спеціальності**
**141 Електроенергетика, електротехніка та електромеханіка**

**(навчальне електронне видання)**

*Рекомендовано Вченою радою*
*факультету електроенерготехніки та автоматики*

**НТУУ “КПІ”**
**2016**

Методичні вказівки виконання практичних робіт з дисципліни «Обчислювальна техніка та програмування. Частина 1» для студентів спеціальності 141 Електроенергетика, електротехніка та електромеханіка / Уклад.: Д.В. Настенко, А.Б. Нестерко, Г.О. Труніна – Київ: НТУУ “КПІ”, 2016.

*Гриф надано Вченою радою ФЕА НТУУ “КПІ”*
*(Протокол № 10 від 30 червня 2016 р.)*

Навчальне електронне видання

# МЕТОДИЧНІ ВКАЗІВКИ ТА ЗАВДАННЯ
до виконання практичних робіт з дисципліни

# «ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ. ЧАСТИНА 1»
141 Електроенергетика, електротехніка та електромеханіка

Укладачі: Настенко Дмитро Васильович, ст. викл.,
Нестерко Артем Борисович, к.т.н., ст. викл.
Труніна Ганна Олексіївна, асистент.

Відповідальний
Редактор О.С. Яндульський, професор, д.т.н.

Рецензент Т.Л. Кацадзе, канд. техн. наук

*За редакцією укладачів*

# Зміст

Вступ ..... 7
Практичне заняття №1. Розробка елементарної програми мовою програмування C# ..... 8
Практичне заняття №2. Програмування арифметичних виразів ..... 14
Практичне заняття №3. Робота з текстовими рядками ..... 19
Практичне заняття №4. Оператор розгалуження if/else ..... 24
Практичне заняття №5. Оператор множинного вибору switch/case ..... 26
Практичне заняття №6. Оператори циклу. Частина 1. Цикл for ..... 29
Практичне заняття №7. Оператори циклу. Частина 2. Цикли while і do/while ..... 32
Практичне заняття №8. Одновимірні масиви ..... 39
Практичне заняття №9. Двомірні масиви. Основи роботи з матрицями ..... 46
Список рекомендованої літератури ..... 51

## Вступ

Microsoft .NET — програмна технологія компанії Microsoft, яка є платформою для створення як звичайних додатків, так і веб- додатків. .NET-додатки можуть розроблятися і виконуватися як в середовищі операційних систем Microsoft, так і в інших включаючи Mac OS X, різні дистрибутиви Linux, Solaris, а також на мобільних пристроях iOS і Android (через API-інтерфейс MonoTouch).

C# – це об'єктно-орієнтована мова програмування, розроблена в 1998-2001 роках групою інженерів під керівництвом Андерса Хейлсберга в компанії Microsoft як мова розробки додатків для платформи Microsoft .NET. Синтаксис C# близький до C++ і Java. На сьогодні C# є флагманською мовою корпорації Microsoft для програмування в середовищі Windows.

Основним інтегрованим середовищем розробки, тобто комп'ютерною програмою, що допомагає програмістові розробляти нове програмне забезпечення, для C# є Visual Studio. Альтернативними середовищами програмування є SharpDevelop та MonoDevelop.

C # – не єдина мова, яка може використовуватися для побудови .NET-додатків. Середовище Visual Studio дає змогу використовувати п'ять мов, а саме – C #, Visual Basic, C ++/CLI, JavaScript і F#. Існують .NET-компілятори які призначені для мов Ruby, Python, Pascal тощо.

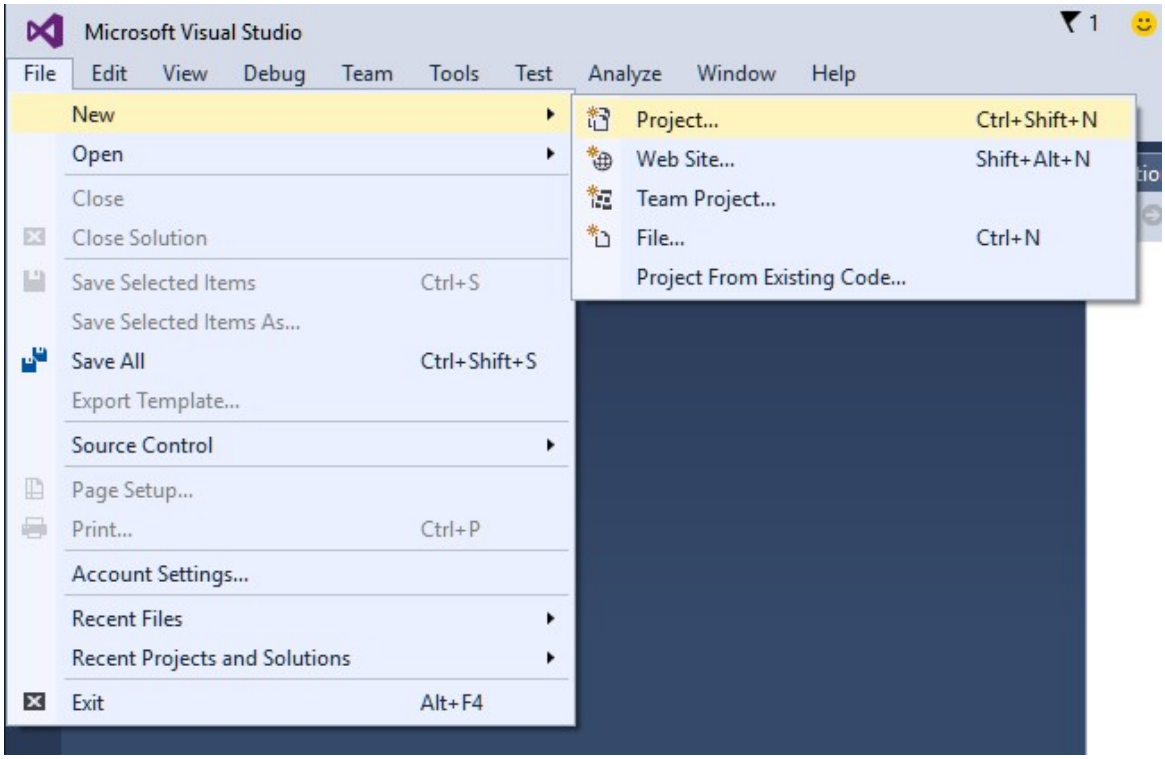
Visual Studio працює на платформі Windows і орієнтована на створення Windows- і веб-додатків, однак є можливість роботи і з консольними додатками. При запуску консольного додатка операційна система створює так зване консольне вікно, через яке відбувається взаємодія з програмою. Консольні додатки найкраще підходять для вивчення мови, тому що в них використовується мінімум стандартних об'єктів. У першій частині курсу ми будемо створювати тільки консольні додатки, щоб зосередити увагу на базових властивостях мови C#.

## Практичне заняття №1. Розробка елементарної програми мовою програмування C#

Робота з вихідним кодом та іншими об’єктами програми у Visual Studio відбувається в контексті рішення. Рішення – це віртуальний контейнер найвищого рівня для всіх елементів розробки. Рішення може містити один або кілька проектів, а також файли різноманітних типів (текстові документи, зображення тощо), але не може містити всередині себе інші рішення.

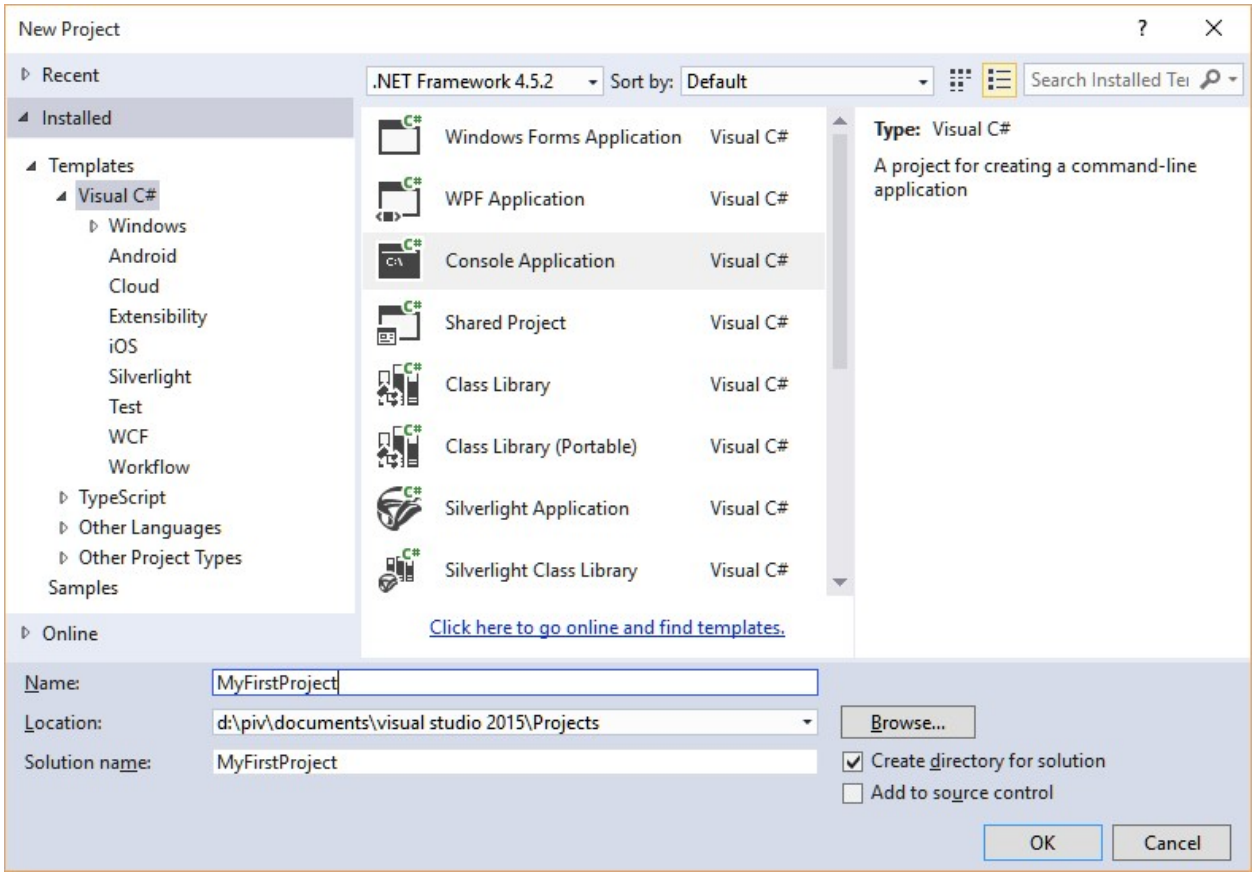
Проект – це контейнер більш низького рівня, який завжди знаходиться всередині якогось рішення і використовується в якості організаційної одиниці для розміщення і групування файлів програми.

Для створення проекту слід після запуску Visual Studio в головному меню вибрати команду File > New > Project…




Створення проекту за допомогою меню File

У лівій частині діалогового вікна потрібно вибрати пункт Visual C#, в правій – пункт Console Application. У полі Name можна ввести ім'я проекту, а в полі Location – місце його збереження на диску.




Вибір типу проекту

У верхній частині екрана розташовується головне меню та панелі інструментів.

У верхній лівій частині екрана розташовується вікно управління проектом Solution Explorer (якщо воно не відображається, слід скористатися командою View > Solution Explorer головного меню). У вікні перераховані всі ресурси, що входять в проект. У нижній лівій частині екрана розташоване вікно властивостей Properties (якщо вікна не видно, скористайтеся командою View > Properties головного меню). У вікні властивостей відображаються найважливіші характеристики виділеного елемента.

Основний простір екрана займає вікно редактора, в якому розташовується текст програми, створений середовищем автоматично. Текст являє собою каркас, в який програміст додає код.

Для початку створимо новий проект консольного додатку з іменем MyFirstProject. Автоматично згенерований код програми має такий вигляд:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MyFirstProject
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Перші рядки містять директиву using, яка повідомляє компілятору, де він повинен шукати класи (типи), не визначені в даному просторі імен. За замовчуванням вказано стандартний простір імен System, де визначена основна частина типів середовища .NET.

Наступний рядок

```
namespace MyFirstProject
```

визначає простір імен додатку. За замовчаннями як ім'я простору імен вибирається ім'я проекту. Область дії простору імен визначається блоком коду між фігурними дужками. Простір імен забезпечує спосіб відокремлення одного набору імен від іншого. Імена, оголошені в одному просторі імен, не конфліктують з іменами, оголошеними в іншому просторі імен.

Слово class, розташоване в першому рядку тексту першої програми, відноситься до об'єктно орієнтованої частини мови, і буде детально розглянуто пізніше. Слово class повинне бути присутнім у будь-якій програмі на C# хоча б один раз.

Фраза static void Main () є заголовком методу Main. Наступний блок фігурних дужок обмежує тіло методу Main. Ім'я методу Main не може бути змінено, оскільки система саме з цієї підпрограми починає виконання додатку (так звана точка входу).

Додамо в тіло методу Main два рядки:

```
static void Main(string[] args)
        {
            Console.WriteLine("Hello world!");
            Console.ReadLine();
        }
```

Рядок

```
Console.WriteLine("Hello world!");
```

виводить повідомлення на екран. Оператор закінчується символом «;», який є обов'язковим для завершення більшості конструкцій C#.

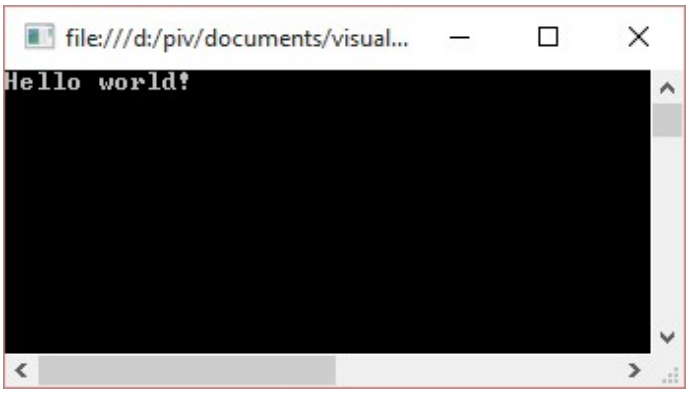
Рядок

```
Console.ReadLine();
```

вказує програмі очікувати введення даних з клавіатури та натиснення клавіші Enter.

Скомпілюємо і запустимо додаток на виконання. Для цього необхідно вибрати пункт меню Debug > Start Debugging, натиснути кнопку 

на панелі інструментів або натиснути клавішу F5. В результаті виконання програми з'явиться вікно:




Результат виконання програми

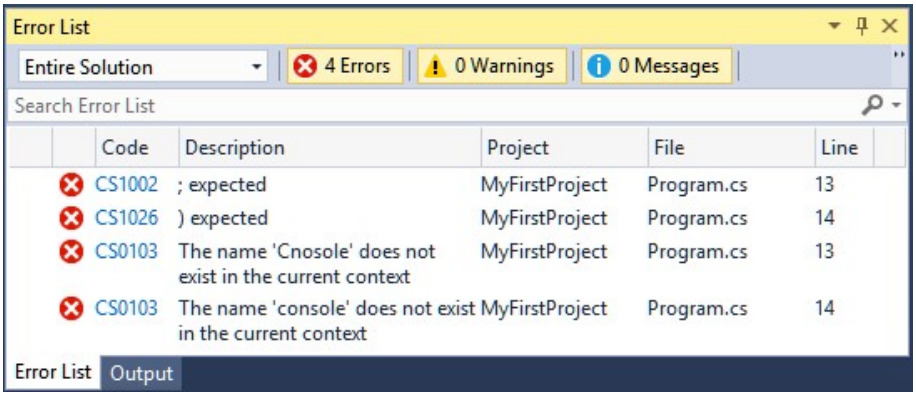
Часто в програмах містяться помилки. Деякі з цих помилок автоматично виявляються засобами Visual Studio. Після виявлення помилки процес компіляції і запуску програми переривається і видається відповідне діагностичне повідомлення для користувача.

Розглянемо автоматичне виявлення помилок на окремому прикладі. Для цього створимо консольний додаток раніше описаним способом. У вікні редактора коду наберіть такий текст програми

```
static void Main(string[] args)
        {
            Cnosole.WriteLine("Hello world!")
            console.LineRead(;
        }
```

Спробуйте скомпілювати і запустити додаток виконання. Після цього у вікні Error List повинні з'явитися повідомлення про виявлені помилки.

У відкритому вікні Error List двічі клацніть на знайденій помилці і переконайтеся, що при цьому в редакторі коду курсор переміститься на рядок з помилкою. Повідомлення про помилку містить номер помилки і короткий її опис.




Вікно зі списком помилок

Перегляньте весь список виявлених компілятором помилок і ознайомтеся з їх описом. Використовуючи отриману інформацію, виправте усі помилки. Якщо всі синтаксичні помилки були виправлені, то компіляція пройде успішно і програма запуститься.

Важливою частиною будь-якої мови програмування є коментарі, які дозволяють зручно описувати та пояснювати різні ділянки коду. У C# використовуються традиційні коментарі в стилі C – однорядкові (// ...) та багаторядкові (/ *... * /). Все, що знаходиться в коментарі (для однорядкового

– від // до кінця рядка; для багаторядкового – весь текст розташований між /* та */). ігнорується компілятором:

```
static void Main(string[] args)
        {
            //Вивід повідомлення на екран
            Console.WriteLine("Hello world!");
            //Очікування вводу
            Console.ReadLine();
        }
```

Розглянемо інші функції головного меню середовища Visual Studio.

## Практичне заняття №2. Програмування арифметичних виразів

Вираз – це правило обчислення значення. Вираз складають операнди, об'єднані знаками операцій. Операндами найпростішого вирази можуть бути константи, змінні і виклики функцій.

За кількістю операндів що беруть участь в одній операції, операції діляться на унарні, бінарні і тернарний.

### Оператори мови C#.

| Категорія | Знак операції | Назва |
|---|---|---|
| Первинні | `x()` | Виклик методу або делегата |
| | `x[]` | Доступ до елементу |
| | `X++` | Постфіксний інкремент |
| | `X--` | Постфіксний декремент |
| | `new` | Виділення пам'яті |
| | `typeof` | Отримання типу |
| Унарні | `+` | Унарний плюс |
| | `-` | Унарний мінус (арифметичне заперечення) |
| | `!` | Логічне заперечення |
| | `~` | Порозрядне заперечення |
| | `++X` | Префіксний інкремент |
| | `--X` | Префіксний декремент |
| | `(тип)x` | Перетворення типу |
| Мультиплікативні | `*` | Множення |
| | `/` | Ділення |
| | `%` | Залишок від ділення |
| Адитивні | `+` | Додавання |
| | `-` | Віднімання |
| Зсуву | `<<` | Зсув вліво |
| | `>>` | Зсув вправо |
| Відношення та перевірки типу | `<` | Менше |
| | `>` | Більше |
| | `<=` | Менше або дорівнює |
| | `>=` | Більше або дорівнює |
| | `is` | Перевірка приналежності типом |
| | `as` | Приведення типу |

| **Категорія** | **Знак операції** | **Назва** |
|---|---|---|
| Перевірки на рівність | `==` | дорівнює |
| | `!=` | Не дорівнює |
| Порозрядні логічні | `&` | Поразрядна кон'юнкція (І) |
| | `^` | Порозрядне виключаюче АБО |
| | `\|` | Поразрядна диз'юнкція (АБО) |
| Умовні логічні | `&&` | Логічне І |
| | `\|\|` | Логічне АБО |
| Умовний | `?` | Умовна операція |
| Присвоєння | `=` | Присвоєння |
| | `*=` | Множення з присвоєнням |
| | `/=` | Ділення з присвоєнням |
| | `%=` | Залишок від ділення з присвоєнням |
| | `+=` | Додавання з присвоєнням |
| | `-=` | Віднімання з присвоєнням |
| | `<<=` | Зсув вліво з присвоєнням |
| | `>>=` | Зсув вправо з присвоєнням |
| | `&=` | Порозрядне І з присвоєнням |
| | `^=` | Порозрядне виключає АБО з присвоєнням |
| | `\|=` | Порозрядне АБО з присвоєнням |

Операції у виразі виконуються в певному порядку відповідно до пріоритетів, як і в математиці. В таблиці операції розташовані за спаданням пріоритетів, рівні пріоритетів розділені в таблиці горизонтальними лініями.

Для зміни порядку виконання операцій використовуються круглі дужки, рівень їх вкладеності практично не обмежений.

При обчисленні виразів може виникнути необхідність у перетворенні типів. Якщо операнди, що входять у вираз, одного типу і операція для цього типу визначена, то результат виразу буде мати той же тип.

Якщо операнди різного типу або операція для цього типу не визначена, перед обчисленнями автоматично виконується перетворення типу за правилами, що забезпечує приведення коротших типів до більш довгих для збереження значущості і точності.

Якщо неявного перетворення з одного типу в інший не існує, програміст може задати явне перетворення типу за допомогою операції (тип) x.

Операції інкремента (++) і декремента, що називають також операціями збільшення та зменшення на одиницю, мають дві форми запису – префіксну, коли знак операції записується перед операндом, і постфіксну.

У префіксній формі спочатку змінюється операнд, а потім його значення стає результуючим значенням виразу, а в постфіксній формі значенням виразу є початкове значення операнда, після чого він змінюється.

Арифметичне заперечення (унарний мінус -) змінює знак операнда на протилежний. Стандартна операція заперечення визначена для типів int, long, float, double і decimal.

Логічне заперечення (!) Визначено для типу bool. Результат операції – значення false, якщо операнд дорівнює true, і значення true, якщо операнд дорівнює false.

Порозрядне заперечення (~) інвертує кожен розряд в двійковому представленні операнда типу int, uint, long або ulong.

Операції зсуву (<< та >>) застосовуються до цілочисельних операндів. Вони зсувають двійкове представлення першого операнда вліво або вправо на кількість двійкових розрядів, задане другим операндом. При зсуві вліво (<<) звільнені розряди обнуляються. При зсуві вправо (>>) звільнені біти заповнюються нулями, якщо перший операнд беззнакового типу (тобто виконується логічний зсув), і знаковим розрядом – в іншому випадку (виконується арифметичний зсув).

Операції відношення (<, <-,>,> -, -,! -) порівнюють перший операнд з другим. Операнди повинні бути арифметичного типу. Результат операції – логічного типу.

Порозрядні логічні операції (&, |, ^) застосовуються до цілочисельних операндів і працюють з їх двійковими представленням. При виконанні операцій операнди зіставляються побітно (перший біт першого операнда з

першим бітом другого, другий біт першого операнда з другим бітом другого і т. Д.). Стандартні операції визначені для типів int, uint, long і ulong.

При поразрядной кон'юнкції, або порозрядному І (операція позначається &), біт результату дорівнює 1 тільки тоді, коли відповідні біти обох операндів рівні 1. При поразрядной диз'юнкції, або порозрядному АБО (операція позначається |), біт результату дорівнює 1 тоді, коли відповідний біт хоча б одного з операндів дорівнює 1.

При порозрядному виключає АБО (операція позначається ~) біт результату дорівнює 1 тільки тоді коли відповідний біт тільки одного з операндів дорівнює 1.

Умовні логічні операції І (&&) і АБО (||) найчастіше використовуються з операндами логічного типу. Результатом логічної операції є true або false.

Результат операції логічне І має значення true, тільки якщо обидва операнда мають значення true. Результат операції логічне АБО має значення true, якщо хоча б один з операндів має значення true.

Операції присвоювання (=, + =, -=, *= і т. д.) задають нове значення змінної.

Механізм виконання операції присвоювання такий: обчислюється вираз і його результат заноситься в пам'ять за адресою, що визначається ім'ям змінної, що знаходиться зліва від знака операції. Значення, що раніше зберігалося в цій області пам'яті, втрачається.

У складних операціях присвоєння (+ =, -=, *= і т. д.) при обчисленні виразу, що стоїть в правій частині, використовується значення з лівої частини. Наприклад, при додаванні з присвоюванням до другого операнд додається перший, і результат записується в перший операнд, тобто вираз

*a + = 5;*

є більш компактною формою запису виразу

*a += a + 5;*

У виразах часто використовуються математичні функції, наприклад синус або піднесення до степені. Вони реалізовані в класі Math, визначеному в просторі імен System.

**Основні поля і методи класу Math**

| Ім'я | Опис | Пояснення |
|---|---|---|
| **Abs** | Модуль | \| x \| записується як **Abs (x)** |
| **Acos** | Арккосинус | **Acos (double x)** |
| **Asin** | Арксинус | **Asin (double x)** |
| **Atan** | Арктангенс | **Atan (double x)** |
| **Atan2** | Арктангенс | **Atan2 (double x, double y)** – кут, тангенс якого є результат ділення y на x |
| **Ceiling** | Округлення до більшого цілого | **Ceiling (double x)** |
| **Cos** | Косинус | **Cos (double x)** |
| **Cosh** | Гіперболічний косинус | **Cosh (double x)** |
| **E** | База натурального логарифма (число e) | 2,71828182845905 |
| **Exp** | Експонента | $e^x$ записується як **Exp (x)** |
| **Floor** | Округлення до меншого цілого | **Floor (double x)** |
| **IEEE Remainder** | Залишок від ділення | **IEEERemainder (double x, double y)** |
| **Log** | Натуральний логарифм | $\text{Log}_e$ x записується як **Log (x)** |
| **Log10** | Десятковий логарифм | $\log_{10}$ x записується як **Log10 (x)** |
| **Max** | Максимум з двох чисел | **Max (x, y)** |
| **Min** | Мінімум з двох чисел | **Min (x, y)** |
| **PI** | Значення числа я | 3,14159265358979 |
| **Pow** | Степінь | $x^y$ записується як **Pow (x, y)** |
| **Round** | Округлення | **Round (3.1)** дасть в результаті 3, **Round (3.8)** дасть в результаті 4 |
| **Sign** | Знак числа | - |
| **Sin** | Синус | **Sin (double x)** |

## Практичне заняття №3. Робота з текстовими рядками

Текстові дані в C# представляються за допомогою ключових слів **string** і **char**, які є скороченими позначеннями для типів **System.String** і **System.Char**. **string** представляє набір символів (наприклад, "Text"), а **char** – одиночний символ (наприклад, 'T').

Символьний тип char призначений для зберігання символів в кодуванні Unicode. У цьому класі визначені статичні методи, що дозволяють задати вид і категорію символу, а також перетворити символ в верхній або нижній регістр і в число.

**Основні методи типу char**

| Метод | Пояснення |
|---|---|
| `GetNumericValue` | Повертає числове значення символу, якщо він є цифрою, або -1 в іншому випадку |
| `GetUnicodeCategory` | Повертає категорію символа |
| `IsControl` | Повертає true, якщо символ є керуючим |
| `IsDigit` | Повертає true, якщо символ є десяткової цифрою |
| `IsLetter` | Повертає true, якщо символ є буквою |
| `IsLetterOrDigit` | Повертає true, якщо символ є буквою або цифрою |
| `IsLower` | Повертає true, якщо символ заданий у нижньому регістрі |
| `IsNumber` | Повертає true, якщо символ є числом (десятковим або шістнадцятковим) |
| `IsPunctuation` | Повертає true, якщо символ є знаком пунктуації |
| `IsSeparator` | Повертає true, якщо символ є роздільником |
| `IsUpper` | Повертає true, якщо символ записаний у верхньому регістрі |
| `IsWhiteSpace` | Повертає true, якщо символ є пробільним (пробіл, символ нового рядка та символ повернення каретки) |
| `Parse` | Перетворити рядок в символ (рядок повинен складатися з одного символу) |
| `ToLower` | Перетворює символ в нижній регістр |
| `ToUpper` | Перетворює символ у верхній регістр |

Тип string, призначений для роботи з рядками символів в кодуванні Unicode. Тип string містить методи для визначення довжини символьних даних, пошуку підрядку в поточному рядку і перетворення символів між верхнім і нижнім регістрами.

Для рядків визначені такі оператори операції:

- присвоювання (=);
- перевірка на рівність (==);
- перевірка на нерівність (!=);
- звернення за індексом ([]);
- зчеплення (конкатенація) рядків (+).

Рядки рівні, якщо мають однакову кількість символів і збігаються посимвольно.

**Основні елементи типу string**

| **Елемент** | **Пяснення** |
|---|---|
| **`Compare`** | Порівняння двох рядків у лексикографічному (алфавітному) порядку |
| **`CompareOrdinal`** | Порівняння двох рядків за кодами символів |
| **`CompareTo`** | Порівняння поточного екземпляра рядки з іншим рядком |
| **`Concat`** | Конкатенація рядків. Метод допускає зчеплення довільного числа рядків |
| **`Copy`** | Створення копії рядка |
| **`Empty`** | Порожній рядок |
| **`Format`** | Форматування відповідно до заданих специфікаторами формату |
| **`IndexOf`, `IndexOfAny`, `LastIndexOf`, `LastIndexOfAny`** | Визначення індексів першого і останнього входження заданого підрядка або будь-якого символу із заданого набору |
| **`Insert`** | Вставка підрядка в задану позицію |
| **`Join`** | Злиття масиву рядків в єдиний рядок. Між елементами масиву вставляються роздільники |
| **`Length`** | Довжина рядка (кількість символів) |
| **`PadLeft`, `PadRight`** | Вирівнювання рядків по лівому або правому краю шляхом вставки потрібного числа пробілів на початку або в кінці рядка |
| **`Remove`** | Видалення підрядка із заданої позиції |

Рядкові літерали C# можуть містити керуючі послідовності, які дозволяють уточнювати те, як символьні дані виводяться на екран. Кожна керуюча послідовність починається з символу зворотної косої риски, за яким слідує знак, що інтерпретується.

**Керуючі послідовності в рядкових літералах**

| Керуюча послідовність | Опис |
|---|---|
| `\'` | Вставляє в строковий літерал символ одинарної лапки |
| `\"` | Вставляє в строковий літерал символ подвійної лапки |
| `\\` | Вставляє в строковий літерал символ зворотної косої риски. Особливо корисна при визначенні шляхів до файлів і мережевих ресурсів |
| `\a` | Змушує систему видавати звуковий сигнал, який в консольних додатках може служити аудіо-підказкою користувачеві |
| `\n` | Вставляє символ нового рядка |
| `\r` | Вставляє символ повернення каретки |
| `\t` | Вставляє в строковий літерал символ горизонтальної табуляції |

Метод Format замінює всі входження заповнювачів у фігурних дужках значеннями відповідних змінних зі списку виводу. Після номера заповнювача можна задати мінімальну ширину поля виводу, а також вказати специфікатор формату, який визначає форму подання виведеного значення.

У загальному вигляді заповнювач задається таким чином:

*{n [, m [: специфікатор_формата [число]]]}*

де n – номер заповнювача. Заповнювачі нумеруються з нуля, нульовий заповнювач замінюється значенням першої змінної зі списку виведення, перший заповнювач – значенням другої змінної і т. д. Параметр m визначає мінімальну ширину поля, яке відводиться під виведене значення. Якщо виведеному числа досить меншої кількості позицій, невикористовувані позиції заповнюються пробілами. Якщо числу потрібно більше позицій, параметр ігнорується.

Специфікатор формату визначає формат виведення значення. Наприклад, специфікатор C (Currency) означає, що параметр повинен фрматуватися як валюта з урахуванням національних особливостей подання, а специфікатор X (Hexadecimal) задає шестнадцатеричну форму подання виведеного значення. Після деяких специфікаторів можна задати кількість позицій, що відводяться під дробову частину виведеного значення.

Можливо використовати власні шаблони форматування. Після двокрапки задається вид виведеного значення посимвольно, причому на місці кожного

символу може стояти або #, або 0. Якщо вказаний знак #, на цьому місці буде виведена цифра числа, якщо вона не дорівнює нулю. Якщо вказаний 0, буде виведена будь-яка цифра, в тому числі і 0.

```
static void BasicStringFunctionality ()
{
        Console.WriteLine ("=> Basic String functionality:");
        string firstName = "Freddy";
        // Значення firstName.
        Console.WriteLine ("Value of firstName: (0)", firstName);
        // Довжина firstname.
        Console.WriteLine ("firstName has (0) characters.", FirstName.Length);
        // FirstName у верхньому регістрі.
        Console.WriteLine ("firstName in uppercase: {0}", firstName.ToUpper ());
        // FirstName в нижньому регістрі.
        Console.WriteLine ("firstName in lowercase: {0}", firstName.ToLower ());
        // Чи містить firstName букву y?
        Console.WriteLine ("firstName contains the letter y ?: {0}", firstName.Contains ("y"));
        // FirstName після заміни.
        Console.WriteLine ("firstName after replace: {0}", firstName.Replace ("dy", ""));
        Console.WriteLine ();
}
```

Тут пояснювати особливо нічого: метод просто викликає різні члени, такі як ToUpper () і Contains (), на локальній змінній string для отримання різноманітних форматів і виконання перетворень.

**Конкатенація строк**

Змінні string можуть бути зчеплені разом для створення рядків більшого розміру за допомогою операції + мови C#. Цей прийом формально називається конкатенацією рядків. Розглянемо наступну допоміжну функцію:

```
static void StringConcatenation ()
{
        Console.WriteLine ("=> String concatenation:");
        string si = "Programming is";
        string s2 =" awesome";
        string s3 = si + s2;
        Console.WriteLine (s3);
        Console.WriteLine ();
}
```

В результаті обробки компілятором символу + в C # видається виклик статичного методу String.Concat(). Тому конкатенацію рядків можна також здійснювати, викликаючи метод String.Concat() безпосередньо (хоча насправді це не дає якихось переваг, а лише збільшує обсяг введення).

Крім того, нижче наведений ще один приклад, в якому для залучення уваги кожен строковий літерал оснащений звуковим сигналом:

```
static void EscapeChars ()
{
        Console.WriteLine ("=> Escape characters: \ a");
        string strWithTabs = "Model \ tColor \ tSpeed \ tPet Name \ a
        Console.WriteLine (strWithTabs);
        Console.WriteLine ("Everyone loves V'Hello World \" \ a ");
        Console.WriteLine ("C: \\ MyApp \\ bin \\ Debug \ a");
        // Додати 4 порожніх рядки і знову видати звуковий сигнал.
        Console.WriteLine ("All finished. \ N \ n \ n \ a");
        Console.WriteLine ();
}
```

**Визначення дослівних рядків**

За рахунок додавання до строкових літералів префікса @ можна створювати так звані дослівні рядки. Дослівні рядки дозволяють відключати обробку керуючих послідовностей в літералах і виводити значення string в тому вигляді, в якому вони є. Ця можливість найбільш корисна при роботі з рядками, що представляють шляху до каталогів і мережевих ресурсів. Таким чином, замість використання керуючої послідовності \\ можна написати наступний код:

*Console.WriteLine (@"C:\MyApp\bin\Debug");*

Також зверніть увагу, що дослівні рядки можуть застосовуватися для зберігання пробілів у рядках, розділених на кілька рядків виводу.

Використовуючи дослівні рядки, можна також безпосередньо вставляти в літерали символи подвійної лапки, просто дублюючи лексему ":

*Console.WriteLine (@ "Cerebus said" "Darrr! Pret-ty sun-sets”);*

## Практичне заняття №4. Оператор розгалуження if/else

Рядкові літерали C# можуть містити керуючі послідовності, які Умовний оператор **if/else** використовується для розгалуження процесу обчислень на два напрямки.

Формат оператора:

*if (логічний_вираз) оператор_1; [else оператор_2; ]*

Спочатку обчислюється логічний вираз. Якщо він має значення true, виконується перший оператор, інакше – другий. Після цього управління передається на оператор, наступний за умовним. Гілка else може бути відсутня.

Якщо в гілці потрібно виконати декілька операторів, їх необхідно об’єднати в блок за допомогою фігурних дужок { }, інакше компілятор не зможе зрозуміти, де закінчується розгалуження. Блок може містити будь-які оператори, у тому числі опису та інші умовні оператори (але не може складатися з одних описів). Необхідно враховувати, що змінна, описана в блоці, поза блоком не існує.

Формат оператора:

*if (<логічний_вираз>) <оператор_1>; [Else <оператор_2>; ]*

Спочатку обчислюється логічний вираз. Якщо він має значення true, виконується перший оператор, інакше - другий. Після цього управління передається на оператор, наступний за умовним. Гілка else може бути відсутня.

Якщо в гілці потрібно виконати декілька операторів, їх необхідно об’єднати в блок, інакше компілятор не зможе зрозуміти, де закінчується розгалуження. Блок може містити будь-які оператори, у тому числі опису та інші умовні оператори (але не може складатися з одних описів). Необхідно враховувати, що змінна, описана в блоці, поза блоком не існує.

Якщо потрібно перевірити кілька умов, їх об'єднують знаками логічних умовних операцій. Наприклад, вираз у прикладі 2 буде істинним в тому випадку, якщо виконається одночасно умова a <b і одна з умов в дужках.

Якщо опустити внутрішні дужки, буде виконано спочатку логічне І, а потім - АБО.

Поширена помилка початківців - невірний запис перевірки на приналежність діапазону. Наприклад, щоб перевірити умову 0<x<1, не можна записати її в умовному операторі безпосередньо. Правильний спосіб запису: if (0 <x && x <1).

Слід уникати перевірки дійсних величин на рівність. Замість цього краще порівнювати модуль їх різниці з деяким малим числом. Це пов'язано з похибкою представлення дійсних значень у пам'яті:

```
float a, b; ...
if (a == b) ... // не рекомендується!
if (Math.Abs(a - b) <1e-6) ... // надійно!
```

Значення величини, з якою порівнюється модуль різниці, слід вибирати залежно від задачі, яка розв’язується. Знизу ця величина обмежена в класах Single і Double константою Epsilon (це мінімально можливе значення змінної таке, що 1.0 + Epsilon! = 1.0).

## Практичне заняття №5. Оператор множинного вибору switch/case

Оператор **switch/case** призначений для розгалуження процесу обчислень на кілька напрямів.

Формат оператора:

```
switch (вираз) {
case константний_вираз_1: [список_операторів_1]
case константний_вираз_2: [список_операторів_2]
case константний_вираз_n: [список_операторів_3]
[default: оператори]
```

Виконання оператора починається з обчислення виразу. Тип виразу найчастіше цілочисельний або рядковий. У загальному випадку вираз може бути будь-якого типу, для якого існує неявне перетворення до зазначених, а також типу переліку. Потім управління передається першому оператору зі списку, позначеного константним виразом, значення якого співпало з обчисленим.

Всі константні вирази повинні мати можливість неявного приведення до типу виразу в дужках. Якщо збігів не сталося, виконуються оператори, розташовані після слова **default** (а при його відсутності управління передається наступному за **switch/case** оператору).

Кожна гілка повинна закінчуватися явним оператором переходу, а саме оператором **break**, **goto** або **return**:

- оператор **break** виконує вихід з самого внутрішнього з операторів **switch/case**;
- оператор **goto** виконує перехід на вказану після нього мітку, зазвичай це мітка **case** однієї з нижчих гілок оператора **switch/case**;

- оператор **return** виконує вихід з функції, в тілі якої він записаний.

Оператор switch призначений для розгалуження процесу обчислень на кілька напрямів. Структурна схема оператора наведена на рис. 5.1.

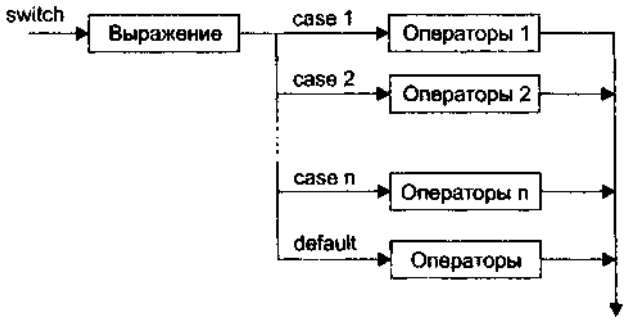



Рис. 5.2. Структурна схема оператора switch

Формат оператора:

*switch (<вираз>) {*
*case <константний_вираз_1>: [<список_операторів_1>]*
*case <константний_вираз_2>: [<список_операторів_2>]*
*case <константний_вираз_n>: [<список_операторів_3>]*
*[default: <оператори>]*

Виконання оператора починається з обчислення виразу. Тип виразу найчастіше цілочисельний або строковий. У загальному випадку вираз може бути будь-якого типу, для якого існує неявне перетворення до зазначених, а також типу переліку. Потім управління передається першому оператору зі списку, позначеного константним виразом, значення якого співпало з обчисленим.

Кожна гілка повинна закінчуватися явним оператором переходу, а саме оператором break, goto або return:

- оператор break виконує вихід з самого внутрішнього з операторів switch;
- оператор goto виконує перехід на вказану після нього мітку, зазвичай це мітка case однієї з нижчих гілок оператора switch;
- оператор return виконує вихід з функції, в тілі якої він записаний.

Оператор goto зазвичай використовують для послідовного виконання кількох гілок перемикача, однак оскільки це порушує читабельність програми, такого рішення слід уникати.

Розглянемо приклад програми, що реалізує найпростіший калькулятор на чотири дії.

```
using System;
namespace ConsoleApplicationl
{
        class Classl
        {
                static void Main()
                {
                        string buf;
                        double a, b, res;
                        Console.WriteLine ("Введіть першу операнд:");
                        buf = Console.ReadLine();
                        a = double.Parse(buf);
                        Console.WriteLine("Введіть знак операції");
                        char op = (char) Console.Read();
                        Console.ReadLine();
                        Console.WriteLine("Введіть другий операнд:");
                        buf = Console.ReadLine();
                        b = double.Parse(buf);
                        bool ok = true:
                        switch (op)
                        {
                                case '+': res = a + b: break:
                                case '-': res = a - b: break:
                                case '*': res = a * b; break:
                                case '/': res = a / b: break:
                                default: res = double.NaN; ok = false; break:
                        }
                        if (ok) Console.WriteLine ("Результат:" + res);
                        else Console.WriteLine ("Неприпустима операція");
                }
        }
}
```

Хоча наявність гілки default і не обов'язково, рекомендується завжди обробляти випадок, коли значення виразу не збігається ні з однією з констант. Це полегшує пошук помилок при налагодженні програми.

## Практичне заняття №6. Оператори циклу. Частина 1. Цикл for

Цикл – різновид керуючої конструкції у високорівневих мовах програмування, призначена для організації багаторазового виконання набору інструкцій. Цикл багаторазово повторює набір інструкцій, доки не будете виконана умова виходу з циклу. Одне повторення набору інструкцій (тіло циклу) також ще називають ітерацією.

Параметром циклу називається змінна, яка використовується при перевірці умови продовження циклу і примусово змінюється на кожній ітерації, причому, як правило, на одну і ту ж величину. Якщо параметр циклу цілочисельний, він називається лічильником циклу. Кількість повторень такого циклу можна визначити заздалегідь.

Цикл завершується, якщо умова його продовження не виконується. Можливо примусове завершення як поточної ітерації, так і циклу в цілому. Для цього служать оператори **break**, **continue**, **return** і **goto**.

Цикл з параметром має наступний формат:

```
for (ініціалізація; вираз; модифікації) оператор;
```

Ініціалізація служить для оголошення величин, що використовуються в циклі, і присвоєння їм початкових значень. У цій частині можна записати декілька операторів, розділених комою. Областю дії змінних, оголошених в частині ініціалізації циклу, є весь цикл. Ініціалізація виконується один раз на початку виконання циклу.

Вираз типу **bool** визначає умову виконання циклу: якщо його результат дорівнює **true**, цикл виконується.

Модифікації виконуються після кожної ітерації циклу і служать зазвичай для зміни параметрів циклу. У частині модифікацій можна записати декілька операторів через кому.

**Початкові установки** служать для того, щоб до входу в цикл задати значення змінних, які в ньому використовуються.

**Перевірка умови** продовження циклу виконується на кожній ітерації або до тіла циклу (тоді говорять про цикл із **передумовою**, схема якого

показана на рис. 4.4, а), або після тіла циклу (цикл з **післяумовою**, рис. 4.4, б). Різниця між ними полягає в тому, що тіло циклу з післяумовою 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.

**Параметром циклу** називається змінна, яка використовується при перевірці умови продовження циклу і примусово змінюється на кожній ітерації, причому, як правило, на одну і ту ж величину. Якщо параметр циклу цілочисельний, він називається **лічильником циклу**. Кількість повторень такого циклу можна визначити заздалегідь. Параметр є не у всякого циклу.

Цикл завершується, якщо умова його продовження не виконується. Можливо примусове завершення як поточної ітерації, так і циклу в цілому. Для цього служать оператори **break, continue, return і goto**. Передавати керування ззовні всередину циклу забороняється (при цьому виникає помилка компіляції).

Якщо в тілі циклу необхідно виконати більше одного оператора, необхідно укласти їх в блок за допомогою фігурних дужок.

### Цикл з параметром for

Цикл з параметром має наступний формат:

*for (ініціалізація; вираз; модифікації) оператор;*

**Ініціалізація** служить для оголошення величин, що використовуються в циклі, і присвоєння їм початкових значень. У цій частині можна записати декілька операторів, розділених комою, наприклад:

```
for ( int i = 0, j = 20; ...
int k, m;
for ( k = 1, m = 0; ...
```

**Областю дії змінних**, оголошених в частині ініціалізації циклу, є цикл. Ініціалізація виконується один раз на початку виконання циклу.

Вираз типу **bool** визначає **умову виконання циклу**: якщо його результат дорівнює **true**, цикл виконується. Цикл з параметром реалізований як цикл з передумовою.

**Модифікації** виконуються після кожної ітерації циклу і служать зазвичай для зміни параметрів циклу. У частині модифікацій можна записати декілька операторів через кому, наприклад:

```
for ( int i = 0, j = 20; i < 5 && j > 10; i++, j-- ) ...
```

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

Для прикладу обчислимо суму чисел від 1 до 100:

```
int s = 0;
for ( int i = 1; i <= 100; i++ ) s += i;
```

## Практичне заняття №7. Оператори циклу. Частина 2. Цикли while і do/while

Цикл з післяумовою **do/while** має такий формат:

```
do
   оператор
while вираз;
```

Спочатку виконується простий або складений оператор, який утворює тіло циклу, а потім обчислюється вираз (він повинен мати тип **bool**). Якщо вираз істинний, тіло циклу виконується ще раз і перевірка повторюється. Цикл завершується, коли вираз стане рівним **false** або в тілі циклу буде виконаний який-небудь оператор передачі управління.

Цей вид циклу застосовується в тих випадках, коли тіло циклу необхідно обов'язково виконати хоча б один раз, наприклад, якщо в циклі вводяться дані і виконується їх перевірка. Якщо ж такої необхідності немає, переважно користуватися циклом з передумовою.

Оператор **while** має такий формат:

```
while (вираз) оператор;
```

Вираз повинен бути логічного типу. Наприклад, це може бути операція відношення або просто логічна змінна. Якщо результат обчислення виразу дорівнює **true**, виконується простий або складений оператор (блок операторів). Ці дії повторюються до того моменту, поки результатом виразу не стане значення **false**. Після закінчення циклу управління передається на наступний за ним оператор.

Вираз обчислюється перед кожною ітерацією циклу. Якщо при першій перевірці вираз дорівнює **false**, цикл **while** не виконається жодного разу.

Поширеним прийомом програмування є організація нескінченного циклу із заголовком

```
while (true)
```

і примусовим виходом з тіла циклу за допомогою оператора передачі управління **break**.

Вираз повинен бути логічного типу. Наприклад, це може бути операція відношення або просто логічна змінна. Якщо результат обчислення виразу дорівнює true, виконується простий або складений оператор (блок). Ці дії повторюються до того моменту, поки результатом виразу не стане значення false. Після закінчення циклу управління передається на наступний за ним оператор.

Вираз обчислюється перед кожною ітерацією циклу. Якщо при першій перевірці вираз дорівнює false, цикл не виконається жодного разу.

В якості прикладу розглянемо програму, що виводить для аргументу x, що змінюється в заданих межах з заданим кроком, таблицю значень наступної функції:

$$y = \begin{cases} t, & x < 0 \\ tx, 0 & x < 10 \\ 2t, & x \geq 10 \end{cases}.$$

Зверніть увагу на те, що умова продовження циклу записано в його заголовку і перевіряється до входу в цикл. Таким чином, якщо задати кінцеве значення аргументу, меншу початкового, навіть при негативному кроці цикл не буде виконаний жодного разу.

Параметром цього циклу, тобто змінної, управляє його виконанням, є x. Блок модифікації параметра циклу представлений оператором, що виконуються на кроці 4. Для переходу до наступного значення аргументу поточне значення нарощується на величину кроку і заноситься в ту ж змінну. Початківці часто забувають про модифікацію параметра, в результаті програма «зациклюється». Якщо з вами сталася така неприємність, спробуйте для завершення програми натиснути клавіші Ctrl + Break, а надалі перед запуском програми перевіряйте:

- присвоєно Чи параметру циклу вірне початкове значення;
- чи змінюється параметр циклу на кожній ітерації циклу;
- чи вірно записано умова продовження циклу.

**Цикл з післяумовою do**

Цикл з післяумовою реалізує структурну схему, наведену на рис. 4.4, б, і має вигляд

*do оператор while вираз;*

Спочатку виконується простий або складений оператор, який утворює тіло циклу, а потім обчислюється вираз (воно повинно мати тип bool). Якщо вираз істинний, тіло циклу виконується ще раз і перевірка повторюється. Цикл завершується, коли вираз стане рівним false або в тілі циклу буде виконаний який-небудь оператор передачі управління.

Цей вид циклу застосовується в тих випадках, коли тіло циклу необхідно обов'язково виконати хоча б один раз, наприклад, якщо в циклі вводяться дані і виконується їх перевірка. Якщо ж такої необхідності немає, переважно користуватися циклом з передумовою.

Приклад програми, що виконує перевірку введення, приведений в лістингу:

```
using System.
namespace ConsoleApplication1
{   class Class1
    {   static void Main()
        {
            char answer;
            do
            {
                Console.WriteLine( "Купи слоника, а?" );
                answer = (char) Console.Read();
                Console.ReadLine();
            } while ( answer != 'y' );
```

Розглянемо ще один приклад застосування циклу з постусловіем - програму, що визначає корінь рівняння cos (x) = x методом розподілу навпіл з точністю 0,0001.

Вихідні дані для цього завдання - точність, результат - число, що представляє собою корінь рівняння 1. Обидва значення мають дійсний тип.

Суть методу розподілу навпіл дуже проста. Здається інтервал, в якому є рівно один корінь (отже, на кінцях цього інтервалу функція має значення різних знаків). Обчислюється значення функції в середині цього інтервалу. Якщо воно того ж знака, що і значення на лівому кінці інтервалу, значить, корінь знаходиться в правій половині інтервалу, інакше - в лівій. Процес повторюється для знайденої половини інтервалу до тих пір, поки його довжина не стане менше заданої точності.

У наведеній далі програмою (лістинг 4.6) вихідний інтервал заданий за допомогою констант, значення яких взяті з графіка функції. Для рівнянь, що мають кілька коренів, можна написати додаткову програму, що визначає (шляхом обчислення та аналізу таблиці значень функції) інтервали, що містять рівно один корінь 2.

```
using System;
namespace ConsoleApplication1
{   class Class1
    {   static void Main()
        {
            double x, left = 0, right = 1;
            do
            {
                x = ( left + right ) / 2;
                if ( ( Math.Cos(x) - x ) * ( Math.Cos(left) - left ) < 0 )
                     right = x;
                else left = x;
            } while ( Math.Abs( right - left ) < 1e-4 );
            Console.WriteLine( "Корень равен " + x );
        }
    }
}
```

У цю програму для надійності дуже корисно додати підрахунок кількості виконаних ітерацій і примусовий вихід з циклу при перевищенні їх розумної кількості.

Будь-який цикл **while** може бути приведений до еквівалентного йому циклу for і навпаки.

## Рекомендації по вибору оператора циклу

Оператори циклу взаємозамінні, але можна навести деякі рекомендації з вибору найкращого в кожному конкретному випадку.

Оператор do while зазвичай використовують, коли цикл потрібно обов'язково виконати хоча б раз, наприклад, якщо в циклі виробляється введення даних.

Оператором while зручніше користуватися в тих випадках, коли або число ітерацій заздалегідь невідомо, яких очевидних параметрів циклу немає, або модифікацію параметрів зручніше записувати не в кінці тіла циклу.

Оператор foreach застосовують для перегляду елементів різних колекцій об'єктів.

Оператор for підходить в більшості інших випадків. Однозначно - для організації циклів з лічильниками, тобто з цілочисельними змінними, які змінюють своє значення при кожному проході циклу регулярним чином (наприклад, збільшуються на 1).

Початківці часто роблять помилки при записі циклів. Щоб уникнути цих помилок, рекомендується:

- перевірити, чи всім змінним, яке трапляється в правій частині операторів присвоювання в тілі циклу, присвоєні до цього правильні початкові значення (а також можливо виконання інших операторів);
- перевірити, чи змінюється в циклі хоча б одна змінна, що входить до умова виходу з циклу;
- передбачити аварійний вихід з циклу після досягнення деякої кількості ітерацій (приклад наведено у наступному розділі);
- і, звичайно, не забувати про те, що якщо в тілі циклу потрібно виконати більше одного оператора, потрібно укладати їх у фігурні дужки.

## Оператори передачі управління

У C # є п'ять операторів, що змінюють природний порядок виконання обчислень:

- оператор безумовного переходу goto;
- оператор виходу з циклу break;
- оператор переходу до наступної ітерації циклу continue;
- оператор повернення з функції return;
- оператор генерації виключення throw.

Ці оператори можуть передати управління в межах блоку, в якому вони використані, і за його межі. Передавати керування всередину іншого блоку забороняється.

Перші чотири оператора розглядаються в цьому розділі, а оператор throw - далі в цій главі на с. 93.

### Оператор goto

Оператор безумовного переходу goto використовується в одній з трьох форм:

```
goto метка;
goto case константное_выражение;
goto default;
```

У тілі тієї ж функції повинна присутні рівно одна конструкція виду мітка: оператор;

Оператор goto мітка передає управління на позначений оператор. Метка - це звичайний ідентифікатор, областю видимості якого є функція, в тілі якої він заданий. Мітка повинна знаходитися в тій же області видимості, що і оператор переходу. Використання цієї форми оператора безумовного переходу виправдано у двох випадках:

- примусовий вихід вниз по тексту програми з декількох вкладених циклів або перемикачів;
- перехід з декількох точок функції вниз по тексту в одну точку (наприклад, якщо перед виходом з функції необхідно завжди виконувати які-небудь дії).

В інших випадках для запису будь-якого алгоритму існують більш відповідні конструкції, а використання оператора goto призводить тільки до ускладнення структури програми і утруднення налагодження. Застосування цього оператора порушує принципи структурного і модульного програмування, по яким всі блоки, що утворюють програму, повинні мати тільки один вхід і один вихід.

Друга і третя форми оператора goto використовуються в тілі оператора вибору switch. Оператор goto case константное_вираженіе передає управління на відповідну вирази зі сталими гілка, а оператор goto default - На гілку default. Треба зазначити, що реалізація оператора вибору в C # на рідкість невдала, і наявність у ньому оператора безумовного переходу ускладнює розуміння програми, тому краще обходитися без нього.

**Оператор break**

Оператор break використовується всередині операторів циклу або вибору для переходу в точку програми, що знаходиться безпосередньо за оператором, всередині якого знаходиться оператор break.

**Оператор continue**

Оператор переходу до наступної ітерації поточного циклу continue пропускає всі оператори, що залишилися до кінця тіла циклу, і передає керування на початок наступної ітерації.

## Практичне заняття №8. Одновимірні масиви

Масив – це сукупність даних одного типу, об’єднаних однією назвою. Приклади масивів з навколишнього світу: вулиця з будинками, де будинки є елементами масиву, колона автомобілів, шафа з книжками тощо.

Масив (структура даних) – це сукупність даних одного типу, об’єднаних однією назвою, що зберігаються в одному місці пам’яті комп’ютера та впорядковані за номерами комірок. Номера комірок називають індексами.

Масив в мові C# відноситься до змінних-посилань, тобто вони розташовуються в динамічній області пам'яті, тому створення масиву починається з виділення пам'яті під його елементи. Елементами масиву можуть бути величини як значущих, так і посилальних типів (у тому числі масиви). Масив значущих типів зберігає певні конкретні значення, а масив з даними посилальних типів – посилання на елементи. Всім елементам при створенні масиву присвоюються значення за замовчуванням: нулі для значущих типів і **null** – для посилальних.

Елементи масиву нумеруються з нуля, тому максимальний індекс елемента завжди на одиницю менше розмірності. Для звернення до елемента масиву після імені масиву вказується номер елемента в квадратних дужках.

З елементом масиву можна виконувати всі дії, що припустимі для змінних такого ж типу. При роботі з масивом автоматично виконується контроль виходу за його межі: якщо значення індексу виходить за межі масиву, генерується виключення під час роботи програми.

Масиви одного типу можна присвоювати один одному . При цьому відбувається присвоювання посилань, а не елементів, як і для будь-якого іншого об'єкта посилального типу.

У мові C # можна виділити три різновиди масивів: одновимірні, багатовимірні і ступінчасті (невирівняні).

В одновимірних масивах положення елементу задається тільки одним числом. Варіанти опису одновимірних масивів:

```
тип[] ім’я_масиву;
тип[] ім’я_масиву = new тип[кількість елементів];
тип[] ім’я_масиву = {значення елементів};
тип[] ім’я_масиву = new тип[] {значення елементів};
тип[] ім’я_масиву = new тип[кількість елементів]
{значення елементів};
```

де **тип** визначає тип даних кожного елементу масиву.

Звернемося до конкретному прикладу. У наведеної нижче рядку коду створюється масив типу int, Котрий складається з десяти елементів і зв'язується з змінної посилання на масив, іменованої sample.

*int [] sample = new int [10];*

У змінної sample зберігається посилання на область пам'яті, виділюваної для масиву оператором new. Ця область пам'яті повинна бути достатньо великий, щоб в ній могли зберігатися десять елементів масиву типу int.

Як і при створенні примірника класу, наведене вище оголошення масиву можна розділити на два окремих оператора. Наприклад:

*int [] sample;*
*sample = new int [10];*

У даному випадку змінна sample НЕ посилається на якийсь певний фізичний об'єкт, коли вона створюється в перший операторі. І лише після виконання другого оператора ця змінна посилається на масив.

Доступ до окремому елементу масиву здійснюється по індексом: Індекс позначає становище елемента в масиві. У мовою C # індекс перший елемента всіх масивів виявляється нульовим. У Зокрема, масив sample складається з 10 елементів з індексами від 0 до 9. Для індексування масиву достатньо вказати номер необхідного елемента в квадратних дужках. Так, перший елемент масиву sample позначається як sample [0], а останній його елемент - як sample [9]. Нижче наведено приклад програми, в якій заповнюються всі 10 елементів масиву sample.

```
// Продемонструвати одновимірний масив. using System;
class ArrayDemo {
        static void Main ()
        {
                int [] sample = new int [10]; int i;
                for (i = 0; i <10; i = i + 1) sample [i] = i;
                for (i = 0; i <10; i = i + 1)
                Console.WriteLine ("sample [" + i + "]:" + sample [i]);
        }
}
```

При виконанні цієї програми виходить наступний результат.

```
sample [0]: 0
sample [1]: 1
sample [2]: 2
sample [3]: 3
sample [4]: 4
sample [5]: 5
sample [6]: 6
sample [7]: 7
sample [8]: 8
sample [9]: 9
```

Масиви часто застосовуються в програмуванні тому, що вони дають можливість легко звертатися з великим числом взаємопов'язаних змінних. Наприклад, в наведеній нижче програмою виявляється середнє арифметичне ряду значень, що зберігаються в масиві nums, Котрий циклічно опитується з допомогою оператора циклу for.

```
// Обчислити середнє арифметичне ряду значень. using System;
class Average {
   static void Main () {
      int [] nums = new int [10]; int avg = 0;
      nums [0] = 99;
      nums [1] = 10;
      nums [2] = 100;
      nums [3] = 18;
      nums [4] = 78;
      nums [5] = 23;
      nums [6] = 63;
      nums [7] = 9;
      nums [8] = 87;
      nums [9] = 49;
      for (int i = 0; i <10; i ++) avg = avg +
         nums [i];
      avg = avg / 10;
```

```
        Console.WriteLine ("Середнє:" + avg);
    }
}
```

Результат виконання цієї програми виглядає наступним чином.

Середнє: 53

## Присвоєння посилань на масиви

Присвоєння значення однією змінної посилання на масив другий змінної, по суті, означає, що обидві змінні посилаються на один і той ж масив,

і в цьому відношенні масиви нічим НЕ відрізняються від будь-яких інших об'єктів. Таке присвоювання НЕ призводить ні до створенню копії масиву, ні до копіюванню вмісту одного масиву в інший. В якості прикладу розглянемо наступну програму.

```
// Присвоєння посилань на масиви. using System;
class AssignARef {
        static void Main ()
        {
                int i;
                int [] nums1 = new int [10];
                int [] nums2 = new int [10];
                for (i = 0; i <10; i ++) nums1 [i] = i;
                        for (i = 0; i <10; i ++) nums2 [i] = -i;
                                Console.Write ("Вміст масиву nums1:");
                for (i = 0; i <10; i ++)
                        Console.Write (nums1 [i] + "");
                Console.WriteLine ();
                Console.Write ("Вміст масиву nums2:");
                for (i = 0; i <10; i ++)
                        Console.Write (nums2 [i] + "");
                Console.WriteLine ();
                nums2 = nums1; // Тепер nums2 посилається на nums1
                Console.Write ("Вміст масиву nums2 \ n" + "після присвоювання:");
                for (i = 0; i <10; i ++)
                        Console.Write (nums2 [i] + "");
                Console.WriteLine ();
                nums2 [3] = 99;
                Console.Write ("Вміст масиву nums1:");
                for (i = 0; i <10; i ++) Console.Write (numsl [i] + "");
                        Console.WriteLine ();
        }
}
```

Виконання цієї програми призводить до наступного результату.

Вміст масива nums1: 0 1 2 3 4 5 6 7 8 9

Вміст масива nums2: 0 -1 -2 -3 -4 -5 -6 -7 -8 -9

Вміст масива nums2: 0 1 2 3 4 5 6 7 8 9

Вміст масиву nums1: 0 1 2 99 4 5 6 7 8 9

Як бачите, після присвоювання змінної nums2 значення змінної nums1 обидві змінні посилання на масив посилаються на один і той же об'єкт.

**Застосування властивості Length**

Реалізація в C # масивів в вигляді об'єктів дає цілий ряд переваг. Одне з них полягає в тому, що з кожним масивом пов'язано властивість Length, містить число елементів, з яких може складатися масив. Отже, у кожного масиву мається спеціальне властивість, дозволяє визначити його довжину. Нижче наведено приклад програми, в якій демонструється цю властивість.

```
// Використовувати властивість Length масиву. using
System;
class LengthDemo {static void Main () {
      int [] nums = new int [10];
      Console.WriteLine ("Довжина масиву nums дорівнює" + nums.Length);
      // Використовувати властивість Length для ініціалізації масиву nums. for (int i =
      0; i <nums.Length; i ++)
        nums [i] = i * i;
      // А тепер скористатися властивістю Length
      // Для виведення вмісту масиву nums. Console.Write ("Вміст
      масиву nums:"); for (int i = 0; i <nums.Length; i ++)
        Console.Write (nums [i] + "");
      Console.WriteLine ();
    }
}
```

При виконанні цієї програми виходить наступний результат.

Довжина масиву nums дорівнює 10

Вміст масиву nums: 0 1 4 9 16 25 36 49 64 81

Зверніть увага на то, як в класі LengthDemo властивість nums.Length використовується в циклах for для управління числом повторюваних кроків циклу. У кожного масиву мається своя довжина, тому замість відстеження розміру масиву вручну можна використовувати інформацію про його довжині. Слід, однак, мати в увазі, що значення властивості Length ніяк НЕ відображає число елементів, які в ньому використовуються на самому справі.

Властивість Length містить лише число елементів, з яких може складатися масив.

Коли запитується довжина багатовимірного масиву, то повертається загальне число елементів, з яких може складатися масив, як у наведеному нижче прикладі коду.

```
// Використовувати властивість Length тривимірного масиву. using
System;
class LengthDemo3D {static void Main
  () {
    int [,,] nums = new int [10, 5, 6];
    Console.WriteLine ("Довжина масиву nums дорівнює" + nums.Length);
  }
}
```

При виконанні цього коду виходить наступний результат.

Довжина масиву nums дорівнює 300

Як підтверджує наведений вище результат, властивість Length містить число елементів, з яких може складатися масив (в даному випадку - 300 (10 × 5 × 6) елементів). Тим НЕ менш властивість Length не можна використовувати для визначення довжини масиву в окремому його вимірі.

Завдяки наявності у масивів властивості Length операції з масивами під багатьох алгоритмах стають більш простими, а значить, і більш надійними. У якості прикладу властивість Length використовується в наведеної нижче програмою з метою поміняти місцями вміст елементів масиву, скопіювавши їх в зворотному порядку в інший масив.

```
// Поміняти місцями вміст елементів масиву. using System;
class RevCopy {
  static void Main () {int i, j;
    int [] nums1 = new int [10]; int [] nums2 =
    new int [10];
    for (i = 0; i <nums1.Length; i ++) nums1 [i] = i;
    Console.Write ("Початковий вміст масиву:"); for (i = 0; i
    <nums2.Length; i ++)

      Console.Write (nums1 [i] + ""); Console.WriteLine ();
    // Скопіювати елементи масиву nums1 в масив nums2 в зворотному порядку.
    if (nums2.Length> = nums1.Length) // перевірити, чи достатньо
                                          // Довжини масиву nums2 for (i = 0, j =
      nums1.Length-1; i <nums1.Length; i ++, j--)
            nums2 [j] = nums1 [i];
```

```
        Console.Write ("Вміст масиву в зворотному порядку:"); for (i = 0; i
        <nums2.Length; i ++)
          Console.Write (nums2 [i] + "");
        Console.WriteLine ();
    }
}
```

Виконання цієї програми дає наступний результат.

Початковий вміст масиву: 0 1 2 3 4 5 6 7 8 9

Вміст масиву в зворотному порядку: 9 8 7 6 5 4 3 2 1 0

У даному прикладі властивість Length допомагає виконати два важливі функції. По -перше, воно дозволяє переконатися в тому, що довжини цільового масиву досить для зберігання вмісту вихідного масиву. І по-друге, воно надає умова для завершення циклу for, в якому виконується копіювання вихідного масиву в зворотному порядку. Звичайно, в цьому простому прикладі розміри масивів неважко з’ясувати і без властивості Length, але аналогічний підхід може бути застосований в цілому ряді інших, більш складних ситуацій.

## Практичне заняття №9. Двомірні масиви. Основи роботи з матрицями

Багатовимірним називається такий масив, який характеризується двома або більше вимірами, причому доступ до кожному елементу такого масиву здійснюється за допомогою певної комбінації двох або більше індексів.

Найпростішою формою багатовимірного масиву є двовимірний масив. Положення будь-якого елемента в двовимірному масиві визначається двома індексами. Такий масив можна уявити в вигляді таблиці, на рядки якої вказує один індекс, а на стовпці – інший.

Варіанти опису двовимірного масиву:

```
тип[,] ім’я_масиву;
тип[,] ім’я_масиву = new тип[розмір_1, розмір_2];
тип[,] ім’я_масиву = {значення елементів};
тип[,] ім’я_масиву = new тип[,] {значення елементів};
тип[,] ім’я_масиву = new тип[розмір_1, розмір_2]
{значення елементів};
```

Якщо список ініціалізації не заданий, розмірності можуть бути не тільки константами, але і виразами типу, що приводиться до цілого. До елементу двовимірного масиву звертаються, вказуючи номери рядка і стовпця, на перетині яких він розташований.

```
using System;
class ThreeDMatrix
{
        static void Main ()
        {
                int [,,] m = new int [3, 3, 3];
                int sum = 0;
                int n = 1;
                for (int x = 0; x <3; x ++)
                        for (int y = 0; y < 3; y ++)
                                for (int z = 0; z <3; z ++) m [x, y, z] = n ++;
                sum = m [0, 0, 0] + m [1, 1,1] + m [2, 2, 2];
                Console.WriteLine ("Сума значень по першій діагоналі:" + sum);
        }
}
```

Ось який результат дає виконання цієї програми.

Сума значень по першій діагоналі: 42

## Ініціалізація багатовимірних масивів

Для ініціалізації багатовимірного масиву достатньо укласти в фігурні дужки список ініціалізаторов кожного його розміру. Нижче в якості прикладу наведена загальна форма ініціалізації двовимірного масиву:

```
тип [,] імя_массіва = {
{val, val, val, ..., Val},
{val, val, val, ..., Val},
{val, val, val, ..., Val}
};
```

де val позначає инициализирующее значення, а кожен внутрішній блок - окремий ряд. Перше значення в кожному ряду зберігається на перший позиції в масиві, друге значення - на другий позиції і т.д. Зверніть увага на то, що блоки ініціалізаторов розділяються запитом, а після завершальній ці блоки закриває фігурної дужки ставиться крапка з комою.

У якості прикладу нижче наведена програма, в якої двовимірний масив sqrs инициализируется числами від 1 до 10 і квадратами цих чисел.

```
// Ініціалізувати двовимірний масив.
using System;
class Squares
{
        static void Main ()
        {
                int [,] sqrs = {
                        {1, 1},
                        {2, 4},
                        {3, 9},
                        {4, 16},
                        {5, 25},
                        {6, 36},
                        {7, 49},
                        {8, 64},
                        {9, 81},
                        {10, 100}
                };
                int i, j;
                for (i = 0; i <10; i ++)
                {
                        for (j = 0; j <2; j ++)
                        Console.Write (sqrs [i, j] + "");
                        Console.WriteLine ();
                }
        }
}
```

При виконанні цієї програми виходить наступний результат.

1 січня
4 лютого
3 вересня
16 квітня
25 травня
6 36
7 49
8 64
9 81
10100

## Ступінчасті масиви

У наведених вище прикладах застосування двовимірного масиву, по суті, створювався так званий прямокутний масив. Двовимірний масив можна представити у вигляді таблиці, в якої довжина кожної рядки залишається незмінною по всьому масиву. Але в C # можна також створювати спеціальний тип двовимірного масиву, званий ступінчастим масивом. Ступінчастий масив представляє собою масив масивів, в якому довжина кожного масиву може бути різною. Отже, ступінчастий масив може бути використаний для складання таблиці з рядків різної довжини.

Ступінчасті масиви оголошуються з допомогою ряду квадратних дужок, в яких вказується їх розмірність. Наприклад, для оголошення двовимірного ступеневої масиву служить наступна загальна форма:

*тип [] [] імя_массіва = new тип [розмір] [];*

де розмір позначає число рядків в масиві. Пам'ять для самих рядків розподіляється індивідуально, і тому довжина рядків може бути різною. Наприклад, в наведеному нижче фрагменті коду оголошується ступінчастий масив jagged. Пам'ять спочатку розподіляється для його перший вимірювання автоматично, а потім для другий вимірювання вручну.

```
int [] [] jagged = new int [3] [];
jagged [0] = new int [4];
jagged [1] = new int [3];
jagged [2] = new int [5];
```

Після виконання цього фрагмента коду масив jagged виглядає так, як показано нижче.

Тепер неважко зрозуміти, чому такі масиви називаються ступінчастими! Після створення ступеневої масиву доступ до його елементам здійснюється по індекс, що вказує в окремих квадратних дужках. Наприклад, в наступної рядку коду елементу масиву jagged, знаходиться на позиції з координатами (2,1), присвоюється значення 10.

*jagged [2] [1] = 10;*

Зверніть увага на синтаксичні відмінності в доступі до елементу ступеневої та прямокутного масиву.

У наведеному нижче прикладі програми демонструється створення двовимірного ступеневої масиву.

```
// Продемонструвати застосування східчастих масивів. using System;
class Jagged {
  static void Main () {
    int [] [] jagged = new int [3] [];
    jagged [0] = new int [4];
    jagged [1] = new int [3];
    jagged [2] = new int [5];
    int i;
    // Зберегти значення в першому масиві.
    for (i = 0; i <4; i ++)
      jagged [0] [i] = i;
    // Зберегти значення в другому масиві.
    for (i = 0; i <3; i ++)
      jagged [1] [i] = i;
    // Зберегти значення в третьому масиві.
    for (i = 0; i <5; i ++)
      jagged [2] [i] = i;
    // Вивести значення з перших масиву.
    for (i = 0; i <4; i ++)
      Console.Write (jagged [0] [i] + "");
    Console.WriteLine ();
    // Вивести значення з другого масиву.
    for (i = 0; i <3; i ++)
      Console.Write (jagged [1] [i] + "");
      Console.WriteLine ();
    // Вивести значення із третього масиву.
    for (i = 0; i <5; i ++)
      Console. Write (jagged [2] [i] + "");
    Console.WriteLine ();
  }
```

```
}
```

Виконання цієї програми призводить до наступного результату.

0 1 2 3

0 1 2

0 1 2 3 4

Ступінчасті масиви знаходять корисне застосування НЕ у всіх, а лише в деяких випадках. Так, якщо потрібно дуже довгий двовимірний масив, Котрий заповнюється не повністю, тобто такий масив, в якому використовуються НЕ всі, а лише окремі його елементи, то для цієї мети ідеально підходить ступінчастий масив.

Ступінчасті масиви представляють собою масиви масивів, і тому вони НЕ обов'язково повинні складатися з одновимірних масивів. Наприклад, у наведеній нижче рядку коду створюється масив двовимірних масивів.

```
int [] [,] jagged = new int [3] [,];
```

У наступної рядку коду елементу масиву jagged [0] присвоюється посилання на масив розмірами 4 × 2.

```
jagged [0] = new int [4, 2];
```

А в наведеної нижче рядку коду елементу масиву jagged [0] [1,0] присвоюється значення змінної i.

```
jagged [0] [1,0] = i;
```

## Список рекомендованої літератури


1. Голуб Б.М. C#. Концепція та синтаксис. Навч. посібник. – Львів: Видавничий центр ЛНУ імені Івана Франка, 2006. – 136 с.
2. Троелсен Э. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е изд.: Пер. с англ. – М.: ООО “И.Д. Вильямс”, 2013. – 1312 с.
3. Шилдт Г. C# 3.0. Полное руководство / Пер. с англ. – М.: Диалектика-Вильямс, 2009. – 992 с.
4. Котов, О.М. Язык C#: краткое описание и введение в технологии программирования: учебное пособие / О.М. Котов. – Екатеринбург: Изд-во Урал. ун-та, 2014. – 208 с.
5. Уотсон К. Microsoft Visual C# 2008. Базовый курс / К. Уотсон, К. Нейгел, Я.Х. Педерсен, Дж. Д. Рид, М. Скиннер, Э. Уайт. / Пер. с англ. – М.: Диалектика-Вильямс, 2009. – 1216 с.


## Додаткова література:


1. Павловская Т.А. C#. Программирование на языке высокого уровня: Учебник для вузов. – СПб.: Питер, 2014. – 432 с.
2. Нейгел К., Ивьен Б., Глинн Д. C# 4.0 и платформа .NET 4 для профессионалов / Пер. с англ. – К.: Диалектика, 2011. – 1440 с.
3. Рихтер Дж. Программирование на платформе Microsoft .NET Framework 2.0 на языке C#. / Пер. с англ. – СПб: Питер, М: Русская Редакция, 2007. – 656 с.
4. Агапов В.П. Основы программирования на языке C#: учебное пособие / В. П. Агапов. – Москва: МГСУ, 2012. – 128 с.
5. Петцольдт Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 1. /Пер. с англ. – М.: Издательско-торговый дом «Русская Редакция», 2002. – 576 с.
6. Андрианова А.А., Исмагилов Л.Н., Мухтарова Т.М. Объектно-ориентированное программирование на C#: Учебное пособие / А.А. Андрианова, Л.Н. Исмагилов, Т.М. Мухтарова. – Казань: Казанский (Приволжский) федеральный университет, 2012. – 134 с.
7. Вирт, Н. Алгоритмы и структуры данных [Текст]: пер. с англ. / Никлаус Вирт. – СПб: Невский Диалект, 2008. – 352 с.
8. Культин Н. Б. C# в задачах и примерах. – СПб.: БХВ-Петербург, 2007. – 240 с.
9. Ватсон К. C# / К. Ватсон, М. Беллиназо, О. Корне, Д. Эспиноза, 3. Грин-фосс и др. / Пер. с англ. яз. – М.: Изд. «Лори». – 2005, 862 с.
10. Бокс Д., Селлз К. Основы платформы .NET, том 1. Общеязыковая исполняющая среда.: Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 288 с.
11. Либерти Дж., Программирование на C#. Создание .NET-приложений. Изд. 2-е. / Пер. с англ. – Спб.: Изд. «Символ», 2003. – 68