МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

# ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ

## КОНСПЕКТ ЛЕКЦІЙ

## ЧАСТИНА 1

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського*
*як навчальний посібник для здобувачів ступеня бакалавра*
*за спеціальністю 141 «Електроенергетика, електротехніка та електромеханіка»*

Київ
КПІ ім. Ігоря Сікорського
2020

Обчислювальна техніка та програмування: Конспект лекцій (Частина 1) [Електронний ресурс] : навч. посіб. для студ. спеціальності 141 «Електроенергетика, електротехніка та електромеханіка» / КПІ ім. Ігоря Сікорського ; уклад.: Г.О.Труніна, Д.В.Настенко, А.Б.Нестерко. – Електронні текстові данні (1 файл: 1,514 Кбайт). – Київ : КПІ ім. Ігоря Сікорського, 2020. – 117 с.

*Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 3 від 05.11.2020р.) за поданням Вченої ради Факультету електроенерготехніки та автоматики (протокол № 2 від 28.09.2020 р.)*

Електронне мережне навчальне видання

# ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ

## КОНСПЕКТ ЛЕКЦІЙ

## ЧАСТИНА 1

Укладачі: *Труніна Ганна Олексіївна,* канд. техн. наук, ст.викл.,
*Настенко Дмитро Васильович*, ст.викл.,
*Нестерко Артем Борисович,* канд. техн. наук, ст.викл.

Відповідальний редактор: *Яндульський О.С.,* д.т.н., проф.

Рецензент: *Кацадзе Т.Л., к.т.н., доц.*
*КПІ ім.Ігоря Сікорського, ФЕА,*
*кафедра електричних мереж*

В навчальному посібнику викладено основи програмування з використанням платформи .NET. Посібник призначено для здобувачів ступеня бакалавра, що вивчають одну з найпоширеніших мов програмування – C#. В посібнику наведено методи та алгоритми розв'язання інженерних задач за допомогою сучасних алгоритмічних мов з використанням сучасних комп'ютерних технологій. Посібник містить 16 підрозділів, в яких наведено теоретичні основи програмування та приклади з детальним описом послідовності розв'язання задач на мові C#.

**ЗМІСТ**

# ВСТУП

Навчальна дисципліна «Обчислювальна техніка та програмування» належить до обов'язкових дисциплін циклу природничо-наукової підготовки спеціальності 141 «Електроенергетика, електротехніка та електромеханіка», освітньо-кваліфікаційного рівня бакалавр.

Розвиток економіки, промисловості, науки і техніки, сфери освіти сьогодні значною мірою залежить від масового запровадження та використання обчислювальної техніки. Це вимагає підготовки і перепідготовки фахівців з програмування і використання персональних комп'ютерів.

Предмет навчальної дисципліни: методи та алгоритми розв'язання інженерних задач за допомогою сучасних алгоритмічних мов з використанням сучасних комп'ютерних технологій. Посібник призначено для студентів, що вивчають одну з найпоширеніших мов програмування – C#. Вибір мови програмування C# пояснюється такими чинниками:

- простотою і природністю основних конструкцій мови, що дозволяє швидко її освоїти і створювати алгоритмічно складні програми;
- можливістю використання розвинених засобів подання структур даних, що забезпечує зручність роботи як з числовою, так і з символьною інформацією;
- відповідністю принципам об'єктно-орієнтованого програмування, що робить програми наочними;
- наявністю бібліотеки процедур і функцій для роботи як з текстовою, так і з графічною інформацією, що дозволяє створювати досить складні програми.

# 1. ПОНЯТТЯ АЛГОРИТМУ, ЙОГО ВЛАСТИВОСТІ, БАЗОВІ ЕЛЕМЕНТИ ПОБУДОВИ АЛГОРИТМІВ. МОВИ ПРОГРАМУВАННЯ ТА СФЕРИ ЇХ ВИКОРИСТАННЯ

## 1.1. Поняття алгоритмізації та алгоритму.

Процес вирішення багатьох завдань людина може передавати технічним пристроям - персональним комп’ютерам, автоматам, роботам і т.д. Застосування технічних пристроїв пред'являє дуже суворі вимоги до точності опису правил і послідовності виконання дій.

**Алгоритмізація** це розділ інформатики, що вивчає методи і прийоми побудови алгоритмів, а також їх властивості.

Джерелами виникнення алгоритмів служать: спостереження, експеримент, наукова теорія, минулий досвід і ін.

Перед розв’язанням будь-якої задачі за допомогою персонального комп'ютера виконуються наступні етапи:

- постановка цієї задачі,
- побудова сценарію і алгоритмізація задачі.

На етапі постановки завдання

- описуються вихідні дані та передумови,
- формуються правила початку та закінчення розв’язання задачі (досягнення мети), тобто розробляється інформаційна або еквівалентна їй математична модель.

Методом проб і помилок ведеться пошук методу розв'язання задачі (методу обчислень, методу перебору варіантів, методу розпізнавання). На підставі цього методу розробляється вихідний алгоритм, реалізація якого принципово можлива за допомогою персонального комп’ютера.

**Алгоритмізація задачі** - процес розробки (проектування) алгоритму розв'язання задачі за допомогою персонального комп’ютера на основі її умови і вимог до кінцевого результату.

При розробці вихідного алгоритму і навіть при виборі моделі користувач, тобто людина, яка вирішує конкретну задачу, повинна мати уявлення про математичне забезпечення персонального комп’ютера.

Результатом алгоритмізації задачі є алгоритм.

**Алгоритм** — набір інструкцій, які описують порядок дій виконавця, щоб досягти результату розв'язання задачі за скінченну кількість дій; система правил виконання дискретного процесу, яка досягає поставленої мети за

скінченний час. Для візуалізації алгоритмів часто використовують блок-схеми [1].

Алгоритм по відношенню до персонального комп'ютера - набір операцій і правил їх чергування, за допомогою якого, починаючи з деяких вихідних даних, можна вирішити завдання.

**Операція (команда) алгоритму** - вказівка про виконання окремої закінченої дії.

Термін «алгоритм» походить від імені узбецького вченого IX ст. Аль-Хорезмі, який у своїй праці «Кітаб аль-джебр ва-ль-мукабала», перекладеній в XII в. з арабської на латинь, виклав правила арифметичних дій над числами в позиційній десятковій системі числення. Ці правила і називали алгоритмами [1].

Самі алгоритми залежно від мети, початкових умов завдання, шляхів її вирішення і визначеності дій виконавця підрозділяються на:

- детерміновані, або жорсткі;
- гнучкі алгоритми, наприклад, стохастичні.

**Детермінований** алгоритм задає певні дії, позначаючи їх в єдиній і визначеній послідовності, забезпечуючи тим самим однозначний результат, якщо виконуються ті умови процесу чи завдання, для яких розроблений алгоритм.

**Імовірнісний (стохастичний)** алгоритм надає програму розв'язання задачі кількома шляхами або способами, що призводять до ймовірного досягненню результату.

**Евристичний алгоритм** (від грецького слова "еврика" - "Я знайшов") - це такий алгоритм, у якому досягнення кінцевого результату програми дій однозначно не визначено, так само як не визначена вся послідовність дій виконавця.

Часто, для отримання нових алгоритмів, використовуються вже існуючі алгоритми. Це здійснюється комбінуванням вже відомих алгоритмів або за допомогою еквівалентних перетворень алгоритмів.

Алгоритми називаються **еквівалентними**, якщо результати, одержані за допомогою цих алгоритмів для одних і тих же вхідних даних, однакові.

Типовий приклад еквівалентного перетворення алгоритмів - переклад з однієї алгоритмічної мови на іншу.

У загальному випадку алгоритмізація обчислювального процесу включає наступні дії:

- виділення незалежних етапів обчислювального процесу і розбивку кожного етапу на окремі кроки - декомпозицію задачі;
- формальний запис змісту кожного етапу або кроку;
- визначення загального порядку виконання етапів або кроків;
- перевірку правильності алгоритму.

**Декомпозиція** передбачає поділ складного завдання на сукупність простіших підзадач.

## 1.2. Властивості алгоритмів

Властивості алгоритму - набір властивостей, що відрізняють алгоритм від будь-яких інших наборів дій і забезпечують його автоматичне виконання.

Алгоритми мають ряд важливих властивостей [2]:

**Скінченність** - алгоритм має завжди завершуватись після виконання скінченної кількості кроків. Процедуру, яка має решту характеристик алгоритму, без, можливо, скінченності, називають методом обчислень.

**Дискретність** - процес, що визначається алгоритмом, можна розділити на окремі елементарні етапи, кожен з яких називається кроком алгоритму.

**Визначеність** - кожен крок алгоритму має бути точно визначений. Дії, які необхідно здійснити, повинні бути чітко та недвозначно визначені для кожного можливого випадку.

**Вхідні дані** - алгоритм має деяку кількість (можливо, нульову) вхідних даних, тобто, величин, заданих до початку його роботи або значення яких визначають під час роботи алгоритму.

**Вихідні дані** - алгоритм має одне або декілька вихідних даних, тобто, величин, що мають досить визначений зв'язок із вхідними даними.

**Ефективність** - Алгоритм вважають ефективним, якщо всі його оператори досить прості для того, аби їх можна було точно виконати за скінченний проміжок часу.

**Масовість** - властивість алгоритму, яка полягає в тому, що алгоритм повинен забезпечувати розв'язання будь-якої задачі з класу однотипних задач за будь-якими вхідними даними, що належать до області застосування алгоритму.

## 1.3. Способи опису алгоритмів. Програми

Для чіткого завдання різних алгоритмів потрібно мати таку систему формальних позначень і правил, щоб сенс всякої послідовності дій

трактувався точно і однозначно. Відповідні системи правил називають мовами опису алгоритмів.

Опис алгоритму залежить від виконавця – людини чи технічного засобу (комп’ютера).

До засобів опису алгоритмів відносяться такі основні способи їх подання:

- словесний;
- графічний;
- псевдокод;
- табличний;
- програмний.

На практиці в якості виконавців алгоритмів використовуються комп'ютери чи інші обчислювальні пристрої. Тому алгоритм, призначений для виконання на комп'ютері, повинен бути записаний на зрозумілій йому мові.

Мова для запису алгоритмів повинна бути формалізованою. Таку мову прийнято називати **мовою програмування**, а запис алгоритму на цій мові - програмою для комп'ютера.

Автором першої **програми** вважать доньку відомого англійського поета лорда Байрона - Аду Лавлейс. Яка описала алгоритм обчислення чисел Бернуллі (1.1) на аналітичній машині Беббіджа. Цей алгоритм визнано першою програмою, спеціально реалізованою, щоб виконуватися на ЕОМ, і через це його розробницю вважають першим програмістом, дарма що машина Беббіджа не була сконструйована за життя Ади. Вона ввела у вжиток терміни «цикл» і «робоча комірка».

$$B_n = \frac{-1}{n+1}\sum_{k=1}^{n} C_{n+1}^{k+1} B_{n-k}, \quad n \in N \tag{1.1}$$

## 1.4. Мови програмування

**Мова програмування** – це формальна знакова система, призначена для запису комп'ютерних програм.

Наведемо приклади таких мов. За даними індексу Tiobe, що аналізує популярність мов програмування по всьому світу на основі згадування тієї чи іншої мови в популярних пошукових системах, таких як Google, Bing і Baidu, десятка мов з найвищим рейтингом на червень 2018 така (табл.1).

Таблиця 1.1 – Рейтинг Tiobe на грудень 2019

| Dec 2019 | Dec 2018 | Change | Programming Language | Ratings | Change |
|---|---|---|---|---|---|
| 1 | 1 | | Java | 17.253% | +1.32% |
| 2 | 2 | | C | 16.086% | +1.80% |
| 3 | 3 | | Python | 10.308% | +1.93% |
| 4 | 4 | | C++ | 6.196% | -1.37% |
| 5 | 6 | ^ | C# | 4.801% | +1.35% |
| 6 | 5 | v | Visual Basic .NET | 4.743% | -2.38% |
| 7 | 7 | | JavaScript | 2.090% | -0.97% |
| 8 | 8 | | PHP | 2.048% | -0.39% |
| 9 | 9 | | SQL | 1.843% | -0.34% |
| 10 | 14 | ^^ | Swift | 1.490% | +0.27% |
| 11 | 17 | ^^ | Ruby | 1.314% | +0.21% |
| 12 | 11 | v | Delphi/Object Pascal | 1.280% | -0.12% |
| 13 | 10 | v | Objective-C | 1.204% | -0.27% |
| 14 | 12 | v | Assembly language | 1.067% | -0.30% |

Залежно від ступеня деталізації кроків алгоритму для комп’ютера, зазвичай визначається рівень мови програмування - чим менше деталізація, тим вище рівень мови.

За цим критерієм можна виділити такі рівні мов програмування:

- машинні;
- машинно-орієнтовані (мови асемблера);
- машинно-незалежні (мови високого рівня).

Машинні та машинно-орієнтовані мови - це мови низького рівня, що вимагають вказівки дрібних деталей процесу обробки даних. Мови ж високого рівня імітують природні мови, використовуючи деякі слова розмовної мови і загальноприйняті математичні символи. Ці мови більш зручні для людини.

Мови високого рівня поділяються на:

- процедурні (алгоритмічні) (Basic, Pascal, C та ін.), які призначені для однозначного опису алгоритмів;
- логічні або функційні (Prolog, Lisp та ін.), які орієнтовані не на розробку алгоритму розв'язання задачі, а на систематичний і формалізований опис завдання з тим, щоб рішення випливало з складеного опису;

- об'єктно орієнтовані (C ++, C #, Java, Object Pascal, та ін.), в основі яких лежить поняття об'єкта, який поєднує у собі дані і дії над ними.

В даному курсі вивчатиметься мова програмування 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 Framework. (Microsoft .NET — програмна технологія, запропонована фірмою Microsoft як платформа для створення як звичайних програм, так і веб-застосунків.)

Синтаксис C# близький до C++ і Java. На сьогодні C# є флагманською мовою корпорації Microsoft для програмування в середовищі Windows.

Для розробки програмного забезпечення на мові C# можна використовувати Microsoft Visual Studio або SharpDevelop.

Microsoft Visual Studio — серія продуктів фірми Microsoft, які включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Ці продукти дозволяють розробляти як консольні програми, так і програми з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-застосунки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight.

SharpDevelop — відкрите середовище розробки для C#, Visual Basic .NET, Boo, IronPython, IronRuby, F#, C++.

**Контрольні запитання**

1. Що таке алгоритмізація задачі?
2. Поясніть поняття «алгоритм».
3. Назвіть основні види алгоритмів та поясніть їх суть.
4. Назвіть основні властивості алгоритмів та поясніть їх.
5. Які основні способи опису алгоритмів Ви знаєте?
6. Назвіть та поясніть основні рівні мов програмування з огляду на ступень деталізації кроків алгоритму.

# 2. БАЗОВІ ЕЛЕМЕНТИ МОВИ ПРОГРАМУВАННЯ C#. ТИПИ ДАНИХ. ОПЕРАЦІЇ КОНСОЛЬНОГО ВВЕДЕННЯ ТА ВИВЕДЕННЯ.

## 2.1. Складові мови програмування

Програма записана певною мовою називається **початковим кодом** (вихідним текстом) і зазвичай зберігається в одному або декількох текстових файлах**.**

Для перекладу програми на мову низького рівня, зрозумілу комп'ютеру, існують спеціальні програми - **компілятори**. Результатом роботи компілятора (іншими словами, результатом процесу компіляції) є виконуваний код, який при програмуванні в середовищі Windows записується у файл з розширенням .exe.

Компілятор перед створенням виконуваного коду перевіряє синтаксис програми, що подається йому для перекладу. Пошуком же семантичних (змістових) помилок займається програміст в процесі тестування і налагодження своєї програми.

Для виконання програма не завжди повинна бути перекладена компілятором, існує також інший принцип: покрокове виконання програмних інструкцій інтерпретатором [1].

Мову програмування C# (як і будь-яку іншу мову) утворюють три складові: алфавіт, синтаксис і семантика.

**Алфавіт** - це фіксований для даної мови набір основних символів, тобто «Букв алфавіту», з яких повинен складатися будь-який текст на цій мові.

**Синтаксис** - це правила побудови фраз, що дозволяють визначити, правильно чи неправильно написана та чи інша фраза.

**Семантика** - правила тлумачення окремих мовних конструкцій.

**Алфавіт** мови C# складається з букв, цифр та спеціальних символів. В якості букв використовуються прописні букви латинського та національних алфавітів, наприклад від A до Z і рядкові від a до z, а також знак підкреслення _. В якості десяткових цифр використовуються арабські цифри від 0 до 9. Спеціальні символи в мові C# застосовуються для різних цілей: від організації тексту програми до визначення вказівок компілятору. Серед них .,!»№;%: і т.д.

**Лексема** - це мінімальна одиниця мови, що має самостійний сенс. Існують наступні види лексем:

- імена (ідентифікатори);
- ключові слова;
- знаки операцій;
- роздільники;
- літерали (константи).

З лексем складаються **вирази** й **оператори**. Вираз задає правило обчислення деякого значення. Наприклад, вираз a + b задає правило обчислення суми двох величин. Про це детальніше в наступній лекції.

**Ідентифікатори** використовуються як імена змінних, функцій і типів даних. Вони записуються за такими правилами**:**

- Ідентифікатори починаються з літери будь-якого алфавіту Unicode (знак підкреслення також є літерою).
- Ідентифікатор може складатися з літер і цифр (пробіли, точки та інші спеціальні символи при написанні ідентифікаторів недопустимі).
- Між двома ідентифікаторами повинен бути, принаймні, хоча б один роздільник (пробіл, табуляція, крапка і т.п.).

При написанні ідентифікаторів можна використовувати як прописні, так і малі літери. На відміну від інших мов програмування, компілятор мови C# розрізняє їх в записі ідентифікатора.

Для полегшення читання програми слід давати об'єктам осмислені імена, складені відповідно до певних правил. Зрозумілі й узгоджені між собою імена - основа гарного стилю програмування. Існує кілька видів так званих нотацій - угод про правила створення імен.

У нотації Паскаля кожне слово, яке складає ідентифікатор, починається з великої літери, наприклад, MaxLength, MyFuzzyShooshpanchik.

Угорська нотація (її запропонував угорець за національністю, співробітник компанії Microsoft) відрізняється від попередньої наявністю префіксу, відповідного типу величини, наприклад, iMaxLength, lpfnMyFuzzyShooshpanchik.

Згідно нотації Camel, з великої літери починається кожне слово, яке складає ідентифікатор, крім першого, наприклад, maxLength, myFuzzyShooshpanchik. Людині з багатою фантазією абрис імені може нагадувати верблюда, звідки і пішла назва цієї нотації.

Ще одна традиція - розділяти слова, складові ім'я, знаками підкреслення: max_length, my_fuzzy_shooshpanchik, при цьому всі складові частини починаються з малої літери.

У C # для іменування різних видів програмних об'єктів найчастіше використовуються дві нотації: Паскаля і Camel.

**Ключові слова** - це зарезервовані ідентифікатори, які мають спеціальне значення для компілятора. Їх можна використовувати тільки в тому сенсі, в якому вони визначені.

**Зауваження.** Ключові слова не можна використовувати в якості ідентифікаторів. Проте в мові C# можна використати ключове слово як ідентифікатор якщо перед ним поставити символ @. Наприклад @else.

**Знак операції** - це один або більше символів, що визначають дію над операндами. Усередині знаку операції пробіли не допускаються. Наприклад, у виразі a += b знак += є знаком операції, а a і b - операндами. Символи, складові знаків операцій, можуть бути як спеціальними, наприклад, &&, | і <, так і літерними, такими як as або new.

**Літералами або константними значеннями**, називають незмінні величини. У C# є логічні, цілі, речові, символьні і рядкові константи, а також константа null. Компілятор, виділивши константу в якості лексеми, відносить її до одного з типів даних по її зовнішньому вигляду. Програміст може задати тип константи і самостійно.

Опис і приклади літералів кожного типу наведено в табл. 2.1.

Таблиця 2.1 – літерали мови #C

| **Константа** | **Опис** | **Приклади** |
|---|---|---|
| Логічна | true (істина) або false (неправда) | true false |
| Ціла | *Десяткова*: послідовність десяткових цифр (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), за якою може слідувати суфікс (U, u, L, l, UL, Ul, uL, ul, LU, Lu, lU, lu). | 8 0 199226<br>8u 0Lu 199226L |
| | *Шістнадцяткова*: символи 0x, за якими слідують шістнадцятирічні цифри (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F), а за цифрами, у свою чергу, може слідувати суфікс (U, u, L, l, UL, Ul, uL, ul, LU, Lu, lU, lu). | 0xA 0x1B8 0x00FF<br>0xAU 0xlB8LU<br>0x00FFl |

| | | |
|---|---|---|
| Дійсна | *З фіксованою точкою*:<br>[цифри] [.] [цифри] [суфікс]<br>Суфікс - один з символів F, f, D, d, M,m<br><br>*З порядком:*<br>[цифри][.][цифри]{ E\|e}[+ \|-] [цифри] [суфікс]<br>Суфікс - один з символів F, f, D, d, M, m | 5.7 .001 35<br>5.7F .00ld<br>5F .00lf 35m<br><br>0.2E6 .lle+3 5E-10<br>0.2E6D .lle-3<br>5E10 |
| Символьна | Символ, взятий в апострофи | 'A' 'ю' '\0'<br>'\n' '\xF' '\x74'<br>'\uA81B' |
| Рядкова | Послідовність символів, взятих у лапки | "Це текст"<br>"tЗначення грам = \xF5 \n"<br>"Тут був \u0056\u0061"<br>"C: \\temp\\file1.txt"<br>@"C: \temp\file1.txt |
| Константа null | Посилання, яке не указує ні на який об'єкт | Null |

## 2.2. Коментарі

**Коментарі** призначені для запису пояснень до програми і формування документації. Компілятор коментарі ігнорує. У середині коментаря можна використовувати будь-які символи. У C# є два види коментарів: однорядкові і багаторядкові.

За допомогою подвійних косих (//) задаються коментарі до кінця рядка. А за допомогою пар символів /* та */ - багаторядкові коментарі.

## 2.3. Типи даних

Дані, з якими працює програма, зберігаються в оперативній пам'яті. Природно, що компілятору необхідно точно знати, скільки місця вони займають, як саме закодовані і які дії з ними можна виконувати. Все це задається при описі даних за допомогою типу.

**Тип даних** однозначно визначає:

- внутрішнє подання даних, а отже, і множину їх можливих значень;
- допустимі дії над даними (операції і функції).

Наприклад, цілі і дійсні числа, навіть якщо вони займають однаковий обсяг пам'яті, мають зовсім різні діапазони можливих значень; цілі числа можна множити один на одного, а, наприклад, символи - не можна.

Кожен вираз в програмі має певний тип. Величин, що не мають ніякого типу, не існує. Компілятор використовує інформацію про тип при перевірці допустимості описаних в програмі дій.

Система типів мови C# включає наступні дві категорії:

- типи значень;
- типи посилання.

У змінних типу значень зберігаються дані, а в змінних типу посилань – адреси (посилання) за якими зберігаються фактичні данні. Детальніше про те які класи містяться в цих категоріях на рис. 2.1.

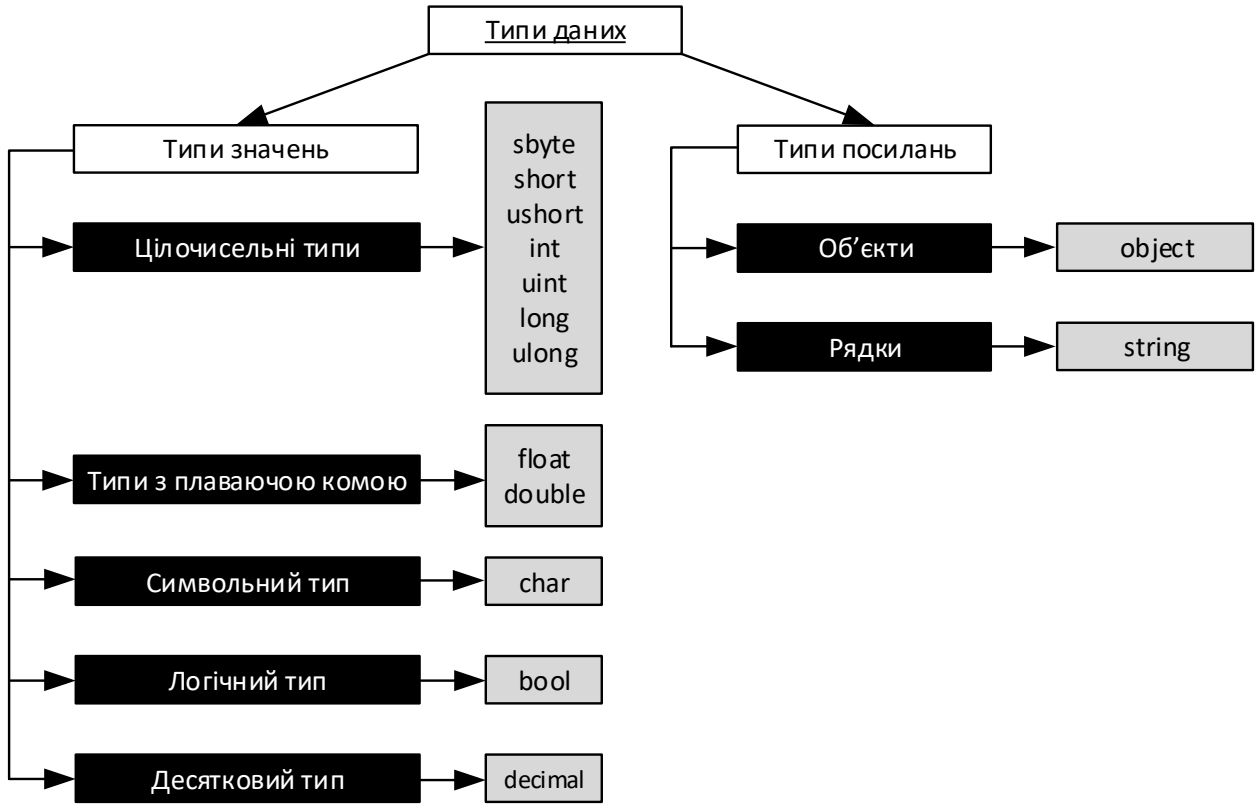


Рисунок 2.1 – Типи значень та типи посилань

Додаткову інформацію про типи даних та змінні C# можна отримати за посиланням [3].

Також типи даних поділяють на вбудовані і ті що задаються користувачем. Вбудовані типи C# наведені в табл. 2.2. Кожному з них відповідає клас або структура бібліотеки .NET.

Таблиця 2.2 - Вбудовані типи C#

| **Ключове слово** | **Тип .NET** | **Діапазон значень** | **Опис** | **Розмір, байт** |
|---|---|---|---|---|
| boo1 | Boolean | true, false | | |
| sbyte | SByte | Від -128 до 127 | Зі знаком | 1 |
| byte | Byte | Від 0 до 255 | Без знаку | 1 |
| short | Intl6 | Від –32768 до 32767 | Зі знаком | 2 |
| ushort | UIntl6 | Від 0 до 65535 | Без знаку | 2 |
| int | Int32 | Від $-2^{31}$ до $2^{31} – 1$ | Зі знаком | 4 |
| uint | UInt32 | Від 0 до $2^{32} – 1$ | Без знаку | 4 |
| long | Int64 | Від $-2^{63}$ до $2^{63} – 1$ | Зі знаком | 8 |
| ulong | UInt64 | Від 0 до $2^{64} – 1$ | Без знаку | 8 |
| char | Char | Від U+0000 до U+ffff | Unicode-символ | 2 |
| float | Single | Від $1.5•10^{-45}$ до $3.4•10^{38}$ | 7 цифр | 4 |
| double | Double | Від $5.0•10^{-324}$ до $1.7•10^{308}$ | 15-16 цифр | 8 |
| decimal | Decimal | Від $1.0•10^{-28}$ до $7.9•10^{28}$ | 28-29 цифр | 16 |
| string | String | Рядок з символів Unicode | Довжина обмежена об'ємом доступної пам'яті | |
| object | Object | Можна зберігати все що завгодно | Загальний предок | |

## 2.4. Змінні і константи

Основними даними, що обробляються програмою є змінні і константи.

**Змінні** – це дані, які можуть змінювати свої значення в процесі виконання програми.

**Константи** – це змінні значення яких задаються при першій згадці, та більше не змінюються в процесі виконання програми.

Всі змінні в мові C# повинні бути описані явно. Це означає, що, по-перше, на початку кожної програми або функції потрібно навести список імен (ідентифікаторів) всіх використовуваних змінних, а по-друге, вказати тип кожної з них.

Опис змінної:

```
{<тип>|var} <ім'я змінної> [=<початкове значення>],
    [<ім'я змінної2> [=<початкове значення2>]];
```

Використання **var** – це **неявна типізація**. Тип неявно типізованих локальних змінних визначає компілятор в залежності від присвоєного початкового значення.

**Ім'я змінної** – є ідентифікатором – тобто будь-яка послідовність великих і малих літер, цифр і символу підкреслення '_'.

Подібно будь-якій іншій мові програмування, в C# визначені ключові слова для фундаментальних типів даних, які застосовуються для опису змінних. Однак на відміну від інших мов програмування, в C# ці ключові слова являють собою щось більше, ніж просто розпізнавані компілятором лексеми. По суті, вони є скороченими позначеннями повноцінних типів з простору імен System.

Простори імен представляють собою спосіб організації різних типів присутніх в програмах C#. Їх можна порівняти з папкою в файловій системі. Подібно до папок, простори імен визначають для класів унікальні повні імена. **Простори імен -** призначені для розмежування різних множин ідентифікаторів і попередження конфліктів між їхніми іменами.

Для оголошення локальної змінної необхідно вказати тип даних і відразу за ним ім'я змінної.

Приклади:

```
int  i, j=0;
double d = 3.7;
```

Можна заборонити змінювати значення змінної, задавши при її описі ключове слово **const**, наприклад:

```
const int b = 1;
```

```
const float x = 0.1, y = 0.1f;  // const для x та y
```

Такі величини називають **константами**. Вони застосовуються для того, щоб замість значень констант можна було використовувати в програмі їх імена. Це робить програму зрозумілішою і полегшує внесення до неї змін, оскільки змінити значення досить тільки в одному місці програми.

О**бласть дії змінної або константи** - це область програми, де можна використовувати змінну, вона починається в точці її опису і триває до кінця блоку, усередині якого вона описана.

**Блок** - це код, укладений у фігурні дужки. Основне призначення блоку - угруповання операторів. У C # будь-яка змінна описана усередині якого-небудь блоку: класу, методу чи блоку всередині методу.

Ім'я змінної повинно бути унікальним в області її дії. Область дії поширюється на вкладені в метод блоки, з цього випливає, що у вкладеному блоці не можна оголосити змінну з таким же ім'ям, що і в охоплює його.

Використання локальної змінної до присвоювання їй початкового значення призведе до помилки на етапі компіляції.

**Числові типи .NET** підтримують властивості **MaxValue** та **MinValue**, які надають інформацію про діапазоні значень, що зберігаються в конкретному типі. Крім властивостей MinValue і MaxValue, кожен числовий тип може визначати додаткові корисні члени.

Розглянемо структуру найпростішої програми на C#:

```
using System;
namespace MyFirstProject {
    class Program {
        static void Main(){
        }
    }
}
```

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 замовчуванням іменем 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. **Main()** – метод, точка входу в програму, з якої починається виконання коду додатку. Метод з ім'ям Main в програмі може бути лише один.

### 2.5. Ввід та вивід за допомогою System.Console

Тепер розглянемо, як організувати ввід та вивід. В консольному додатку найпростіше скористатися методами класу **Console** простору імен **System.** Основні методи цього класу для читання запису наступні:

- **Console.Read()** – зчитати окремий символ з клавіатури. Повертає число int, т.т. його результат потрібно привести до типу char, щоб отримати символ.

```
char ch = (char)Console.Read();
```

- **Console.ReadLine()** – читання рядку символів.

```
string st = Console.ReadLine();
```

Для того, щоб перетворити отриманий рядок у число, можна скористатися об'єктом **Convert** простору імен **System.**

```
double d = Convert.ToDouble(Console.ReadLine());
int i d = Convert.ToInt32(Console.ReadLine());
```

- **Console.Write()** та **Console.WriteLine()** – використовуються для організації форматованого виводу на екран. Причому **Write** - виводить рядок символів на екран, а виводить **WriteLine** рядок символів з переводом виводу на новий рядок.

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

Щоб вивести на екран значення якоїсь змінної, можна скористатися форматованим виводом. Наприклад:

```
int i = 1234;
Console.WriteLine("i = {0,5}",i);
```

Де `{0,5}` форматує значення виразу або змінної, що виводиться. Перше число у фігурних дужках відповідає за порядковий номер змінної у списку змінних перерахованих через кому (***нумерація починається з 0***).

Наступне число – відповідає за кількість позицій на екрані виділених для відображення значення змінної (в наведеному прикладі - 5).

Для виводу чисел з плаваючою крапкою (комою), можна вказати і кількість цифр після крапки. Наприклад:

```
double d = 1234.12345;
Console.WriteLine("d = {0,7:f3}",d);
```

Виведе на екран: `d = 1234,123`. Тут `f` – символ форматування, вказує що число треба виводити, як дійсне з плаваючою крапкою, а число 3 – що дробова частина числа повинна містити три цифри.

Таблиця 2.3 - Основні символів форматування числових даних

| **Символи форматування** | **Опис** |
|---|---|
| `F або f` | Для виводу чисел з фіксованою точністю |
| `E або e` | Для виводу чисел в експоненційному форматі |
| `P або p` | Для виводу процентів |
| `N або n` | Для виводу чисел з розділеними розрядами |
| `C або c` | Для виводу значень чисел в національних валютах |
| `D або d` | Для виводу чисел в десятковому форматі |
| `G або g` | Для виводу чисел з фіксованою точністю або в експоненційному форматі |
| `X або x` | Для виводу цілих шістнадцяткових чисел |

## 2.6. Форматований вивід

Більш строго опишемо форматований вивід. В середині рядка, що виводиться можна використовувати:

**{ index[,alignment][:formatString]}**

Обов'язковий компонент **index** – це число, що визначає порядковий номер об'єкту зі списку після закриваючих текстовий літерал лапок; індексація елементів починається з 0.

Необов'язковий компонент **alignment** – ціле число зі знаком, що використовується для задання бажаної ширини форматування. Якщо модуль значення alignment менше довжини рядка, що вставляється, то alignment – не використовується і рядок вставляється повністю. Якщо модуль alignment більше - рядок вирівнюється по правому краю для додатного значення alignment, і по лівому краю, якщо alignment – від'ємне число. При вирівнюванні рядок, що вставляється доповнюється пробілами. При використанні alignment перед ним необхідно поставити кому.

Необов'язковий компонент **formatString** – це рядок формату, що задає тип форматування (детальніше в таблиці 2.3).

Приклад:

```
double d = 1234.12345;
Console.WriteLine("d = {0,7:f3}",d);
```

Виведе на екран: `1234.123`

**Контрольні запитання**

1. Поясніть поняття «початковий код» та «компілятор».
2. Назвіть складові алфавіту мови C#.
3. Що таке ідентифікатор? Назвіть правила їх запису.
4. Поясніть поняття «літерали» та «константні значення».
5. Принцип роботи з коментарями.
6. Що таке «тип даних». Назвіть та поясніть декілька вбудованих типів даних C#.
7. Поясніть принцип роботи зі змінними та константами. Наведіть приклад опису змінних.
8. Назвіть клас та основні методи, що використовуються для введення та виведення інформації.
9. Поясніть різницю між **`Write`**`()` та **`WriteLine`**`().`
10. Назвіть основні символи форматування числових даних.
11. Поясніть, за яким принципом організовується форматований вивід результату.

# 3. ВИРАЗИ ТА АРИФМЕТИЧНІ ОПЕРАТОРИ

## 3.1. Вирази C#

**Вираз** – послідовність з одного або декількох операндів та від нуля до декількох операторів, яку можна обчислити, отримавши в результаті одне значення, об'єкт або простір імен. Вираз може складатися з літералу, виклику метода, оператору та його операндів, а також з простого імені. В якості простого імені може використовуватися ім'я змінної, члену типу, параметру метода, простору імен або типу.

У виразі можуть використовуватись оператори, які в свою чергу в якості параметрів використовують інші вирази, або виклики методів, для визначення параметрів яких викликаються інші методи. Таким чином вирази бувають прості та складені.

**Операндами** можуть бути: літерали, поля, змінні та константи, вирази і т. п.

Наприклад, a + 2 - це вираз, в якому + є знаком операції, а a і 2 - операндами. Пробіл всередині знаку операції, що складається з декількох символів, не допускаються.

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

## 3.2. Прості оператори C#

В англійській мові для позначення операторів використовують два слова **`operator`** і **`statement`**. Перше operator - використовують для позначення простих операторів, які в нашій літературі по програмуванню називають операціями. Наприклад: +, *, -, / і т.д. Слово statement використовують для позначення всіх операторів вцілому. Тобто прості оператори (operator) є підмножиною операторів(statement).

Найпоширеніші прості оператори C# наведено в табл. 3.1.

Таблиця 3.1 – Основні оператори C#

| Категорія | Знак операції | Назва |
|---|---|---|
| Первинні | x.y | Доступ до складової об'єкту |
| | x () | Виклик методу або делегата |
| | x [] | Доступ до елементу за індексом |
| | x++ | Постфіксний інкремент |
| | x-- | Постфіксний декремент |
| | new | Створення об'єкту |

| | | |
|---|---|---|
| | typeof | Отримання типу |
| | checked | Вмикає перевірку на переповнення для цілочисельних операцій. |
| | unchecked | Вимикає перевірку переповнення для цілочисельних операцій. Прийнято за замовчуванням. |
| Унарні | + | Унарний плюс |
| | - | Унарний мінус |
| | ! | Логічне заперечення |
| | ~ | Порозрядне заперечення |
| | ++x | Префіксний інкремент |
| | --x | Префіксний декремент |
| | (тип) x | Приведення типів |
| Мультиплікативні | * | Множення |
| | / | Ділення |
| | % | Залишок від ділення |
| Адитивні | + | Додавання |
| | - | Віднімання |
| Зсуву | << | Зсув вліво |
| | >> | Зсув вправо |
| | | |
| Відношення та операції перевірки типу | < | Менше |
| | > | Більше |
| | <= | Менше або дорівнює |
| | >= | Більше або дорівнює |
| | is | Перевірка приналежності типу |
| | as | Перетворення типу |
| Перевірки на рівність | == | дорівнює |
| | != | Не дорівнює |
| Порозрядні логічні | & | Поразрядна кон'юнкція (І) |
| | ^ | Виключна диз'юнкція або додавання за модулем два(XOR) |
| | \| | Поразрядна диз'юнкція (АБО) |
| Умовні логічні | && | Логічне І |
| | \|\| | Логічне АБО |
| Умовний | ?: | Умовна операція |
| Присвоювання | = | Присвоєння |
| | *= | Множення з присвоєнням |
| | /= | Ділення з присвоєнням |
| | %= | Залишок відділення 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 |

Оператори у виразі виконуються в певному порядку відповідно до пріоритетів, як і в математиці. В табл. 3.1 оператори розташовані в порядку спадання пріоритетів, тобто чим вище оператор в таблиці тим раніше він має виконуватися. Для задання свого порядку виконання використовуються круглі дужки – ().

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.

### 3.3. Інкремент і декремент

Оператори інкремента **(++)** і декремента **(--)**, що називають також операціями збільшення та зменшення на одиницю, мають дві форми запису - **префіксний**, коли знак операції записується перед операндом, і **постфіксними**.

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

Приклади:

```
int i = 1;
int j;
j = i++;  //після обчислень і рівне 2, а j=1

int i = 1;
int j;
j = ++i;  //після обчислень і рівне 2 і j=2
```

### 3.4. Операції заперечення

**Арифметичне заперечення** (унарний мінус -) змінює знак операнду на протилежний.

```
int i = 5;
int j = -i; //після обчислень і рівне 5 і j=-5
```

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

**Порозрядне заперечення** (~), часто зване побітовим, інвертує кожен розряд в двійковому поданні операнду типу int, uint, long або ulong.

**~01011100 == 10100011**

### 3.5. Явне перетворення типу

Операція використовується, як і випливає з її назви, для явного перетворення величини з одного типу в інший. Це потрібно в тому випадку, коли неявного перетворення не існує. При перетворенні з довшого типу в більш короткий можлива втрата інформації, якщо початкове значення виходить за межі діапазону результуючого типу .

Формат операції:

**(<Тип>) <вираз>**

Тут тип - це ім'я того типу, в який здійснюється перетворення, а вираз найчастіше представляє собою ім'я змінної, наприклад:

```
long b = 300;
int a = (int) b;   // дані не втрачаються
byte d = (byte) a; // дані губляться
```

### 3.6. Множення, ділення і залишок від ділення

**Операція множення** (*) повертає результат перемноження двох операндів. Стандартна операція множення визначена для типів int, uint, long, ulong, float, double і decimal.

**Операція ділення** (/) обчислює частка від ділення першого операнду на другий. Стандартна операція ділення визначена для типів int, uint, long, ulong, float, double і decimal. Проте слід зазначити, що результат ділення залежить від типу операндів, для цілих чисел виконується цілочисельне ділення і результат буде цілим числом, а для дійсних виконується ділення з точністю дійсного типу даних і результатом буде дійсне число.

```
int a = 3 / 2;          // a рівне 1
double d = 3.0 / 2.0; // d рівне 1.5
```

**Операція залишку від ділення (%)** також інтерпретується по-різному для цілих, речових і фінансових величин.

```
int a = 5 % 2;          // a рівне 1
double d = 1.9 % 1.0; // d рівне 0.9
```

### 3.7. Додавання і віднімання

**Операція додавання** (+) повертає суму двох операндів. Стандартна операція додавання визначена для типів int, u1nt, long, ulong, float, double і decimal.

**Операція віднімання** (-) повертає різницю двох операндів. Стандартна операція віднімання визначена для типів int, uint, long, ulong, float, double і decimal.

### 3.8. Операції відношення та перевірки на рівність

**Операції відношення (<, <=, >, >=, ==, !==)** порівнюють перший операнд з другим. Операнди повинні бути арифметичного типу. Результат операції - логічного типу, дорівнює true або false.

### 3.9. Умовні логічні операції

**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 операнду мають значення true. Результат операції логічне АБО має значення true, якщо хоча б один з операндів має значення true.

### 3.10. Умовний тернарний оператор

**Умовний оператор (? :)** - тернарний, тобто має три операнди. Його формат: операнд_1? операнд_2: операнд З

Перший операнд - вираз, для якого існує неявне перетворення до логічного типу. Якщо результат обчислення першого операнду дорівнює true, то результатом умовної операції буде значення другого операнду, інакше - третього операнду. Обчислюється завжди або другий операнд, або третій.

Умовну операцію часто використовують замість умовного оператора if (він розглядається в наступному розділі) для скорочення тексту програми.

Приклад:

```
class Program {
    static void Main() {
        int a = 3;
        int b = 4;
        int c = a > b ? a + 1 : a - 5; // В с буде -2
    }
}
```

### 3.11. Операції присвоювання

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

**<змінна> = <вираз>;**

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 знаку операції. Те, що раніше зберігалося в цій області пам'яті, природно, втрачається. Схематично це можна уявити собі так:

**змінна ← вираз**

Коли ж перед знаком присвоювання стоїть знак іншої операції, це означає, що спочатку треба виконати цю операцію для значення змінної та виразу, а потім присвоїти це значення змінній. Тобто:

**a += b;** еквівалентно **a = a + b;**

### 3.12. Математичні функції - клас Math

Для виконання більш складних математичних обчислень використовуються поля та методи класу **Math** в просторі імен **System**. За допомогою методів цього класу можна обчислити:

- тригонометричні функції: Sin, Cos, Tan;
- обернені тригонометричні функції: ASin, ACos, ATan, ATan2;
- гіперболічні функції: Tanh, Sinh, Cosh;
- експоненту і логарифмічні функції: Exp, Log, Log10;
- модуль (абсолютну величину), квадратний корінь, знак: Abs, Sqrt, Sign;
- округлення: Ceiling, Floor, Round;
- мінімум, максимум: Min, Max;
- степінь: Pow;
- залишок від ділення: DivRem.

Крім того, у класу є два корисних **поля**, що містять числа **π** та **e**.

Опис основних членів класу **Math** наведено в табл. 3.2.

Таблиця 3.2 - Основні поля і статичні методи класу Math

| Ім'я | Опис | Результат | Пояснення |
|---|---|---|---|
| Abs | Модуль | Перевантажений | \| x \| записується як Abs (x) |
| Acos | Арккосинус | double | Acos (double x) |
| Asin | Арксинус | double | Asin (double x) |
| Atan | Арктангенс | double | Atan (double x) |
| Atan2 | Арктангенс | double | Atan2 (double x, double y) - кут, тангенс якого є результат ділення y на x |
| Ceiling | Округлення до більшого цілого | double | Ceiling (double x) |
| Cos | Косинус | double | Cos (double x) |
| Cosh | Гіперболічний косинус | double | Cosh (double x) |
| DivRem | Частка двох int, з залишком у третьому вихідному параметрі | | DivRem (x, y, rem) |
| E | Поле з основою натурального логарифма (число e) | double | 2,71828182845905 |
| Exp | Експонента | double | $e^x$ записується як Exp (x) |
| Floor | Округлення до меншого цілого | double | Floor (double x) |
| IEEE Remainder | Залишок від ділення | double | IEEERemainder (double x, double y) |
| Log | Натуральний логарифм | double | $\text{Log}_e$ x записується як Log(x) або Log(x,y) |
| Log10 | Десятковий логарифм | double | $\log_{10}$ x записується як Log10 (x) |
| Max | Максимум з двох чисел | Перевантажений | Max (x, y) |
| Min | Мінімум з двох чисел | Перевантажений | Min (x, y) |
| PI | Значення числа π | double | 3,14159265358979 |
| Pow | Степінь | double | $x^y$ записується як Pow (x, y) |
| Round | Округлення | Перевантажений | Round (3.1) дасть в результаті 3, Round (3.8) дасть в результаті 4 |
| Sign | Знак числа | int | Аргументи перевантажені |
| Sin | Синус | double | Sin (double x) |
| Sinh | Гіперболічний синус | double | Sinh (double x) |
| Sqrt | Квадратний корінь | double | записується як Sqrt (x) |
| Tan | Тангенс | double | Tan (double x) |
| Tanh | Гіперболічний тангенс | double | Tanh (double x) |
| Truncate | Ціла частина | Перевантажений | Ціла частина дійсного числа |

Приклад 1:

```
using System;

namespace RoundExample {
    class Program {
        static void Main() {
            double a = 1234.5678;
            Console.WriteLine(Math.Round(a));
            Console.WriteLine(Math.Round(a,2));
            Console.ReadKey();
        }
    }
}
```

Виведе на екран

```
1235
1234,57
```

Приклад 2:

Скласти програму для обчислення змінної за заданою формулою:

$$\Delta = \frac{tg\alpha - \ln^2 \beta}{\gamma}$$

1. Ознайомившись з класом Math, знайдемо методи для знаходження тангенсу та натурального логарифму – Math.Tan та Math.Log.

2. Програми для обчислення цього виразу виглядатиме так:

```
class Program{
    static void Main(string[] args)     {
        double alpha = 4.5;
        double beta = 3.2;
        double gamma = 2.3;
        double log = System.Math.Log(beta);
        double delta =(System.Math.Tan(alpha)-
               log * log)/gamma;
    }
}
```

**Контрольні запитання**

1. Що таке вираз? Якими вони бувають?
2. Наведіть класифікацію операцій за кількістю операндів.
3. Назвіть та поясніть основні оператори C#.
4. Поясніть поняття «інкремент» та «декремент». Які форми їх запису Ви знаєте?
5. Які операції заперечення Ви знаєте? Поясніть їх.
6. Поясніть принцип роботи з явним перетворенням типу.
7. Поясніть принцип роботи з операціями множення, ділення, залишку від ділення, додавання, віднімання, присвоювання, відношення та перевірки на рівність.
8. Які умовні логічні операції Ви знаєте? Поясніть їх.
9. Поясніть принцип роботи з умовним тернарним оператором.
10. Назвіть основні поля та статичні методи класу **`Math`**.

# 4. ПРИВЕДЕННЯ І ПЕРЕТВОРЕННЯ ТИПІВ

## 4.1. Особливості перетворення базових типів даних

При розгляді типів даних вказувалося, які значення може мати той чи інший тип і скільки байт пам'яті він може займати. Застосуємо операцію додавання до даних різних типів:

```
byte a = 4;
int b = a + 70;
```

Результатом операції цілком справедливо є число 74, як і очікується.

Але тепер спробуємо застосувати складання до двох змінних типу byte:

```
byte a = 4;
byte b = a + 70;    // помилка
```

Тут змінився тільки тип змінної, яка отримує результат складання - з int на byte. Однак при спробі скомпілювати програму ми отримаємо помилку на етапі компіляції. І якщо ми працюємо в Visual Studio, середовище підкреслить другий рядок червоною хвилястою лінією, вказуючи, що в ній помилка.

При операціях ми повинні враховувати діапазон значень, які може зберігати той чи інший тип. Але в даному випадку число 74, яке ми очікуємо отримати, цілком укладається в діапазон значень типу byte, проте ми отримуємо помилку.

Справа в тому, що операція додавання (та й віднімання) повертає значення типу int, якщо в операції беруть участь цілочисельні типи даних з розрядністю менше або дорівнює int (тобто типи byte, short, int). Тому результатом операції a + 70 буде об'єкт, який має довжину в пам'яті 4 байти. Потім цей об'єкт ми намагаємося привласнити змінній b, яка має тип byte і в пам'яті займає 1 байт.

І щоб вийти з цієї ситуації, необхідно застосувати операцію перетворення типів:

```
byte a = 4;
byte b = (byte) (a + 70);
```

Операція перетворення типів передбачає вказівку в дужках того типу, до якого треба перетворити значення.

## 4.2. Види перетворень

В C # можна виконувати наступні види перетворень:

- неявні перетворення;
- явні перетворення (приведення);
- перетворення за допомогою допоміжних класів;
- перетворення задані користувачем.

## 4.3. Неявні перетворення

Для **неявних перетворень** спеціальний синтаксис не потрібен, вони виконуються, коли дані одного типу присвоюються змінній іншого типу. Неявне перетворення типів відбувається автоматично при виконанні наступних умов:

- обидва типи сумісні;
- діапазон представлення чисел цільового типу ширше, ніж у вихідного типу

Якщо обидві ці умови виконуються, то відбувається розширююче перетворення. Наприклад, тип int досить великий, щоб вміщати в себе всі значення типу byte, а крім того, обидва типи, int і byte, є сумісними цілочисельними типами, і тому для них цілком можливо неявне перетворення.

Загальні правила неявного перетворення наведено на рис. 4.1.

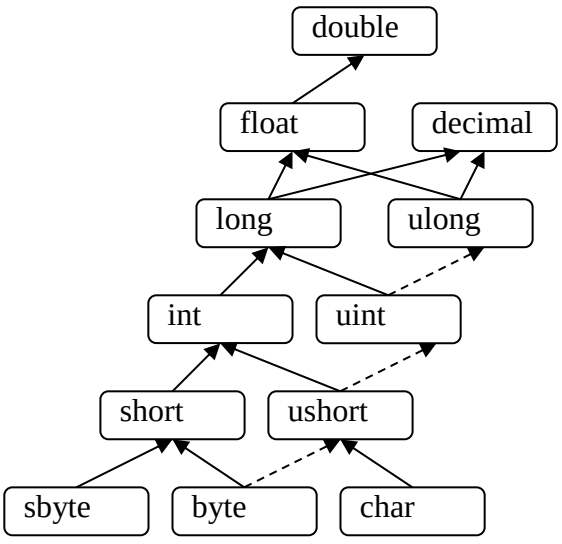


Рисунок 4.1 - Неявні перетворення числових типів

Якщо один з операндів має тип, зображений на нижчому рівні, ніж інший, то він буде приводитися до типу іншого операнду за наявності шляху між ними. Якщо шляху немає, виникає помилка компіляції. Якщо шляхів

декілька, вибирається найбільш короткий, такий, що не містить пунктирних ліній. Перетворення виконується не послідовно, а безпосередньо з початкового типу в результуючий[6].

Зазначимо, що неявного перетворення дійсних типів float і double в decimal не існує.

### 4.4. Явні перетворення (приведення)

**Явні перетворення** або приведення - вимагають використання оператора приведення типів. Приведення потрібно, якщо в ході перетворення дані можуть бути втрачені або перетворення може завершитися зі збоєм з інших причин. Типовими прикладами є числове перетворення в тип з меншою точністю або меншим діапазоном значень або перетворення екземпляру базового класу в похідний клас.

Приведення - це спосіб явно вказати компілятору, що необхідно виконати перетворення і що вам відомо, що може відбутися втрата даних[4]. Щоб виконати приведення: в круглих дужках перед виразом або змінною вкажіть тип, в який проводиться перетворення. У наступній програмі виконується приведення типу double в int. Програма не компілюватиметься без приведення:

```
class Test{
    static void Main(){
        double x = 1234.7;
        int a;
        // Перетворення double в int.
        a = (int)x;
        System.Console.WriteLine(a);
    }
}
// Результат на екрані: 1234
```

### 4.5. Перетворення з використанням допоміжних класів

Для перетворення між несумісними типами, наприклад рядками та цілими можна використовувати класи **System.Convert** і **System.BitConverter**, а також методи **Parse** вбудованих числових типів, такі як **Int32.Parse()**.

В таблиці 4.1 наведено основні методи перетворення класу **System.Convert**.

Таблиця 4.1 – Методи перетворення класу `Convert`

| Назва методу | Тип результату |
|---|---|
| ToBoolean | bool |
| ToByte | byte |
| ToChar | char |
| ToDouble | double |
| ToSingle | float |
| ToInt32 | int |
| ToInt64 | long |
| ToString | string |

Приклад перетворень типів за допомогою класу **`Convert`**:

```
using System;

namespace ConvertExample {
    class Program {
        static void Main() {
            Console.WriteLine("Введіть дійсні значення a та
b:");
            // Читаємо перший рядок
            string str = Console.ReadLine();
            // Конвертуємо рядок в дійсне число
            double a = Convert.ToDouble(str);
            // Читаємо другий рядок і одразу його конвертуємо
            double b = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine("Введіть ціле число c:");
            // Читаємо рядок і конвертуємо в ціле
            int c = Convert.ToInt32(Console.ReadLine());
            // Обчислюємо вираз
            double d = (a + b) / c;
            // Виводимо результат з точністю до третього знаку
            Console.WriteLine("({0}+{1})/{2}={3,0:f3}", a, b, c,
d);
            // Зупинка програми
            Console.ReadLine();
        }
    }
```

```
}
```

Результат роботи програми:

```
Введіть дійсні значення a та b:
2,33
4,55
Введіть ціле число c:
5
(2,33+4,55)/5=1,376
```

Розв'яжемо аналогічну задачу з використанням методів **Parse** для числових типів **double** та **int**.

```
using System;

namespace ParseExample {
    class Program {
        static void Main() {
            Console.WriteLine("Введіть дійсні значення a та
b:");
            // Читаємо перший рядок і конвертуємо в дійсне
            double a = double.Parse(Console.ReadLine());
            // Читаємо другий рядок і конвертуємо в дійсне
            double b = double.Parse(Console.ReadLine());
            Console.WriteLine("Введіть ціле число c:");
            // Читаємо рядок і конвертуємо в ціле
            int c = int.Parse(Console.ReadLine());
            // Обчислюємо вираз
            double d = (a + b) / c;
            // Виводимо результат з точністю до третього знаку
            Console.WriteLine("({0}+{1})/{2}={3,0:f3}", a, b, c,
d);
            // Зупинка програми
            Console.ReadLine();
        }
    }
}
```

Перевіримо роботу програми на тих же вхідних даних:

```
Введіть дійсні значення a та b:
2,33
4,55
Введіть ціле число c:
5
```

```
(2,33+4,55)/5=1,376
```

На однакових вхідних даних отримали однаковий результат, це означає що програми – еквівалентні.

***Зауваження***: При конвертації рядків в дійсні числа слід враховувати, що використання роздільника **крапка** або **кома** - залежить від налаштувань операційної системи комп'ютера на якому виконується програма. Якщо роздільник вказано невірно, то виникне помилка конвертації під час виконання програми.

***Зауваження 2***: При використанні числових лексем в тексті програми – завжди слід використовувати крапку в якості роздільника.

**Контрольні запитання**

1. Назвіть основні види перетворення базових типів даних.
2. Поясніть особливості роботи з неявним перетворенням типів.
3. Поясніть принцип роботи з явним перетворенням (приведенням) типів.
4. В якому випадку використовується перетворення з використанням допоміжних класів? Поясніть принцип роботи з даним видом перетворення.
5. Назвіть основні методи перетворення класу **`System.Convert`**.

# 5. СИМВОЛЬНИЙ ТИП ДАНИХ. ТЕКСТОВІ РЯДКИ. РОБОТА ІЗ РЯДКОВИМИ ДАНИМИ

Досить часто в програмуванні виникають задачі пов'язані з обробкою текстової інформації, для вирішення цих задач в середовищі .Net передбачено два базові типи даних: символьний тип даних - **System.Char** та рядки - **System.String**.

## 5.1. Символьний тип даних

Символьний тип char або Char - є структурою і призначений для зберігання символів в кодуванні Unicode (UTF-16). В Unicode кожен символ належить до певної категорії: цифри, літери, роздільники, тощо. Деякі символи можуть належати до декількох категорій одночасно. Для визначення виду та категорії символів та для переведення символів з верхнього регістру в нижній та навпаки визначені спеціальні статичні методи для типу. Основні з цих методи наведено в табл. 5.1.

Таблиця 5.1 - Основні методи класу System.Char

| **Метод** | **Пояснення** |
|---|---|
| CompareTo(Char) | Порівнює поточний символ з заданим і повертає від'ємне число, якщо поточний символ знаходиться в кодовій таблиці раніше, додатне – якщо пізніше. |
| GetNumericValue(Char) | Перетворює числовий символ в дійсне число типу double, інакше повертає -1.0 |
| GetUnicodeCategory(Char) | Повертає категорію символу<br>Наприклад:<br>Console.WriteLine(Char.GetUnicodeCategory('w'));<br>// Виведе: "LowercaseLetter"<br>Console.WriteLine(Char.GetUnicodeCategory('4'));<br><br>// Виведе: "DecimalDigitNumber" |
| IsControl(Char) | Повертає true, якщо символ є керуючим |
| IsDigit(Char) | Повертає true, якщо символ є десяткової цифрою |
| IsLetter(Char) | Повертає true, якщо символ є буквою |
| IsLetterOrDigit(Char) | Повертає true, якщо символ є буквою або цифрою |
| IsLower(Char) | Повертає true, якщо символ заданий у нижньому регістрі |

| | |
|---|---|
| IsNumber(Char) | Повертає true, якщо символ є числом (десятковим або шістнадцятковим) |
| IsPunctuation(Char) | Повертає true, якщо символ є знаком пунктуації |
| IsSeparator(Char) | Повертає true, якщо символ є роздільником |
| IsUpper(Char) | Повертає true, якщо символ записаний у верхньому регістрі |
| IsWhiteSpace(Char) | Повертає true, якщо символ є пробільним (пробіл, символ нового рядка та символ повернення каретки) |
| Parse(String) | Перетворить рядок в символ (рядок повинен складатися з одного символу) |
| ToLower(Char) | Перетворює символ в нижній регістр |
| ToUpper(Char) | Перетворює символ у верхній регістр |

Більш детально про цю структуру можна прочитати за посиланням [7] .

В якості прикладу використання методів символьний типу char розглянемо задачу: Ввести з клавіатури два символи і перевірити чи буде перший з них цифрою, а другий літерою.

```
using System;

namespace CharExample {
  class Program {
    static void Main() {
      Console.WriteLine(
        "Введіть два символи 1й цифру, 2й букву"
      );
      // Для перетворення рядка в число – метод Parse
      char ch1 = char.Parse(Console.ReadLine());
      char ch2 = char.Parse(Console.ReadLine());
      // Для цифри - метод IsDigit, для букви IsLetter
      Console.WriteLine(
        char.IsDigit(ch1) && char.IsLetter(ch2)?"Вірно":"Хибно"
      );
      Console.ReadKey();
    }
  }
}
```

Результат роботи цієї програми може бути наступним:

```
Введіть два символи 1й цифру, 2й букву
1
A
Вірно
```

### 5.2. Рядки типу string

Клас String – тип даних для зберігання текстової інформації. Тип даних String – це послідовність, що не містить жодного або довільну кількість символів Юнікоду в кодуванні UTF-16, тобто по 2 байти на кожен символ. В C# ключове слово string є псевдонімом для String. Тому string і String – еквівалентні[5].

Максимальний розмір об'єкту String в пам'яті складає 2 ГБ, близько 1 000 000 000символів[8].

Для рядків визначені такі операції:

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

Не дивлячись на те, що `String` – об'єктний тип, оператори == та != визначені для порівняння вмісту об'єктів типу рядок, а не посилань як для інших класів. Наприклад:

```
using System;
using System.Collections.Generic;
using System.Text;

class Program {
    static void Main(string[] args) {
        string a = "hello";
        string b = "h";
        // Append to contents of 'b'
        b += "ello";
        Console.WriteLine(a == b);
        Console.WriteLine((object)a == (object)b);
        Console.ReadKey();
    }
}
```

Виведе на екран:

```
True
False
```

Що показує, те що вміст рядків однаковий, хоча об’єкти різні.

Для рядків `String` визначено оператори `+` та `+=` для додавання. Проте слід зазначити, що ці об’єкти являються незмінними: після створення їх неможливо змінити. Все методи `String` і оператори C#, які, начебто змінюють рядок, насправді повертають в результаті новий об’єкт-рядок[5].

```
using System;

class Program {
    static void Main(string[] args) {
        string a, b = "h";
        a = b;      //a i b посилаються на один об'єкт
        Console.WriteLine((object)a == (object)b);
        b += "ello";    //тепер це різні об'єкти
        Console.WriteLine((object)a == (object)b);
        Console.WriteLine("a==" + a);
        Console.WriteLine("b==" + b);
        Console.ReadKey();
    }
}
```

На екрані буде:

```
True
False
a==h
b==hello
```

Тобто спочатку посилання `a` і `b` були на один об’єкт, а після виконання оператору `+=` змінна `b` стала вказувати на новий об’єкт.

Так само, і оператор [] служить тільки для доступу для читання окремих символів, та не може використовуватись для зміни значень.

```
using System;
class Program {
    static void Main(string[] args) {
        string a = "hello";
        Console.WriteLine(a[0]);
        Console.WriteLine(a[1]);
        Console.WriteLine(a[4]);
        Console.ReadKey();
    }
}
```

Результат:

```
h
e
o
```

### 5.3. Керуючі послідовності

Рядкові константи можна задавати в двох варіантах:

- в подвійних лапках ;
- в подвійних лапках з @ на початку.

В першому варіанті, використовуються Escape–послідовності або керуючі послідовності, а в другому ні. Тому другий варіант зручний для того, щоб вказувати шлях до файлу. Види Escape – послідовностей перераховані в таблиці 5.2.

Таблиця 5.2 – Керуючі послідовності

| **Escape – послідовність** | **Опис** |
|---|---|
| \a | Дзвоник |
| \b | Повернення на одну позицію |
| \f | Перехід на нову сторінку |
| \n | Перехід на новий рядок |
| \r | Повернення каретки |
| \t | Горизонтальна табуляція |
| \v | Вертикальна табуляція |
| \0 | null |
| \’ | Символ - ’ |
| \” | Символ – ” |
| \\ | Символ - \ |
| \udddd або \xdddd | Представляє знак Юнікоду де dddd – шістнадцяткове число |

Наприклад:

```
using System;
class Program {
    static void Main(string[] args) {
        Console.WriteLine("\u0041BC\n\\");
        Console.WriteLine(@"C:\Temp\file1.txt");
        Console.ReadKey();
    }
```

```
}
```

Виведе на екран:

```
ABC
\
C:\Temp\file1.txt
```

## 5.4. Основні елементи класу System.String

Для роботи з рядками в класі String передбачено ряд властивостей та методів (див. табл. 5.3).

| Елемент | Тип елементу | Пояснення |
|---|---|---|
| Compare | Статичний метод | Порівняння двох рядків у лексикографічному (алфавітному) порядку. Різні реалізації методу дозволяють порівнювати рядки і підрядка з урахуванням і без урахування регістра і особливостей національного уявлення дат і т. Д. |
| CompareOrdInal | Статичний метод | Порівняння двох рядків за кодами символів. Різні реалізації методу дозволяють порівнювати рядки і підрядка |
| CompareTo | Метод | Порівняння поточного екземпляра рядки з іншого рядком |
| Concat | Статичний метод | Конкатенація рядків. Метод допускає зчеплення довільного числа рядків |
| Copy | Статичний метод | Створення копії рядку |
| Empty | Статичне поле | Порожній рядок (тільки для читання) |
| Format | Статичний метод | Форматування відповідно до заданих специфікаторами формату, як для Console.WriteLine |
| IndexOf, IndexOfAny, LastIndexOf, LastIndexOfAny | Методи | Визначення індексів першого і останнього входження заданого підрядку або будь-якого символу із заданого набору |

| | | |
|---|---|---|
| Insert | Метод | Вставка підрядку в задану позицію |
| Join | Статичний метод | Злиття масиву рядків в єдину рядок. Між елементами масиву вставляються роздільники (див. Далі) |
| Length | Властивість | Довжина рядка (кількість символів) |
| PadLeft, PadRight | Методи | Вирівнювання рядки по лівому або правому краю шляхом вставки потрібного числа пробілів на початку або в кінці рядка |
| Remove | Метод | Видалення підрядку із заданої позиції |
| RepIace | Метод | Заміна всіх входжень заданого підрядку або символу новими підрядком або символом |
| Split | Метод | Розділяє рядок на елементи, використовуючи задані роздільники. Результати поміщаються в масив рядків |
| StartsWith, EndsWith | Методи | Повертає true або false залежно від того, починається або закінчується рядок заданої підрядком |
| Substring | Метод | Виділення підрядка, починаючи з заданої позиції |
| ToCharArray | Метод | Перетворення рядка в масив символів |
| ToLower, ToUpper | Методи | Перетворення символів рядка до нижнього або верхнього регістру |
| Trim, TrimStart, TrimEnd | Методи | Видалення пробілів на початку і наприкінці рядка або тільки з одного її кінця (зворотні по відношенню до методів PadLeft і PadRight дії) |

Приклад. За допомогою методів класу System.String – написати перше та третє слова фрази "a black cat" з великої літери:

```
using System;

class Lab05 {
    static void Main(string[] args) {
        string a = "a black cat";
```

```
            // Заміна першої літери на велику
            string s = a.Substring(0, 1).ToUpper() + a.Substring(1);
            // Знаходимо перший пробіл
            int i = a.IndexOf(' ');
            // Знаходимо другий пробіл
            int j = a.IndexOf(' ', i + 1);
            j++;
            // Робимо 1-шу букву 3-го слова великою
            string s1 = s.Substring(j, 1).ToUpper();
            // Вставляємо її замість малої
            s = s.Remove(j, 1).Insert(j, s1);
            // Виводимо результат
            Console.WriteLine(s);
            Console.ReadKey();

        }
}
```

Програма виведе в консоль:

```
A black Cat
```

### 5.5. Інтерполяція рядків на C#

Функція інтерполяції рядків створена на основі форматування і має зручний синтаксис для включення змінних та виразів у форматований рядок-результат.

Для задання інтерполяції рядків перед рядком-літералом додають $. Після цього можна включити будь-який допустимий вираз C#, значення якого можна вставити в інтерпольований рядок-результат. Для включення такого виразу в рядок використовують фігурні дужки, а принципи форматування подібні до тих що наведено раніше.

В наступному прикладі знайдемо площу кола:

```
using System;

class Program {
    static void Main() {
        double r = 2;
        Console.WriteLine(
          $"Площа кола радіуса {r} рівна {Math.PI*r*r:f4}");
        Console.ReadKey();
    }
}
```

Результат наступний:

```
Площа кола радіуса 2 рівна 12,5664
```

Інтерполяція працює для C# починаючи з шостої версії, що з'явилася разом з с Visual Studio 2013.

### 5.6. Клас System.Text.StringBuilder

Об'єкт String є незмінним. Кожного разу при використанні одного з його методів створюється новий об'єкт-рядок, що вимагає виділення нового блоку в пам'яті, а ця операція порівняно повільна. Тому у випадках, коли необхідно виконувати зміни рядка, що повторюються багато разів, витрати, пов'язані зі створенням нових об'єктів String, стають суттєвими. Вирішити цю проблему можна використанням класу System.Text.StringBuilder, методи якого дозволяють змінювати вміст рядка без створення нового об'єкту. Наприклад використання класу StringBuilder може підвищити продуктивність при додаванні великої кількості рядків в циклі [10].

### Контрольні запитання

1. Поясніть суть типу даних **`System.Char`**. Назвіть його основні методи.
2. Який тип даних використовується для роботи з текстовими рядками. Назвіть основі операціїї, визначені для рядків.
3. Поясніть принцип роботи з керуючими послідовностями.
4. Назвіть основні методи класу **`System.String`**.
5. Що таке інтерполяція рядків на C#?
6. Для чого використовується клас **`System.Text.StringBuilder`**?

# 6. ОПЕРАТОРИ РОЗГАЛУЖЕННЯ. УМОВНИЙ ОПЕРАТОР IF ТА ОПЕРАТОР МНОЖИННОГО ВИБОРУ SWITCH

## 6.1. Умовний оператор if

Умовний оператор дозволяє програмі перевірити логічну умову, і в залежності від результату перевірки, виконати той чи інший фрагмент коду. Синтаксис умовного оператору наступний [5]:

```
if (<Умова>)
        оператор1;
[else
        оператор2;]
```

`Умова` – логічний вираз. Якщо `Умова` прийме значення **`true`**, то виконується перший оператор або набір операторів, інакше другий оператор або набір операторів. Причому друга частина умовного оператору **`else`** … - не є обов'язковою.

При використанні набору операторів вони об'єднуються в блок, та обмежуються фігурними дужками, тоді синтаксис умовного оператору набуває наступного вигляду:

```
if (<Умова>){
     оператори1
}
[else{
     Оператори2
}]
```

Структурна схема оператора наведена на рис. 6.1.

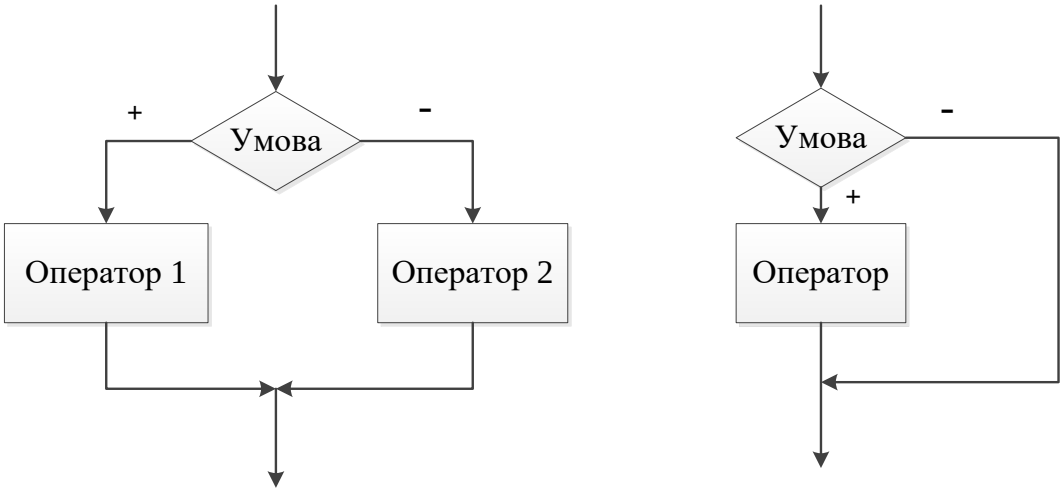


Рисунок 6.1 - Структурна схема умовного оператора

## 6.2. Логічні вирази

Логічні або умовні вирази – це логічні конструкції, що використовуються для керування ходом виконання програми. Вони можуть включати в себе:

- Операції порівняння **(==,!=, >, <, >=, <=)**, які для порівняння використовують два операнди одного типу, і повертають значення **true** або **false**.
- Логічні операції **(!, &, &&, |, || , ^)** – оператори, що використовуються для роботи з логічним типом даних **bool** (або **System.Boolean**).

**Оператор логічного заперечення !** – унарна операція, що повертає **false**, якщо операнд має значення **true**, і **true**, якщо операнд має значення **false**.

**Оператор логічного І &** - результат операції **x & y** приймає значення **true** тоді і тільки тоді, коли обидва операнди **x** та **y** мають значення **true**. Якщо хоча б один з них має значення **false** результатом буде **false.**

Оператор завжди **&** обчислює обидва операнди.

**Умовний оператор логічного І &&** - працює аналогічно до **&**, за одним виключенням: якщо при обчисленні виразу перший операнд отримав значення **false**, то другий операнд не обчислюється і результатом операції буде **false**.

**Оператор логічного АБО |** - результат операції **x | y** приймає значення **true**, якщо хоча б один з операндів **x** або **y** має значення **true**. Коли обидва операнди **false** – результатом буде **false**.

Оператор **|** завжди обчислює обидва операнди.

**Умовний оператор логічного АБО ||** - працює аналогічно до **|**, за одним виключенням: якщо при обчисленні виразу перший операнд отримав значення **true**, то другий операнд не обчислюється і результатом операції буде **true**.

**Оператор логічного виключного АБО ^** - результат операції **x ^ y** приймає значення **true,** якщо значення операндів різні, і **false**, якщо однакові. Тобто для операндів типу **bool**, працює як оператор нерівності **!=**.

Зауваження: Для цілих типів даних операції **&, |, ^** працюють інакше, - як побітові логічні операції.

### 6.3. Порівняння дійсних чисел

Слід уникати перевірки дійсних величин на рівність. Замість цього краще порівнювати модуль їх різниці з деяким малим числом, тобто з заданою точністю. 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 константним полем Epsilon (Наприклад: double.Epsilon == 4.94065645841247E-324 це мінімально можливе значення змінної таке, що 1.0 + double.Epsilon ! = 1.0).

### 6.4. Оператор вибору switch

Інший оператор для керування логікою програми в C# - це оператор **switch**. Він дозволяє керувати ходом програми при скінченому наборі варіантів вибору, що заздалегідь відомі. Синтаксис цього оператору наступний:

```
switch (<вираз>){
    case <значення1>:
        оператори1
        break;
    [[case <значення2>:
        оператори2
        break;]
      . . .
    default:
        оператори_за_замовчуванням
        break;]
}
```

Тут тип `виразу` може бути одним зі злічених типів (байт, символ, ціле і т.д.) або рядком. Якщо значення виразу співпаде зі значенням `значення1`, то виконається набір операторів `оператори1`, якщо зі значенням `значення2`, то виконається набір операторів `оператори2`. Якщо ж не підійде жоден з перерахованих варіантів, то виконається набір

операторів `оператори_за_замовчуванням` після ключового слова **`default`**.

***Зауваження***: Починаючи з C# 7.0 тип виразу може бути довільним, окрім NULL.

Наприклад:

```
int i = 1;
switch (i){
    case 1:
        Console.WriteLine("Один");
        break;
    case 2:
        Console.WriteLine("Два");
        break;
    default:
        Console.WriteLine("Інше число");
        break;
}
```

Блок-схема до цього фрагменту коду матиме вигляд (рис. 6.2):

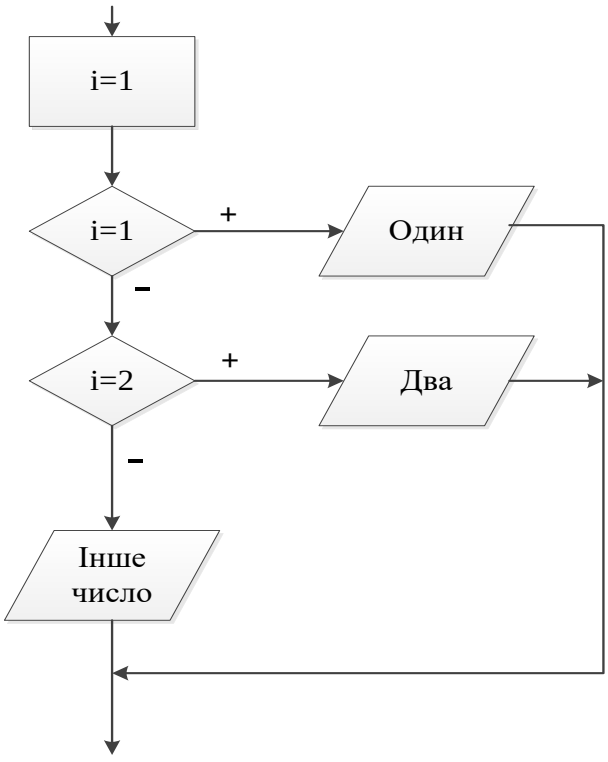


Рисунок 6.2 - Структурна схема оператора switch

Наступний приклад показує як для різних значень (1,2,3) виконати одні й ті самі дії:

```
int i = 2;
switch(n){
    case 1:
```

```
        case 2:
        case 3:
            Console.WriteLine("Це 1, 2 або 3");
            break;
         default:
              Console.WriteLine("Інше число");
               break;
}
```

Щоб моделювати більш складні дії в операторі **switch**, можна замінити оператор **break** на **goto**, наприклад:

```
int d = 2, k=1;
switch (d){
    case 1:
        k++;
        break;
    case 2:
        k += 3;
        goto case 1;
}
```

В цьому прикладі після виконання варіанту 2, буде зроблено перехід до варіанту 1, і виконаються ще і його оператори, тобто спочатку **k** збільшиться на 3, а потім збільшиться ще на один.

Ключове слово **default** – необов'язкове в операторі **switch**.

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.

**Контрольні запитання**

1. Поясніть принцип роботи оператора **if**.
2. Наведіть формат запису умовного оператора **if** та його структуру.
3. Що таке логічні вирази? Які логічні оператори Ви знаєте?
4. Поясніть особливість порівняння дійсних чисел.
5. Поясніть принцип роботи оператора **switch**.
6. Наведіть синтаксис оператора вибору **switch**.
7. Наведіть приклад використання оператора **goto**.

# 7. ІТЕРАЦІЙНІ КОНСТРУКЦІЇ. ЦИКЛ FOR.

## 7.1. Оператори циклу

Цикл — різновид керуючої конструкції у високорівневих мовах програмування, що призначена для організації багаторазового виконання набору інструкцій. Цикли – дозволяють виконувати один і той самий блок коду до тих пір, поки не виконається якась умова.

Блок коду, що виконується в циклі називається тілом циклу. Одноразове виконання тіла циклу називають ітерацією.

Цикли поділяються на:

- безумовні,
- з передумовою (рис. 7.1 а),
- з післяумовою (рис. 7.1 б),
- з виходом з середини,
- з лічильником,
- по колекції.

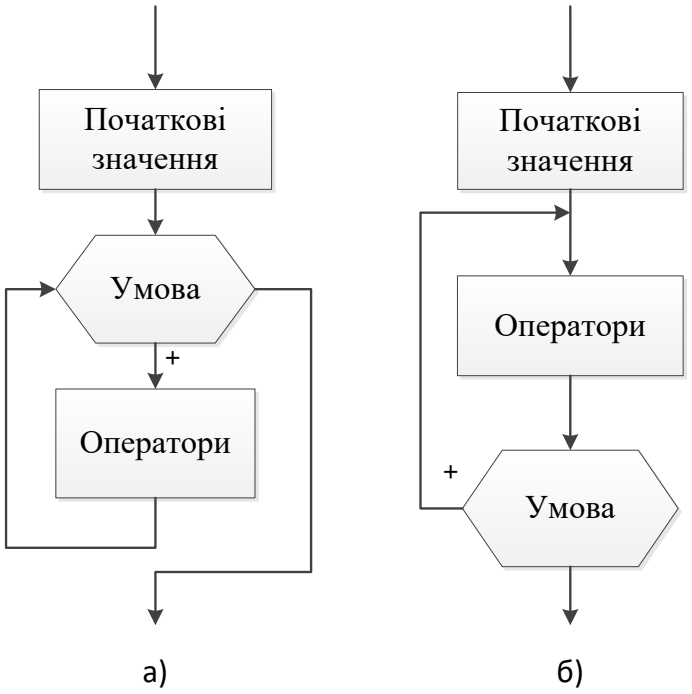


Рисунок 7.1 - Структурні схеми операторів циклу
а) з передумовою, б) з післяумовою

В мові C# передбачено чотири види циклів:

- **`for`** – цикл з параметром
- **`while`** – цикл з передумовою
- **`do ... while`** – цикл з післяумовою
- **`foreach ... in`** - по колекції

Перші три перейшли в мову C# з класичного C, а **`foreach`** – Visual Basic.

### 7.2. Цикл з параметром for

Цикл for – найбільш універсальний з циклів у мовах C, C++ та Java, з його допомогою можна змоделювати роботу всіх інших циклів. Синтаксис цього циклу:

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

**Ініціалізатор** – вираз або набір виразів, перерахованих через кому, що обчислюється перед початком циклу. Зазвичай використовується для задання початкових значень змінних, що використовуються в циклі.

В якості ініціалізатора можна використати одне з двох:

- Оголошення та ініціалізацію локальної змінної циклу, що буде доступна тільки в середині циклу.
- Нуль або більше операторів, розділених комами, з наступного списку:
    - оператор присвоювання;
    - виклик методу;
    - постфіксна або префіксна операція інкременту або декременту;
    - створення об’єкту за допомогою ключового слова new;
    - вираз await для призупинки виконання в асинхронному методі.

**Умова** – логічний вираз, значення якого перевіряється перед кожною операцією циклу, якщо значення true, то виконується тіло циклу, якщо ж false – цикл завершується.

**Ітератор** - вираз або набір виразів, перерахованих через кому, що обчислюються в кінці кожної ітерації циклу. Допустимі вирази:

- Оператор присвоювання.
- Виклик методу.
- Постфіксна або префіксна операція інкременту або декремент.
- Створення об’єкту за допомогою ключового слова new.
- Вираз await.

**Оператор** – тіло циклу, виконується на кожній ітерації. Може замінюватись набором операторів у фігурних дужках.

Слід зазначити, що всі три вищенаведені елементи циклу – необов'язкові. І найпростіший цикл, має вигляд:

```
for (;;){
    //... набір операторів
}
```

В цьому випадку, **for** являє собою нескінчений безумовний цикл, вихід з якого можна організувати за допомогою одного з операторів переходу.

### 7.3. Приклади використання циклу for

Найчастіше цикл **for** використовується у вигляді циклу з лічильником. Приклад 1:

```
for(int i = 1; i < 10; i++){
    Console.WriteLine(i);
}
```

цикл, що виводить на екран значення від 1 до 9 (рис. 7.2). Зауважимо, що змінна і оголошена в середині циклу, тому її область видимості обмежена тілом циклу.

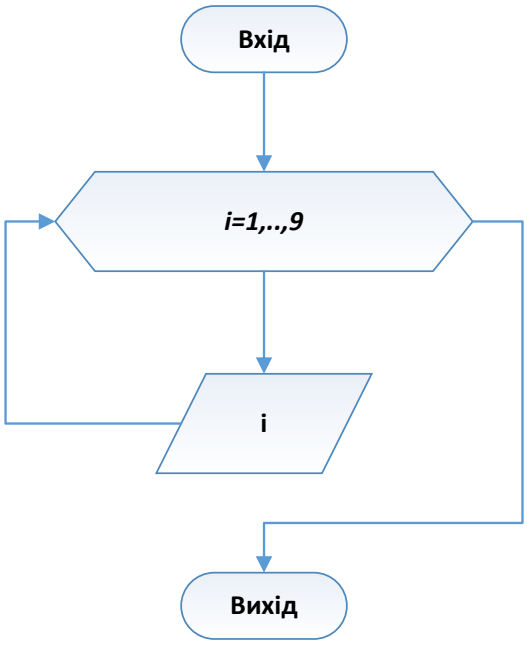


Рисунок 7.2 – Використання циклу for

Приклад 2: цикл з наборами виразів в ініціалізаторі та ітераторі:

```
int i, j;
for(i = 1, j = 2; i < 10 && j < 24; i++, j+=2){
    Console.WriteLine(i*j);
```

}

цикл де одночасно використовуються два лічильники (рис. 7.3).

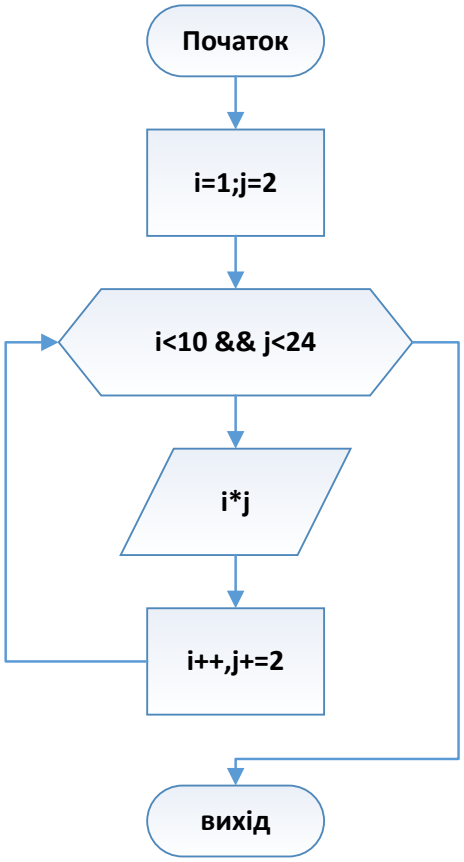


Рисунок 7.3 – Цикл for з двома лічильниками

Приклад 3: Розглянемо задачу - вивести таблицю значень функції з заданим кроком.

$$y = \begin{cases} x^4 + 2x, & якщо \quad x > 1, \\ tg(x), & якщо \; -1 \le x \le 1, \\ |x + 2|, & якщо \quad x < -1. \end{cases}$$

Назвемо початкове значення аргументу a, кінцеве значення аргументу - b, крок зміни аргументу - dx і параметр x. Всі величини дійсні. Програма повинна виводити таблицю, що складається з двох стовпців: значень аргументу і відповідних їм значень функції (рис. 7.4).

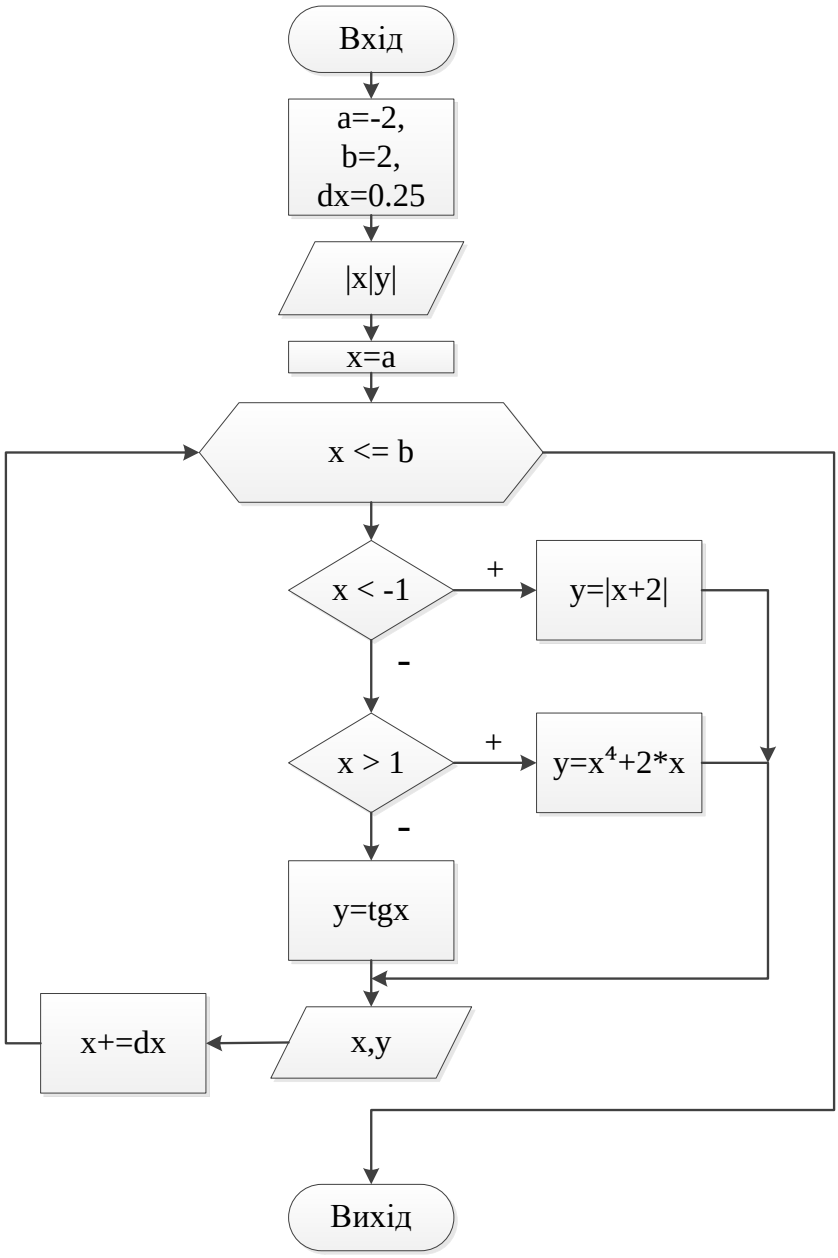


Рисунок 7.4 – Вивід значень функції

Опишемо алгоритм у словесній формі:

1. Взяти перше значення аргументу.
2. Визначити, якому з інтервалів воно належить, і обчислити значення функції за відповідною формулою.
3. Вивести рядок таблиці.
4. Перейти до наступного значення аргументу.
5. Якщо воно не перевищує кінцеве значення, повторити кроки 2-4, інакше закінчити.

Кроки 2-4 повторюються багаторазово, тому для їх виконання треба організувати цикл. Текст програми наведено нижче. Рядки програми позначені відповідними номерами кроків алгоритму.

```
using System;
class ForExample {
    static void Main() {
        double a = -2, b = 2, dx = 0.25, y;
        // Заголовок таблиці
        Console.WriteLine("|  x  |  y  |");
        for (double x = a; x <= b; x += dx) { // Кроки 1, 4, 5
            // Крок 2
            if (x < -1) y = Math.Abs(x + 2);
            else if (x > 1) y = Math.Pow(x, 4) + 2 * x;
            else y = Math.Tan(x);
            // Крок 3
            Console.WriteLine("|{0,5:f2}|{1,5:f2}|", x, y);
        }
        Console.ReadKey();
    }
}
```

Результатом роботи програми буде таблиця:

```
|  x  |  y  |
|-2,00| 0,00|
|-1,75| 0,25|
|-1,50| 0,50|
|-1,25| 0,75|
|-1,00|-1,56|
|-0,75|-0,93|
|-0,50|-0,55|
|-0,25|-0,26|
| 0,00| 0,00|
| 0,25| 0,26|
| 0,50| 0,55|
| 0,75| 0,93|
| 1,00| 1,56|
| 1,25| 4,94|
| 1,50| 8,06|
| 1,75|12,88|
| 2,00|20,00|
```

**Контрольні запитання**

1. Що таке цикл? Що таке ітерація?
2. Які види циклів Ви знаєте? Наведіть їх структурні схеми.
3. Поясніть принцип роботи циклу з параметром **`for`**. Наведіть його синтаксис.
4. Що таке ініціалізатор та ітератор?
5. Наведіть приклади використання циклу **`for`**.

# 8. ІТЕРАЦІЙНІ КОНСТРУКЦІЇ. ЦИКЛИ WHILE І DO / WHILE.

## 8.1. Оператор while

Цикл while – класичний цикл з передумовою, має наступний синтаксис:

```
while (<умова>) оператор;
```

**Умова** – повертає значення **true** або **false**. Це значення обчислюється та перевіряється перед кожною ітерацією, якщо значення **true** – то виконується тіло циклу, якщо **false** – цикл завершується.

**Оператор** – тіло циклу, виконується на кожній ітерації. Може замінюватись набором операторів у фігурних дужках.

Приклад:

```
int n = 1;
while (n < 5){
    Console.WriteLine("n = {0}", n);
    n++;
}
```

Виводить на екран рядки:

```
n = 1
n = 2
n = 3
n = 4
```

## 8.2. Знаходження найбільшого спільного дільника

Найбільший спільний дільник (НСД) двох або більше невід'ємних цілих чисел — найбільше натуральне число, на яке ці числа діляться без остачі[1].

Алгоритм Евкліда для обчислення найбільшого спільного дільника базується на тому, що при діленні чисел націло одне на одне, якщо залишок від ділення 0, то дільник і буде НСД, якщо залишок – відмінний від 0 то він також ділитиметься на НСД.

```
a = bq + r,
b = r*q1 + r1,
r = r1*q2 + r2,
...
rk-2 = rk-1*qk  ( rk  = 0)
```

Доведення нехай n – НСД(a,b)

Тоді:

**a = l*n, b = m*n**

З **a = bq + r** підставивши, отримаємо **l*n = m*n*q + r**

Виразимо **r:**

**r = l*n - m*n*q = (l – m*q) * n**

**(l – m*q)** – ціле число, отже **r** ділиться на **n** без остачі.

Перевіримо на числах. Знайдемо НСД для 30 и 18.

30/18 = 1 (залишок 12)

18/12 = 1 (залишок 6)

12/6 = 2 (залишок 0).

Кінець. НСД (30, 18) = 6

Алгоритм методу Евкліда:

**Крок 1**: Вводимо значення a та b.

**Крок 2**: Якщо b = 0, то виводимо значення a. Задача розв'язана.

Якщо b відмінне від 0, переходимо до наступного кроку.

**Крок 3**: Знаходимо r залишок від ділення a на b.

**Крок 4**: В a запишемо значення b, в b запишемо значення r. Переходимо до другого кроку.

Реалізуємо алгоритм Евкліда у вигляді програми і блок-схеми (рис. 8.1):

```
class Program {
    static void Main(string[] args) {
        Console.WriteLine("Введіть два числа a та b");
        int a = int.Parse(Console.ReadLine());
        int b = int.Parse(Console.ReadLine());
        while (b != 0) {
            int r = a % b;
            a = b;
            b = r;
        }
        Console.WriteLine("НСД = " + a);
        Console.ReadKey();
    }
}
```

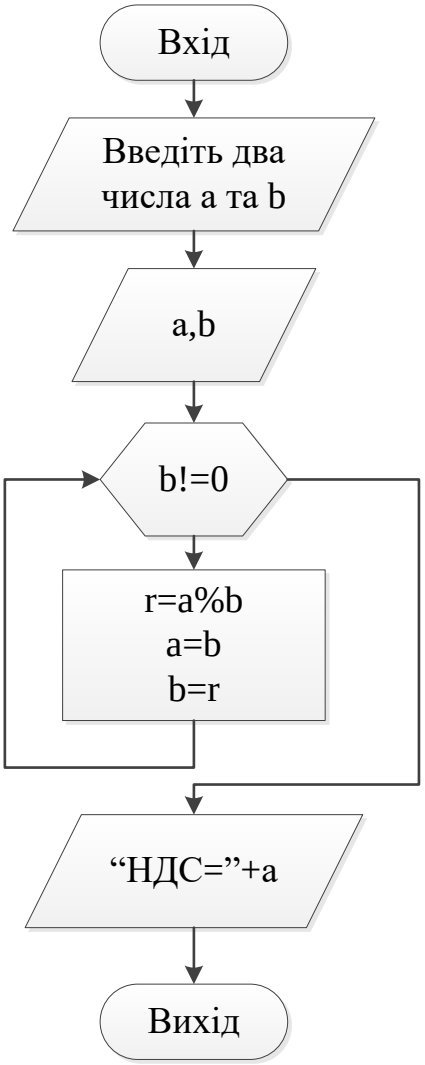


Рисунок 8.1 – Алгоритм Евкліда

### 8.3. Оператор do ... while

Цикл do ... while – цикл з післяумовою. Аналогічний до while за виключенням того, що умова перевіряється в кінці кожної ітерації, а не перед початком. Має наступний синтаксис:

```
do
    оператор;
while (<умова>);
```

**Умова** – повертає значення **true** або **false**. Це значення обчислюється та перевіряється наприкінці кожної ітерації, якщо значення **true** – то ще раз виконується тіло циклу, якщо **false** – цикл завершується.

**Оператор** – тіло циклу, виконується на кожній ітерації. Може замінюватись набором операторів у фігурних дужках.

Приклад:

```
int n = 1;
do {
    Console.WriteLine("n = {0}", n);
    n++;
}
while (n < 0);
```

Виводить на екран рядок:

```
n = 1
```

Стає помітним, що так як умова перевіряється в кінці, то тіло циклу обов'язково виконається хоч один раз.

### 8.4. Метод половинного ділення (Дихотомія)

Нехай дано рівняння f(x)=0. Необхідно знайти його корінь з точністю E на відрізку [a,b], на якому функція безперервна і у кінцях має значення різних знаків, тобто $f(a)\cdot f(b)<0$. Таким чином, згідно теореми, на цьому відрізку існує хоча б один розв'язок рівняння.

Знаходиться середина відрізку [a,b] точка c (рис. 8.2). Корінь може опинитись на відрізку [a,c] або на [c,b], чи співпасти з c. В останньому випадку метод припиняє роботу, інакше за допомогою перевірки виконання умов $f(a)\cdot f(c)<0$ і $f(c)\cdot f(b)<0$ з'ясовується, на якій частині відрізку залишився корінь. Далі процедура повторюється для тієї половини відрізку, на якій є корінь, доки відрізок не зменшиться настільки, що його довжина буде менше від заданої похибки.

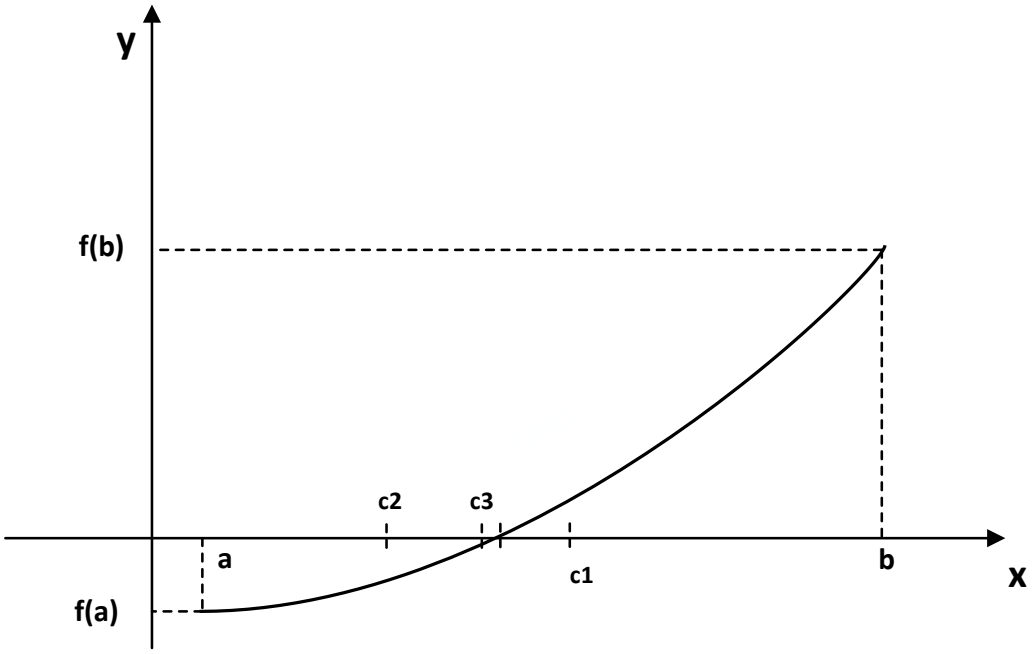


Рисунок 8.2 – Метод половинного ділення

Алгоритм методу:

**Крок 1.** Знаходиться середина відрізку c:= (b+a)/2.

**Крок 2.** Перевіряються наступні умови.

1. Якщо f(c)<E – корінь знайдено.
2. Якщо f(a)·f(c)<0 – корінь на [a,c], тому b:=c.
3. Якщо f(c)·f(b)<0 – корінь на [c,b], тому a:=c.

**Крок 3.** Перевіряється умова b-a > E. Якщо вона виконується, то корінь знайдено. В цьому випадку він дорівнює (a+b)/2. Інакше повертаються до кроку 1.

Похибка розв'язку $\Delta$ через *n* ітерацій знаходиться в межах

$$\Delta \le \frac{1}{2^n}|x_1 - x_0|.$$

Метод має малу швидкість збіжності, оскільки інтервал, де знаходиться корінь, з кожним кроком зменшується не більше, ніж в два рази.

Приклад. Знайдемо корінь функції косинус на проміжку [0,2] (рис. 8.3)

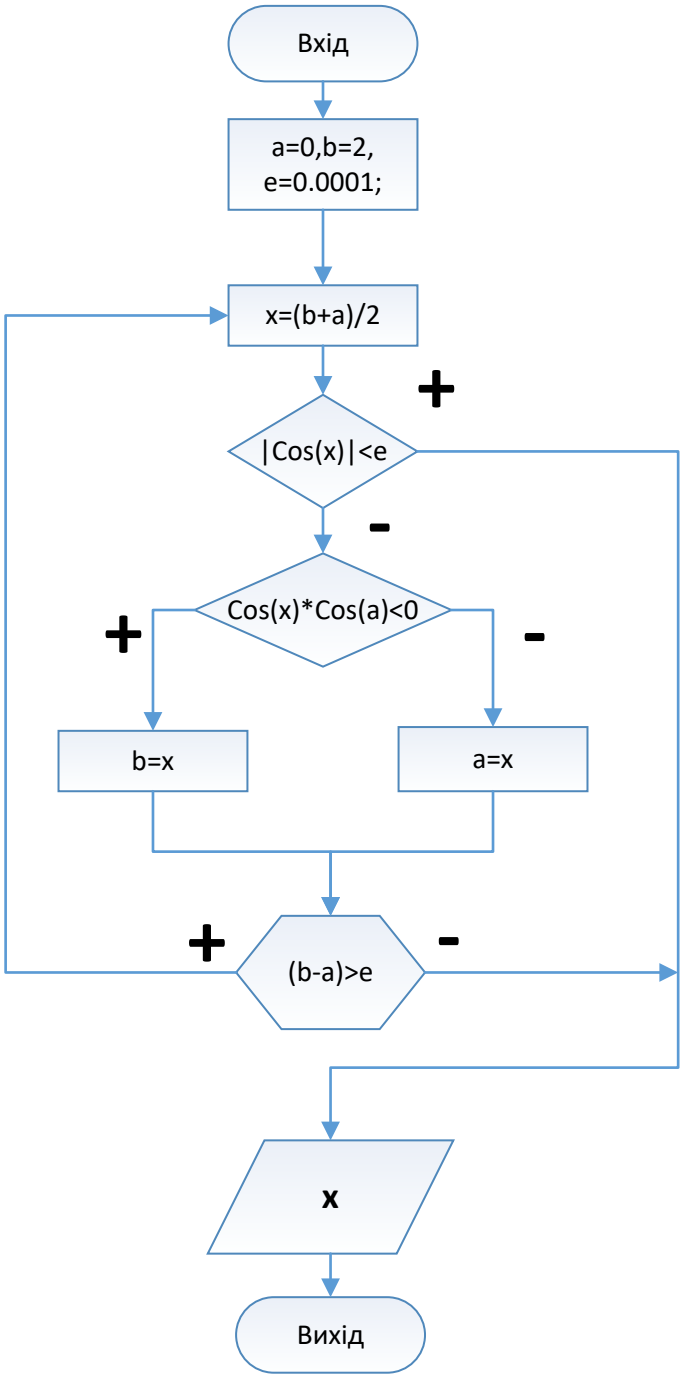


Рисунок 8.3 – Алгоритм методу половинного ділення

Програма:

```
using System;
namespace ConsoleApplication01 {
    class Program {
        static void Main() {
            double a = 0, b = 2, e = 0.0001;
            double x;
            do {
                x = (b + a) / 2;
                if (Math.Abs(Math.Cos(x)) < e) break;
```

```
                if (Math.Cos(x) * Math.Cos(a) < 0) {
                    b = x;
                } else {
                    a = x;
                }
            } while (b - a > e);
            Console.WriteLine("x = {0:F5}", x);
            Console.ReadKey();
        }
    }
}
```

В результаті буде:

```
x = 1,57080
```

Що співпадає зі значенням числа π/2 з заданою точністю.

### 8.5. Оператори переходу (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.

### 8.6. Оператор goto

Оператор безумовного переходу goto використовується в одній з трьох форм:

```
goto мітка;
goto case літерал;
goto default;
```

У тілі цього ж метода повинна бути присутня рівно одна конструкція виду мітка: оператор;

Оператор goto передає управління на позначений оператор. Мітка - це звичайний ідентифікатор, що закінчується двокрапкою. 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).

Друга і третя форми оператора goto використовуються в тілі оператора вибору switch. Оператор **`goto case літерал`** передає управління на гілку відповідну до літералу, 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.

### 8.7. Оператори `break` та `continue`

Оператор **`break`** - перериває виконання циклу і передає управління до наступного за циклом оператора.

Оператор **`continue`** – перериває поточну ітерацію циклу і переходить одразу до наступної ітерації.

Приклад використання операторів break та continue (блок-схема рис.8.4)

```
using System;
class Program {
    static void Main() {
        for (int i = 0; i < 99; i++) {
            if (i % 3 == 0) continue;
            if (i > 10) break;
            Console.WriteLine(i);
        }
        Console.ReadKey();
    }
}
```

Виведе:

```
1
2
4
5
7
```

```
8
10
```

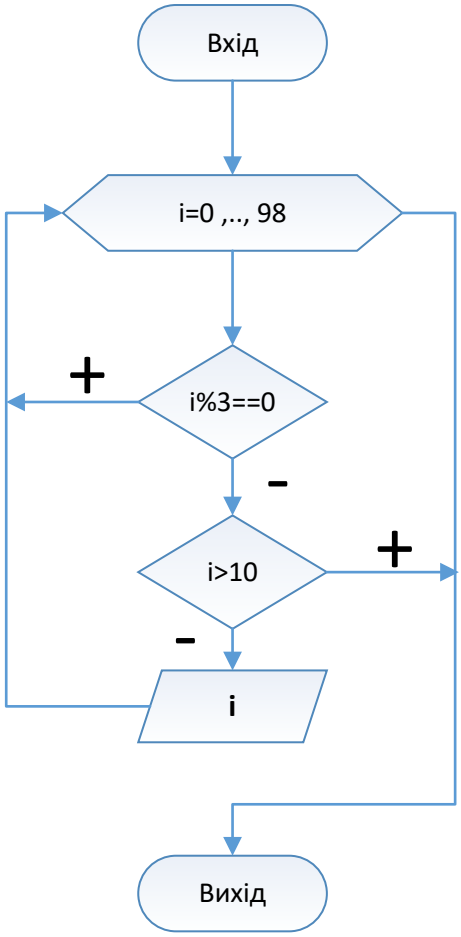


Рисунок 8.4 - Використання операторів break та continue

**Контрольні запитання**

1. Поясніть принцип роботи оператора **`while`**. Наведіть його синтаксис.
2. Поясніть принцип роботи оператора **`do ... while`**. Наведіть його синтаксис.
3. Назвіть основні оператори переходу (передачі управління) та поясніть принцип їх роботи.
4. Поясніть принцип роботи оператора безумовного переходу **`goto`**.
5. Назвіть три форми використання оператора безумовного переходу **`goto`** та поясніть особливості роботи оператора **`goto`** при кожній з цих форм.
6. Поясніть принцип роботи операторів **`break`** та **`continue`**.

# 9. АЛГОРИТМИ З ВИКОРИСТАННЯМ ВКЛАДЕНИХ ЦИКЛІВ.

## 9.1. Пошук найбільшого дільника

Розглянемо задачу. Для всіх чисел, що не більше числа k, перевірити чи є вони простими, якщо ні знайти найбільший дільник відмінний від самого числа. Значення k вводиться з клавіатури.

Щоб розв'язати задачу треба: за допомогою одного циклу, по змінній i, перебрати числа від 2 до k, за допомогою другого, по змінній j, пошукати його дільники в діапазоні від 2 до $\sqrt[2]{i}$. Для найменшого знайденого дільника j, найбільший дільник знаходиться, як i/j. Якщо ж, жодного дільника в заданому діапазоні не виявиться, то число i – просте (рис. 9.1).

```
using System;

class Program {
    static void Main() {
        Console.WriteLine("Введіть число k:");
        int k = int.Parse(Console.ReadLine());
        // Перебираємо числа від 2 до k
        for (int i = 2; i <= k; i++) {
            int divisor = 1;
            // Шукаємо найменший дільник
            for (int j = 2; j <= Math.Sqrt(i); j++) {
                if (i % j == 0) {
                    // Обчислюємо найбільший
                    divisor = i / j;
                    break;
                }
            }
            if (divisor == 1) {
               Console.WriteLine("{0,3} - просте число", i);
            } else {
               Console.WriteLine("{0,3} - дільник {1}", i,
divisor);
            }
        }
        Console.ReadKey();
    }
}
```

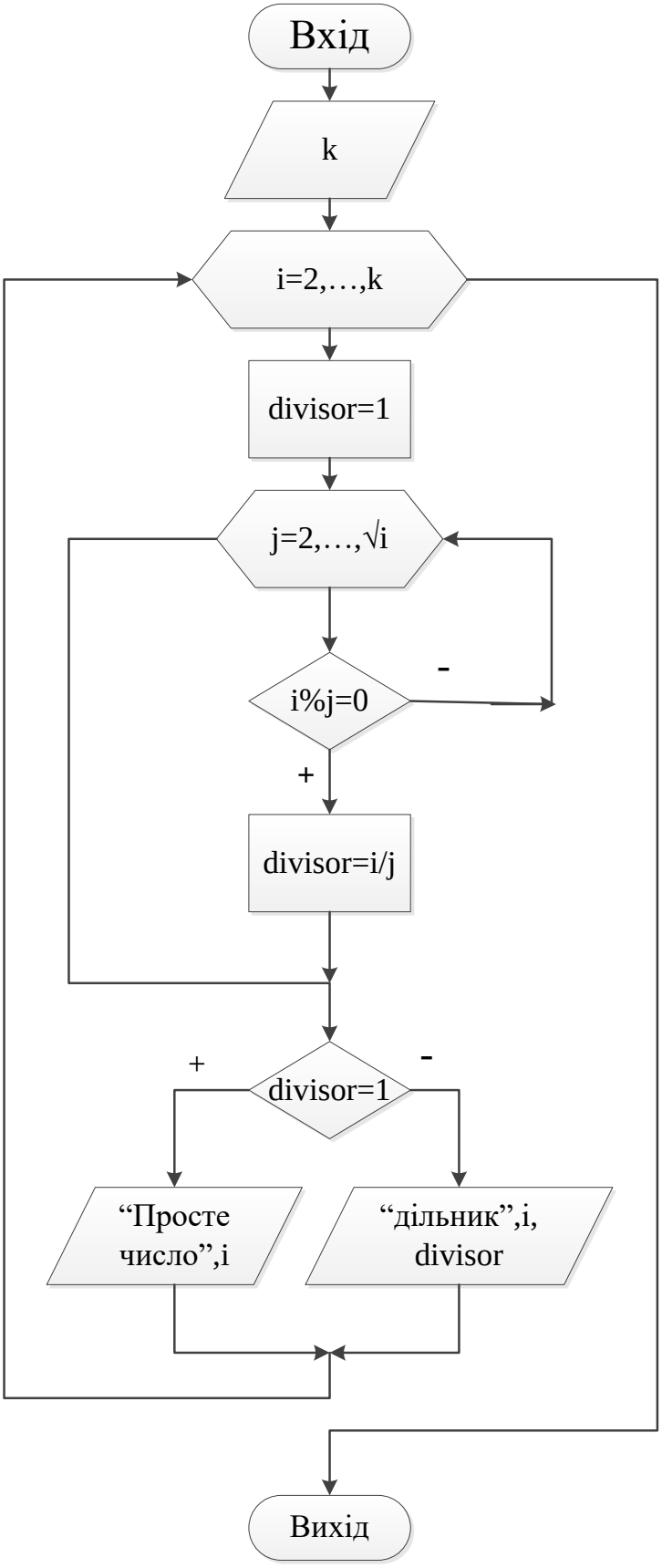


Рисунок 9.1 - Пошук найбільшого дільника

Результат роботи програми для k==15.

```
Введіть число k:
15
  2 - просте число
  3 - просте число
  4 - дільник 2
  5 - просте число
  6 - дільник 3
  7 - просте число
  8 - дільник 4
  9 - дільник 3
 10 - дільник 5
 11 - просте число
 12 - дільник 6
 13 - просте число
 14 - дільник 7
 15 - дільник 5
```

### 9.2. Знаходження суми ряду

Наступна задача. Написати програму на мові C#, без використання методів класу **Math**, яка обчислить суму, що знаходиться ряду праворуч у наведеній формулі. Отримане значення перевірити.

$$e^{-x^2} = 1 - \frac{x^2}{1!} + \frac{x^4}{2!} - \frac{x^6}{3!} + \cdots (-1)^n \frac{x^{2n}}{n!} + \cdots, \qquad -\infty < x < \infty$$

Розв'язання. Оскільки:

$$n! = 1 * 2 * 3 * \ldots * n$$

$$x^{2n} = x * x * \ldots * x, 2n - \text{раз}$$

$$(-1)^n = (-1) * (-1) * \ldots * (-1), n - \text{раз}$$

То, цю задачу можна розв'язати, за допомогою вкладених циклів, кожного разу знаходячи член ряду, по формулі:

$$a_i = (-1)^i \frac{x^{2i}}{i!} \quad \text{де} \quad i = \overline{1,n}$$

Тоді програма матиме вигляд:

```
using System;
namespace ForExample02 {
    class Program {
        static void Main() {
            Console.WriteLine("Введіть n");
            int n = int.Parse(Console.ReadLine());
            Console.WriteLine("Введіть x");
            double x = double.Parse(Console.ReadLine());
```

```
            double sum = 0, a;
            for (int i = 0; i <= n; i++) {
                // Обчислюємо степінь -1
                double step1=1;
                for (int j = 1; j <= i; j++) step1 *= -1;
                // Обчислюємо степінь x
                double stepX = 1;
                for (int j = 1; j <= 2 * i; j++) stepX *= x;
                // Обчислюємо факторіал
                double fact = 1;
                for (int j = 1; j <= i; j++) fact *= j;
                // член ряду
                a = step1 * stepX / fact;
                // сума ряду
                sum += a;
            }
            Console.WriteLine("Sum=" + sum);
            Console.WriteLine("Exp=" + Math.Exp(-x * x));
            Console.ReadKey();
        }
    }
}
```

Результат роботи:

```
Введіть n
100
Введіть x
0,1
Sum=0,990049833749168
Exp=0,990049833749168
```

Як бачимо, обчислення проведено вірно. Але такий підхід призводить до надмірної кількості обчислень. Що б обчислити член ряду $a_i$, треба виконати більше 400 операцій множення, що негативно впливає на швидкість виконання програми. Блок-схема цієї програми на рис. 9.2.

В той же час можна помітити, що $a_i$ - член ряду можна отримати за допомогою $a_{i-1}$, домноживши його на відповідні коефіцієнти.

$$a_1 = -a_0 * \frac{x^2}{1}$$

$$a_2 = -a_1 * \frac{x^2}{2}$$

$$a_i = -a_{i-1} * \frac{x^2}{i}$$

Використавши наведені формули, програму можна зробити набагато коротшою та ефективнішою (рис. 9.3).

```
using System;

namespace ForExample03 {
    class Program {
        static void Main() {
            Console.WriteLine("Введіть n");
            int n = int.Parse(Console.ReadLine());
            Console.WriteLine("Введіть x");
            double x = double.Parse(Console.ReadLine());
            double a = 1, sum = a, x2 = x * x, sum_old;
            int i;
            for (i = 1; i <= n; i++) {
                // член ряду
                a = -(a * x2 / i);
                // запам'ятаємо стару суму
                sum_old = sum;
                // сума ряду
                sum += a;
                // перевірка на досягнення максимальної точності
                if (Math.Abs(sum - sum_old) < double.Epsilon)
break;
            }
            Console.WriteLine("Кроків=" + i);
            Console.WriteLine("Sum=" + sum);
            Console.WriteLine("Exp=" + Math.Exp(-x * x));
            Console.ReadKey();
        }
    }
}
```

Результат роботи:

```
Введіть n
100
Введіть x
0,1
Кроків=7
Sum=0,990049833749168
Exp=0,990049833749168
```

Як бачимо результат роботи співпадає з попереднім. Ще одна новація цієї програми, введення додаткової змінної **sum_old** для збереження попереднього значення суми, а також перевірки:

```
if (Math.Abs(sum - sum_old) < double.Epsilon) break;
```

яка дозволяє завершити виконання програми при досягнення заданої точності. Ці доповнення показали, що, для наведених початкових значень, достатньо виконати всього сім ітерацій для отримання достатньо точного результату, і не має необхідності виконувати решту 93. Такі нововведення називають **оптимізацією коду**, вони призначення для підвищення ефективності програми.

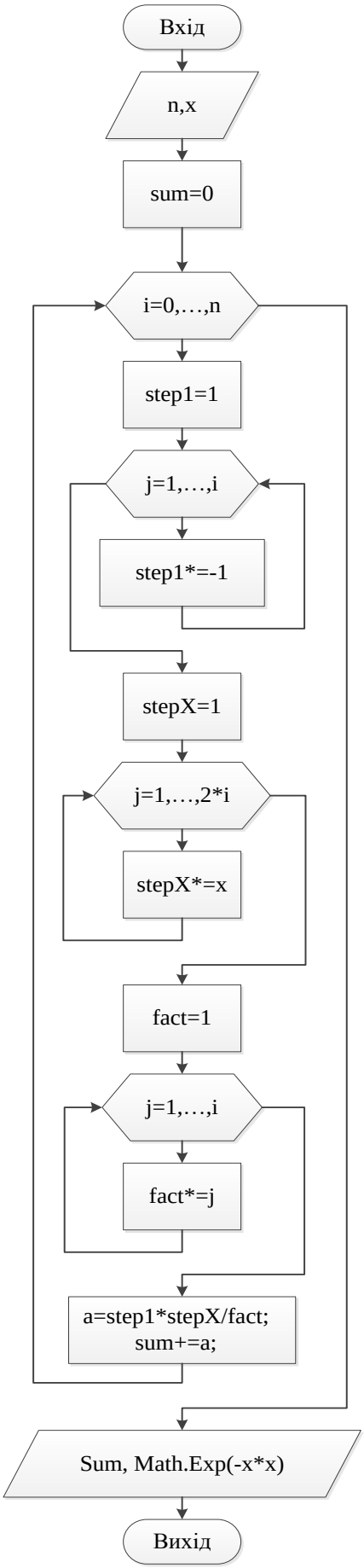


Рисунок 9.2 – Перший варіант знаходження суми ряду

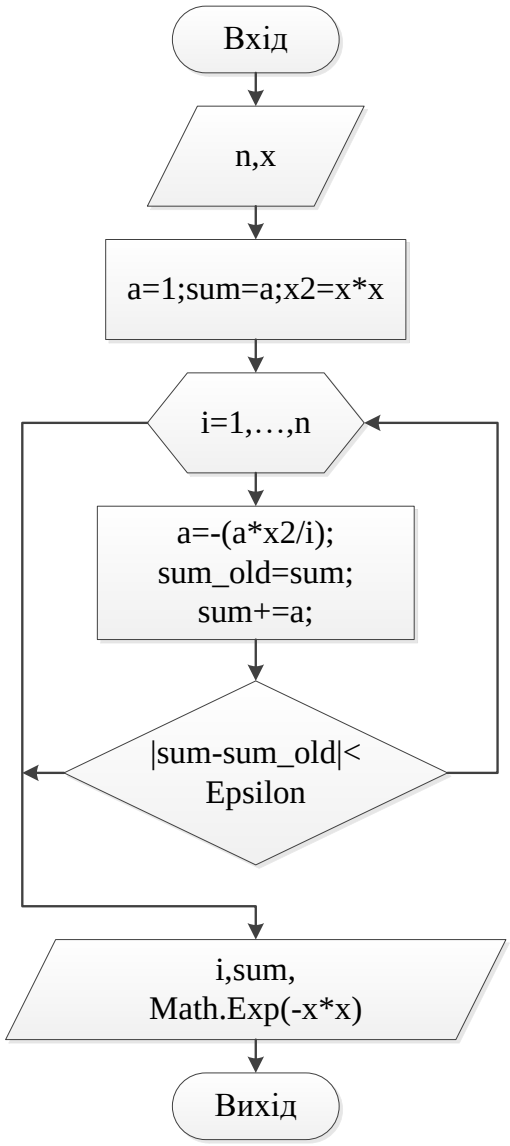


Рисунок 9.3 – Другий варіант знаходження суми ряду

**Контрольні запитання**

1. Наведіть приклад алгоритму з використанням вкладених циклів.
2. Поясніть поняття «оптимізація коду».
3. Наведіть структурну схему вкладених циклів.
4. Запишіть синтаксис вкладених циклів.

# 10. МАСИВИ. ІНІЦІАЛІЗАЦІЯ МАСИВІВ.

## 10.1. Поняття масиву

**Масив** – впорядкований набір елементів однакового типу, звертатися до яких можна за одним спільним ім'ям[1].

Масиви мають наступні властивості [11]:

- Масиви бувають одновимірними, багатовимірними або масивами масивів.
- Кількість вимірів і довжина кожного виміру задаються, коли створюється екземпляр масиву. Ці значення не можна змінити протягом існування екземпляру, тобто без створення нового масиву і перенесення в нього існуючих значень.
- Значення за замовчуванням для елементів числових масивів рівні нулю, для елементів-об'єктів присвоюється значення **`null`**.
- В масиві масивів елементи є типами посилань і ініціалізуються значенням **`null`**.
- Елементи масивів нумеруються від нуля, тобто елементи масиву з N елементів будуть мати індекси від 0 до N-1.
- Елементи масивів можуть мати будь-який тип, в тому числі і масив.
- Масиви – це об'єкти або типи посилань, всі вони є похідними від базового класу Array.

## 10.2. Ініціалізація одновимірних масивів

При оголошенні одновимірного масиву використовують наступний синтаксис:

**`<тип>[] <ім'я_масиву>;`**

де **`тип`** – тип даних, **`ім'я_масиву`** – ім'я масиву за яким в подальшому можна звертатись до його елементів.

Після оголошення масиву необхідно за допомогою оператору **`new`** задати його розмірність.

```
int[] myArray;
//масив цілих з десяти елементів
myArray = new int[10];
// масив з 5ти дійсних
double[] NumArr = new double[5];
```

Можна також одразу ініціювати масив початковими значеннями:

```
int[] myArray = new int[5] { 1, 3, 5, 8, 2 };
```

або ще простіше:

```
int[] myArray = { 1, 3, 5, 8, 2 };
```

без ключового слова **new** розмір масиву визначиться автоматично.

Підсумовуючи, для оголошення одновимірних масивів доступні наступні види синтаксису:

```
<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};
```

### 10.3. Індексація елементів одновимірного масиву

Звертатися до елементів масиву можна виказавши його ім'я та індекс елемента у квадратних дужках. Проте слід зазначити, що у мові C# ***нумерація елементів починається з нуля***, і перший елемент масиву матиме індекс 0.

Для роботи з елементами масиву зручно використовувати цикл **for**. Наприклад:

```
// Приклад використання for для масиву
using System;
class ArrayExample {
    static void Main() {
        // Масив цілих з 10 елементів
        int[] a = new int[10];
        // Заповнимо значеннями від 1 до 10
        for (int i = 0; i < 10; i++) {
            a[i] = i + 1;
        }
        // Виведемо результат
        for (int i = 0; i < 10; i++) {
            Console.WriteLine("a[{0}] = {1,2}", i, a[i]);
        }
        Console.ReadKey();
    }
}
```

При виконанні цієї програми отримаємо наступний результат.

```
a[0] =  1
a[1] =  2
a[2] =  3
```

```
a[3] =  4
a[4] =  5
a[5] =  6
a[6] =  7
a[7] =  8
a[8] =  9
a[9] = 10
```

Блок-схема цієї програми матиме вигляд представлений на рис. 10.1

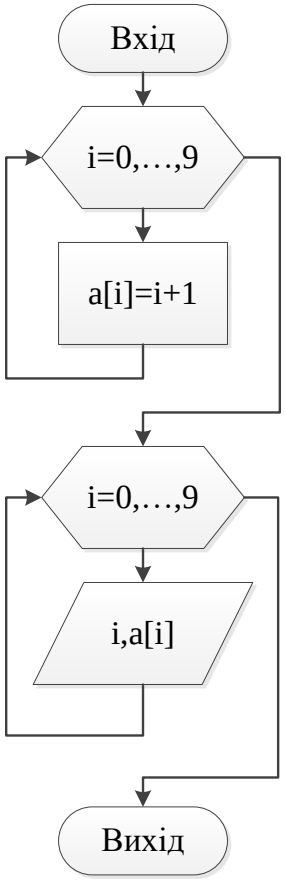


Рисунок 10.1 – Використання циклу for

## 10.4. Приклади застосування масивів

Масиви часто застосовуються в програмуванні, оскільки вони надають можливість легко зберігати та зчитувати велике число однотипних значень. Наприклад, в наведеній нижче програмі знаходиться середнє арифметичне чисел, що зчитуються з клавіатури та зберігаються в масиві arr, який циклічно опрацьовується за допомогою оператора циклу for.

```
// Обчислити середнє арифметичне ряду значень.
using System;
class Program {
    static void Main() {
```

```
            const int n = 5;
            int[] arr = new int[n];
            Console.WriteLine("Введіть {0} цілих чисел", n);
            double avg = 0;
            // Ввод даних з клавіатури
            for (int i = 0; i < n; i++) {
                arr[i] = int.Parse(Console.ReadLine());
            }
            // Обчислення середнього арифметичного
            for (int i = 0; i < n; i++) {
                avg += arr[i];
            }
            avg /= n;
            Console.WriteLine("avg = {0:F3}", avg);
            Console.ReadLine();
        }
    }
```

Блок-схема програми на рис. 10.2

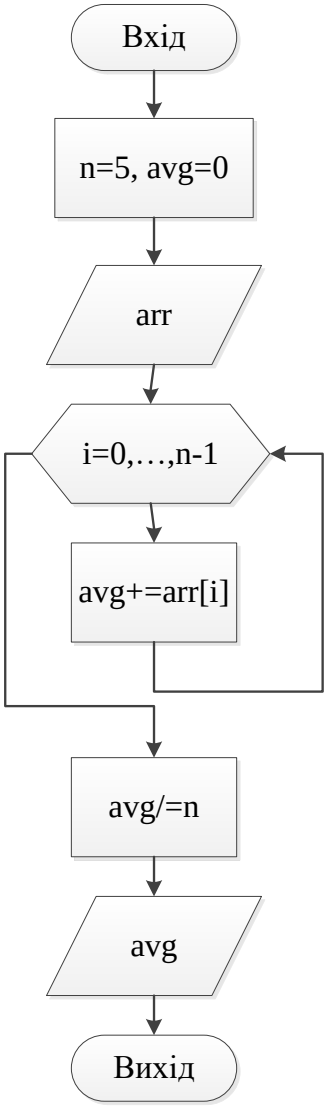

Рисунок 10.2 – Знаходження середнього арифметичного чисел

Результат роботи програми

```
Введіть 5 цілих чисел
123
44
73
225
99
avg = 112,800
```

Інший приклад. Знайти скалярний добуток двох векторів. За означенням: скалярний добуток n - вимірного евклідового простору дорівнює сумі добутків координат векторів.

$$\vec{x} \cdot \vec{y} := \sum_{i=1}^{n} x_i y_i = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n.$$

Блок-схема алгоритму на рис. 10.3

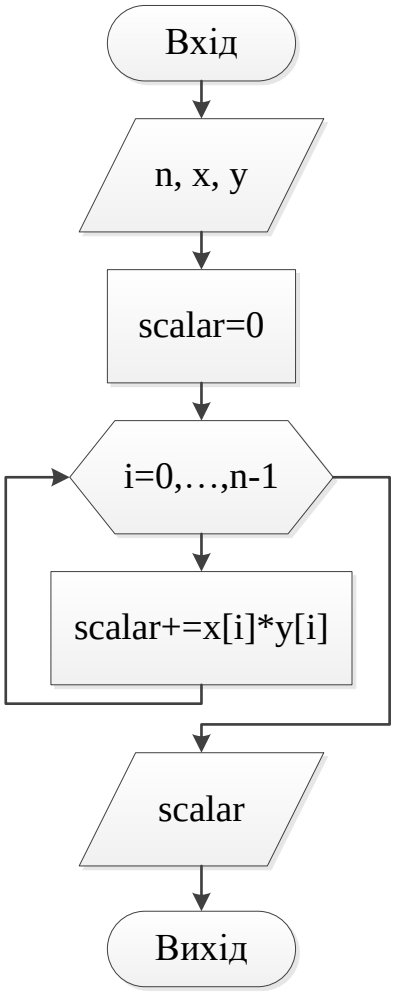


Рисунок 10.3 - Знайти скалярний добуток двох векторів

Програма матиме вигляд:

```
using System;

class Program {
    static void Main() {
        Console.WriteLine("Введіть розмірність векторів");
        int n = int.Parse(Console.ReadLine());
        double[] x = new double[n], y = new double[n];
        Console.WriteLine("Введіть координати x");
        for (int i = 0; i < n; i++) {
            x[i] = double.Parse(Console.ReadLine());
        }
        Console.WriteLine("Введіть координати y");
        for (int i = 0; i < n; i++) {
            y[i] = double.Parse(Console.ReadLine());
        }
        // scalar product
        double scalar = 0;
        for (int i = 0; i < n; i++) {
            scalar += x[i] * y[i];
        }
        Console.WriteLine("Скалярний добуток = {0:F3}", scalar);
        Console.ReadKey();
    }
}
```

Результат роботи програми:

```
Введіть розмірність векторів
3
Введіть координати x
1
2
3
Введіть координати y
4
2
1
Скалярний добуток = 11,000
```

**Контрольні запитання**

1. Що таке масиви? Назвіть їх основні властивості.
2. Наведіть приклад ініціалізації одновимірних масивів.
3. Назвіть види синтаксису для оголошення одновимірних масивів.
4. Поясніть принцип роботи з індексацією елементів одновимірного масиву.
5. Наведіть приклад застосування масивів.

# 11. ІТЕРАЦІЙНІ КОНСТРУКЦІЇ. ЦИКЛ FOREACH.

## 11.1. Цикл foreach...in

Цикл **foreach...in** – спеціальний цикл для роботи з набором елементів – колекцією (прикладом одного з типів колекцій є масив, про масиви і колекції детальніше на наступних заняттях). Має такий синтаксис:

```
foreach(<тип> <ім'я_зміної> in <вираз_колекція>)
    оператор;
```

`ім'я_зміної` – ім'я змінної, яка використовується для доступу перебором до всіх елементів, заданих за допомогою `виразу_колекції`.

`тип` – тип даних змінної.

`оператор` – тіло циклу, виконується на кожній ітерації. Може замінюватись набором операторів у фігурних дужках.

Приклад:

```
int[] myArray = { 1, 3, 5, 8, 2 };
foreach (int i in myArray) {
    Console.WriteLine("i = " + i);
}
```

виведе на екран:

```
i = 1
i = 3
i = 5
i = 8
i = 2
```

Оператор циклу **foreach** діє наступним чином. Коли цикл починається, перший елемент масиву вибирається і присвоюється змінній циклу. На кожному наступному кроці ітерації вибирається наступний елемент масиву, який зберігається в змінній циклу. Цикл завершується, коли всі елементи масиву виявляться вибраними. Отже, оператор **foreach** циклічно опитує масив по окремих його елементів від початку і до кінця.

Слід зазначити, що змінна циклу в операторі **foreach** використовується тільки для читання. Це означає, що, привласнюючи цій змінній нове значення, не можна змінити вміст масиву.

```
int[] myArray = { 1, 3, 5, 8, 2 };
foreach (int i in myArray) {
    i = 0;
}
```

Наведений код, призведе до помилки на етапі компіляції: «Не можна присвоювати значення змінній ‘i’, бо вона являється ітераційною змінною циклу **foreach**».

Нижче наведено простий приклад застосування оператора циклу **foreach**. У цьому прикладі спочатку створюється цілочисельний масив, значення якого вводяться з клавіатури, потім ці значення виводяться на консоль у форматованому вигляді, а також обчислюється їх сума (рис. 11.1).

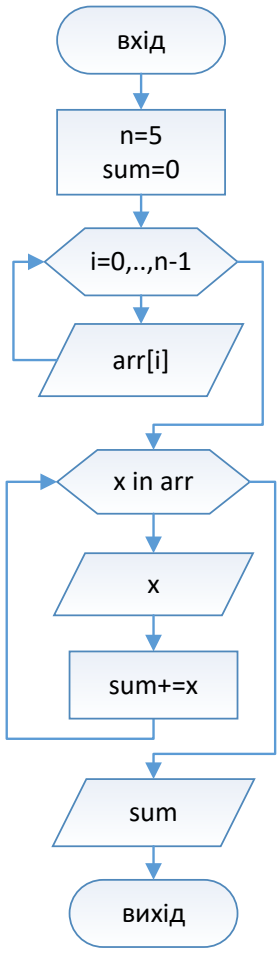


Рисунок 11.1 - Застосування циклу foreach

```
using System;

class ForeachExample {
    static void Main() {
        const int N = 5;
        int sum = 0;
        int[] arr = new int[N];
        Console.WriteLine("Введіть {0} цілих чисел", N);
        // Задати початкові значення елементів масиву nums.
        for (int i = 0; i < N; i++)
            arr[i] = int.Parse(Console.ReadLine());
        Console.WriteLine("Введено масив");
```

```
            // Використовувати цикл foreach для виведення значень
            // елементів масиву та підрахунку їх суми.
            foreach (int x in arr) {
                Console.Write("{0,4}", x);
                sum += x;
            }
            Console.WriteLine("\nСума елементів:" + sum);
            Console.ReadKey();
        }
}
```

Виконання наведеного вище коду дає наступний результат:

```
Введіть 5 цілих чисел
121
23
465
22
777
Введено масив
 121   23 465   22 777
Сума елементів:1408
```

**Контрольні запитання**

1. Поясніть принцип роботи циклу **foreach...in**.
2. Наведіть синтаксис циклу **foreach...in**.
3. Наведіть приклад використання циклу **foreach...in**.

# 12. МАСИВИ. ЗАПОВНЕННЯ МАСИВІВ ЗА ДОПОМОГОЮ ГЕНЕРАТОРА ВИПАДКОВИХ ЧИСЕЛ. ОСНОВНІ ПРИНЦИПИ ВИКОРИСТАННЯ КЛАСУ SYSTEM.ARRAY.

## 12.1. Клас Random

Клас **Random** – реалізує генератор псевдовипадкових чисел, тобто це пристрій, який видає послідовність чисел, яка задовольняє певним статистичним критеріям ймовірності. Методи цього класу наведено в табл. 12.1.

Таблиця 12.1 – Основні методи класу Random

| Метод | Пояснення |
|---|---|
| Next() | Повертає невід'ємне 32-бітне ціле число зі знаком, яке більше або рівне **0** та менше **int.MaxValue** |
| Next(Int32) | Повертає невід'ємне 32-бітне ціле число, що менше за задане значення |
| Next(Int32, Int32) | Повертає 32-бітне ціле число у вказаному діапазоні не включаючи верхню межу |
| NextBytes(Byte[]) | Заповнює елементи масиву байтів випадковими числами |
| NextDouble() | Повертає дійсне число подвійної точності з плаваючою крапкою з діапазону [0, 1) |
| Sample() | Повертає дійсне число подвійної точності з плаваючою крапкою з діапазону [0, 1) |

Розглянемо, як заповнити масиви цілих і дійсних чисел значеннями від -20 до 20.

```
using System;
class Program {
    static void Main(string[] args) {
        const int n = 10;

        int[] a = new int[n];
        double[] b = new double[n];
        Random rnd = new Random();
        // Для цілих
        Console.WriteLine("Int Array");
        for (int i = 0; i < n; i++) {
            a[i] = rnd.Next(-20, 21);
        }
```

```
            for (int i = 0; i < n; i++) {
                Console.Write("{0,5}", a[i]);
            }
            Console.WriteLine();
            Console.WriteLine("Double Array");
            // Для дійсних
            for (int i = 0; i < n; i++) {
                b[i] = rnd.NextDouble() * 40 - 20;
                Console.Write("{0,8:F3}", b[i]);
            }
            Console.ReadKey();
        }
}
```

Перевіримо результат:

```
Int Array
   -5  -19  -11   -3   15   10  -20  -11  -18    9
Double Array
  -1,147 -13,281  -4,384  15,375 -18,534 -10,417  -8,790  -1,375
10,723   9,397
```

## 12.2. Клас Array

Клас **Array** надає методи для створення, зміни, пошуку і сортування масивів, тобто він є базовим класом для всіх масивів.

Властивості:

**Length** – повертає кількість елементів масиву;

**Rank** – повертає розмірність масиву (1 для одновимірного і т.д.).

Розглянемо приклад програми для виводу на екран масиву за допомогою циклу **for**:

```
using System;
class Program {
    static void Main() {
        int[] arr = { 1, 3, 5, 8, 2 };
        for (int i = 0; i < arr.Length; i++) {
            Console.WriteLine("arr[{0}]={1}", i,
arr[i]);
        }
        Console.ReadKey();
    }
}
```

Ця програма виведе на екран наступне:

```
arr[0]=1
```

```
arr[1]=3
arr[2]=5
arr[3]=8
arr[4]=2
```

### 12.3. Цикл foreach

В цій програмі можна замінити цикл **for** на **foreach**:

```
foreach (int i in arr) {
    Console.WriteLine(i);
}
```

але слід враховувати, що у випадку циклу **foreach** ітераційну змінну можна використовувати тільки для читання.

### 12.4. Методи класу Array

Основні методи класу **Array** наведено в таблиці 12.2

Таблиця 12.2 – Методи класу Array

| Метод | Пояснення |
|---|---|
| Clear(Array, Int32, Int32) | Статичний. Присвоює елементам масиву 0, false або null в залежності від типу елементів |
| Clone() | Створює повну копію масиву разом зі значеннями |
| Copy(Array, Array, Int32) | Статичний. Копіює заданий діапазон елементів з одного масиву в інший |
| IndexOf(Array, Object) | Виконує пошук указаного об'єкту в середині одновимірного масиву і повертає індекс першого його входження або -1, якщо значення не знайдено. |
| LastIndexOf(Array, Object) | Виконує пошук указаного об'єкту в середині одновимірного масиву і повертає індекс останнього його входження або -1, якщо значення не знайдено. |
| Sort(Array) | Статичний. Сортує елементи одновимірного масиву Array, використовуючи реалізацію інтерфейсу IComparable для кожного елементу Array |
| Reverse(Array) | Змінює порядок слідування елементів всього масиву |

| | |
|---|---|
| | Array на протилежний |
| GetLength(Int32) | Повертає ціле число, що відповідає кількості елементів у заданому вимірі масиву Array |

Приклад 1. Поміняти перший і останній елементи масиву місцями, розвернути масив:

```
using System;
class Program {
    static void Main(string[] args) {
        Random rnd = new Random();
        int[] a = new int[9];
        for (int i = 0; i < a.Length; i++) {
            a[i] = rnd.Next(-20, 21);
        }
        Console.WriteLine("На початку");
        for (int i = 0; i < a.Length; i++) {
            Console.Write("{0,4}", a[i]);
        }
        // Міняємо перший з останнім
        int x = a[0];
        a[0] = a[a.Length - 1];
        a[a.Length - 1] = x;
        Console.WriteLine("\nПісля заміни");
        for (int i = 0; i < a.Length; i++) {
            Console.Write("{0,4}", a[i]);
        }
        // Розворот за допомогою циклу
        for (int i = 0; i < a.Length / 2; i++) {
            x = a[i];
            a[i] = a[a.Length - 1 - i];
            a[a.Length - 1 - i] = x;
        }
        Console.WriteLine("\nРозворот циклом");
        for (int i = 0; i < a.Length; i++) {
            Console.Write("{0,4}", a[i]);
        }
        // Розворот за допомогою методу Reverse
        Array.Reverse(a);
        Console.WriteLine("\nРозворот методом");
        foreach (int i in a) {
            Console.Write("{0,4}", i);
```

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

Результат:

```
На початку
   7  -7  20 -17  13 -15  -6 -10  -1
Після заміни
  -1  -7  20 -17  13 -15  -6 -10   7
Розворот циклом
   7 -10  -6 -15  13 -17  20  -7  -1
Розворот методом
  -1  -7  20 -17  13 -15  -6 -10   7
```

Приклад 2. Знайти індекси всіх елементів == 5

```
using System;
class Program {
    static void Main(string[] args) {
        Random rnd = new Random();
        int[] a = new int[9];
        for (int i = 0; i < a.Length; i++) {
            a[i] = rnd.Next(3, 8);
        }
        foreach (int i in a) {
            Console.Write("{0,4}", i);
        }
        int j = 0, k;
        Console.WriteLine();
        while ((k = Array.IndexOf(a, 5, j)) > -1) {
            Console.Write("{0,4}", k);
            j = k + 1;
        }
        Console.ReadKey();
    }
}
```

Результат роботи програми:

```
   5   7   4   6   5   5   3   4   5
   0   4   5   8
```

**Контрольні запитання**

1. Поясніть суть класу **`Random`** та наведіть його основні методи.
2. Поясніть суть класу **`Array`** та наведіть його основні методи і властивості.
3. Які особливості заміни циклу **`for`** на **`foreach`**?
4. Наведіть приклад заповнення одновимірного масиву.

# 13. ПРИНЦИПИ ОБРОБКИ ДАНИХ В ОДНОВИМІРНИХ МАСИВАХ. МЕТОДИ СОРТУВАННЯ ТА ПОШУКУ ДАНИХ.

## 13.1. Бульбашкове сортування

Бульбашкове сортування — це найпростіший алгоритм сортування. Він проходить по масиву кілька разів, на кожному етапі переміщуючи найбільше значення з невідсортованих у кінець масиву.

Наприклад, у нас є масив цілих чисел:

**7 10 5 8 7**

При першому проході по масиву ми порівнюємо значення 7 і 10. Оскільки 10 більше 7, ми залишаємо їх як є. Після чого порівнюємо 10 і 5. 5 менше 10, тому ми міняємо їх місцями, переміщуючи 10 на одну позицію ближче до кінця масиву.

**7 5 10 8 7**

Аналогічно міняємо місцями 10 і 8, та 10 і 7. Тепер масив виглядає так:

**7 5 8 7 10**

Десятка зайняла останнє місце найбільшого елементу масиву.

Оскільки був зроблений принаймні один обмін значень, нам потрібно пройти по масиву ще раз. Але розглядатимемо тепер масив розмірності на один менше.

**7 5 8 7**

При проході по цьому масиву 5 міняється місцями з 7, а потім 8 з 7.

**5 7 7 8**

Вісім – на своєму місці. І знову був зроблений як мінімум один обмін, а виходить, проходимо по масиву розміром на один менше ще раз.

**5 7 7**

При наступному проході обміну не проводиться, що означає, що наш масив відсортований, і алгоритм закінчив свою роботу.

Блок-схема алгоритму показана на рис. 13.1.

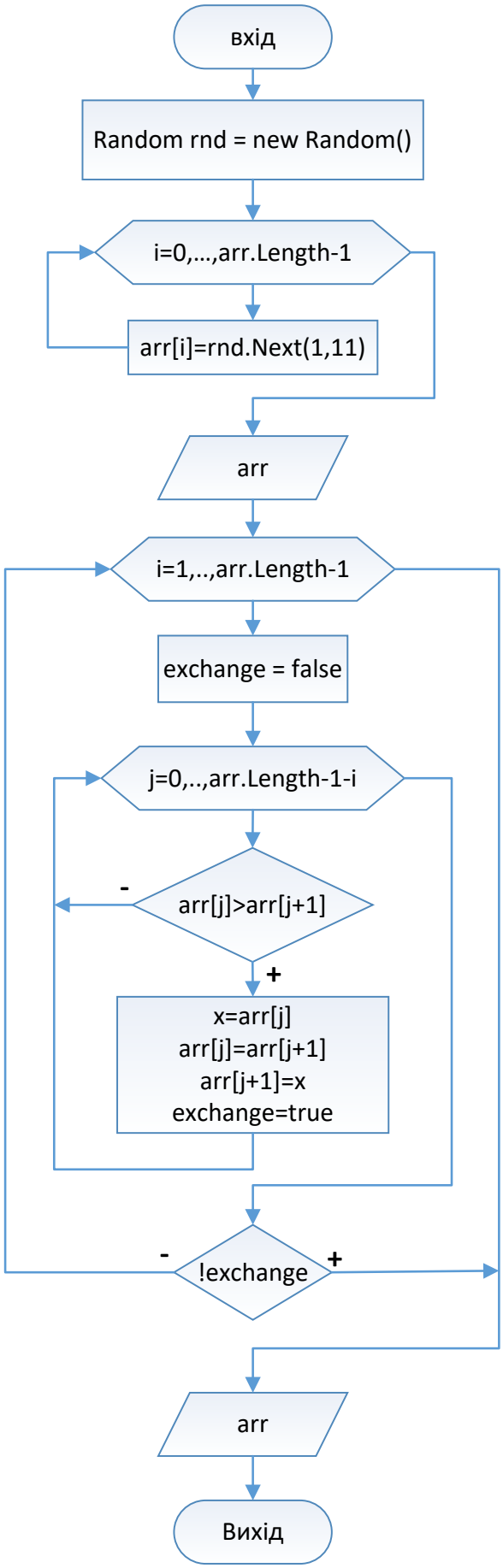


Рисунок 13.1 – Алгоритм бульбашкового сортування

Запрограмувавши цей алгоритм, отримаємо наступний текст програми:

```
using System;

class Program {
    static void Main(string[] args) {
        int[] arr = new int[5];

        Random rnd = new Random();

        for (int i = 0; i < arr.Length; i++) {
            arr[i] = rnd.Next(1, 11);
        }
        Console.WriteLine("До сортування");
        for (int i = 0; i < arr.Length; i++) {
            Console.Write("{0,5}", arr[i]);
        }


        for (int i = 1; i < arr.Length; i++) {
            // Чи були обміни
            bool exchange = false;
            for (int j = 0; j < arr.Length - i; j++) {
                if (arr[j] > arr[j + 1]) {
                    int x = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = x;
                    exchange = true;
                }
            }
            // Обмінів не було виходимо
            if (!exchange) break;
        }
        Console.WriteLine("\nПісля сортування");
        for (int i = 0; i < arr.Length; i++) {
            Console.Write("{0,5}", arr[i]);
        }

        Console.ReadKey();
    }
}
```

Перевіривши роботу отримаємо:

```
До сортування
```

```
    7   10    5    8    7
Після сортування
    5    7    7    8   10
```

### 13.2. Сортування вставками

Переміщуючись по масиву зліва на право, на кожному кроці алгоритму ми вибираємо один з елементів масиву і вставляємо його на потрібну позицію у вже відсортованій частині масиву доти, доки набір вхідних даних не буде вичерпано.

Розглянемо конкретний приклад:

**9 4 8 10 4**

Крок 1. Вибираємо другий елемент – це 4, порівнюємо з 9, дев'ять більше, отже перемінюємо дев'ять праворуч на одну позицію, а чотири опинитися на першій. Перший крок завершено, два перших елементи відсортовані.

**4 9 8 10 4**

Крок 2. Вибираємо наступний елемент, це 8. Вісім менше дев'яти, дев'ять іде праворуч. Порівнюємо вісім з 4, вісім більше отже записуємо 8 праворуч від 4. Відсортовано 3 елементи масиву.

**4 8 9 10 4**

Крок 3. Наступний елемент 10. Десять більше за дев'ять тому залишається на своєму місці. Маємо чотири посортовані елементи.

**4 8 9 10 4**

Крок 4. Останній необроблений елемент 4. Він менше десяти, дев'яти та восьми, тому всі вони ідуть праворуч на одну позицію, а чотири займе місце 8. Весь масив посортовано.

**4 4 8 9 10**

Блок-схема алгоритму показана на рис. 13.1.

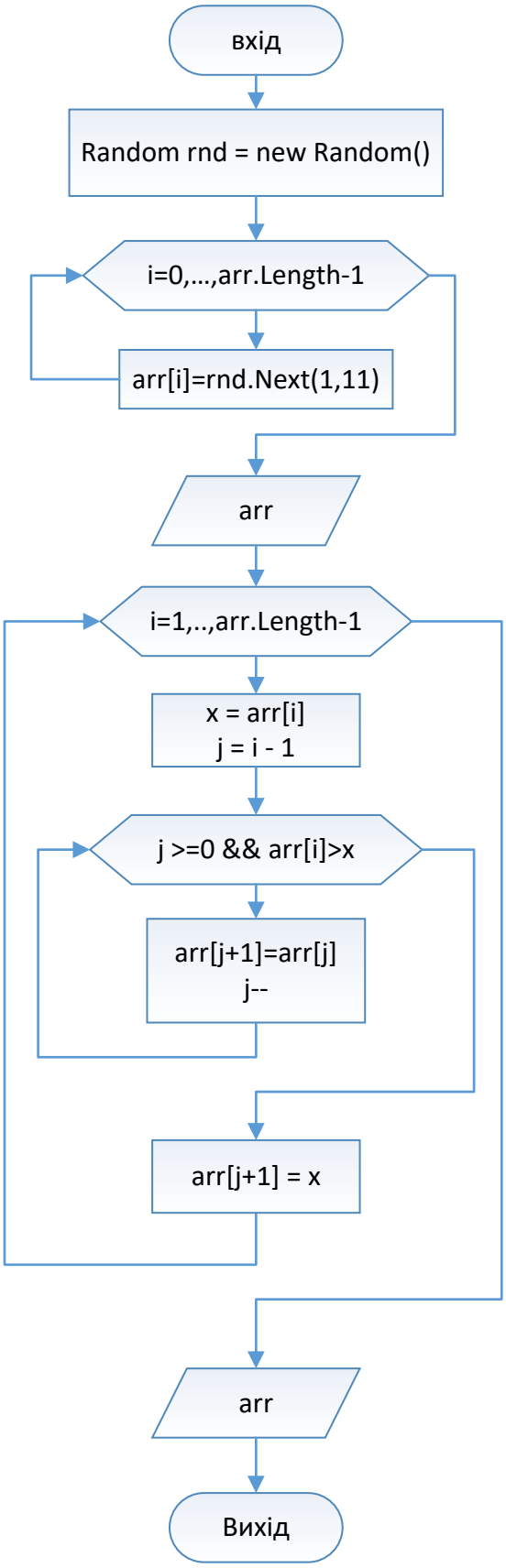


Рисунок 13.1 – Алгоритм сортування вставками

Програма для методу сортування вставками може мати наступний вигляд:

```
using System;

class Program {
    static void Main(string[] args) {
        int[] arr = new int[5];

        Random rnd = new Random();

        for (int i = 0; i < arr.Length; i++) {
            arr[i] = rnd.Next(1, 11);
        }
        Console.WriteLine("До сортування");
        foreach (int a in arr) {
            Console.Write("{0,5}", a);
        }

        for (int i = 1; i < arr.Length; i++) {
            int x = arr[i];
            int j;
            for (j = i - 1; j > -1 && arr[j] > x; j--) {
                arr[j + 1] = arr[j];
            }
            arr[j + 1] = x;
            Console.WriteLine("\nКрок " + i);
            foreach (int a in arr) {
                Console.Write("{0,5}", a);
            }
        }

        Console.WriteLine("\nПісля сортування");
        foreach (int a in arr) {
            Console.Write("{0,5}", a);
        }

        Console.ReadKey();
    }
}
```

Результат роботи програми з виводом проміжних результатів:

```
До сортування
    9    4    8   10    4
Крок 1
```

```
    4    9    8   10    4
Крок 2
    4    8    9   10    4
Крок 3
    4    8    9   10    4
Крок 4
    4    4    8    9   10
Після сортування
    4    4    8    9   10
```

**Контрольні запитання**

1. Поясніть принцип виконання бульбашкового сортування масиву.
2. Поясніть принцип виконання сортування масиву вставками.
3. Сформуйте та назвіть відмінність бульбашкового сортування від сортування вставками.
4. Наведіть приклад бульбашкового сортування.
5. Наведіть приклад сортування вставками.

# 14. ВИКОРИСТАННЯ МЕТОДІВ SPLIT І JOIN ПРИ РОБОТІ З РЯДКАМИ

Після знайомства з масивами, доречно розглянути методи **Split** і **Join** класу **string**, які можуть полегшити розв'язання ряду задач з обробки текстової інформації.

## 14.1. Метод Split

Метод **String.Split** з рядка створює масив підрядків за одним або декількома роздільниками. Досить часто, це найпростіший спосіб розбити речення або текст на слова. Проте, він також може використовуватися для розбиття рядків по іншим символам або рядкам.

В найпростішому випадку в якості роздільника можна вказати пробіл:

```
using System;

class Program {
    static void Main() {
        string phrase =
"Обмін досвідом, цікаві воркшопи, надактуальні теми.";

        string[] words = phrase.Split(' ');

        foreach (var word in words) {
            System.Console.WriteLine(word);
        }

        Console.ReadKey();
    }
}
```

При виконанні прикладу отримаємо:

```
Обмін
досвідом,
цікаві
воркшопи,
надактуальні
теми.
```

Ряд слів при розбитті отримали коми та крапки. Щоб позбутися зайвих символів в словах, можна вказати їх в якості роздільників.

```
        string phrase =
"Обмін досвідом, цікаві воркшопи, надактуальні теми.";
        string[] words = phrase.Split(' ', '.', ',');
```

```
        foreach (var word in words) {
            System.Console.WriteLine(word);
        }
```

Але при виконанні цього варіанту з’являться порожні рядки.

```
Обмін
досвідом

цікаві
воркшопи

надактуальні
теми
```

Причина в тому, що коли йде два довільних роздільника підряд, в результуючому масиві створюється елемент, що містить порожній рядок.

Для того, щоб в результуючий масив не включалися порожні рядки, в методі **Split** потрібно вказати необов’язковий параметр – **StringSplitOptions.RemoveEmptyEntries**, а роздільники вказати у вигляді масиву символів.

```
using System;

class Program {
    static void Main() {
        string phrase =
"Обмін досвідом, цікаві воркшопи, надактуальні теми.";

        char[] chars = { ' ', '.', ',' };
        string[] words =
phrase.Split(chars, StringSplitOptions.RemoveEmptyEntries);

        foreach (var word in words) {
            System.Console.WriteLine(word);
        }

        Console.ReadKey();
    }
}
```

В результаті отримаємо слова:

```
Обмін
```

```
досвідом
цікаві
воркшопи
надактуальні
теми
```

### 14.2. Метод Join

Метод **String.Join** виконує функцію протилежну до методу **Split**, він зчіплює елементи масиву в один рядок, вставляючи між ними вказаний роздільник.

```
using System;

class Program {
    static void Main() {
        string [] words = {"This", "is", "an", "example"};
        int[] numbers = {10, 22, 44, 94};
        // Для слів
        Console.WriteLine(string.Join(" ", words));
        // Для чисел
        Console.WriteLine(string.Join(",", numbers));
        Console.ReadKey();
    }
}
```

Ця програма виведе на екран наступне:

```
This is an example
10,22,44,94
```

Як видно з прикладу, **Join** може з'єднувати не тільки масиви рядків, а й масиви чисел, або навіть довільних об'єктів.

### 14.3. Приклади використання Split та Join

Приклад 1. Ввести з клавіатури два речення. Для першого вивести останнє слово, а для другого – перше.

```
using System;

class Program {
    static void Main(string[] args) {
        string a = Console.ReadLine();
        string b = Console.ReadLine();
```

```
        char[] ch = { ' ', ',', '.', '-' };
        string[] wordsA = a.Split(ch,
StringSplitOptions.RemoveEmptyEntries);
        string[] wordsB = b.Split(ch,
StringSplitOptions.RemoveEmptyEntries);
        if (wordsA.Length > 0) {
            Console.WriteLine(wordsA[wordsA.Length - 1]);
        }
        if (wordsB.Length > 0) {
            Console.WriteLine(wordsB[0]);
        }
        Console.ReadKey();
    }
}
```

Приклад 2. Знайти всі слова в реченні розділені пробілами, комами, дефісами. Відсортувати ці слова за алфавітом та записати в рядок розділивши «;».

```
using System;

namespace StrSplit {
    class Program {
        static void Main(string[] args) {
            string s = "Знайти всі слова в реченні розділені
пробілами, комами, дефісами.";

            string[] words = s.Split(new char[] { ' ', ',' , '-
', '.'}, StringSplitOptions.RemoveEmptyEntries);

            Console.WriteLine("--WORDS--");
            foreach (string word in words) {
                Console.WriteLine(word);
            }

            Array.Sort(words);
            Console.WriteLine("--SORTED--");
            foreach (string word in words) {
                Console.WriteLine(word);
            }

            Console.WriteLine("--STRING--");
            Console.WriteLine(string.Join(";", words));
            Console.ReadKey();
```

```
        }
    }
}
```

Результат роботи програми

```
--WORDS--
Знайти
всі
слова
в
реченні
розділені
пробілами
комами
дефісами
--SORTED--
в
всі
дефісами
Знайти
комами
пробілами
реченні
розділені
слова
--STRING--
в;всі;дефісами;Знайти;комами;пробілами;реченні;розділені;слова
```

**Контрольні запитання**

1. Наведіть приклад роботи з методом **`Split`** та поясніть принцип його роботи.
2. Чому при розбитті тексту на підрядки методом **`Split`** можуть виникнути порожні рядки?
3. Поясніть принцип роботи методу **`Join`**.
4. Наведіть формат запису методів **`Split`** та **`Join`**.

## 15. БАГАТОВИМІРНІ ПРЯМОКУТНІ МАСИВИ.

***Багатовимірний масив*** – це масив який визначається двома або більше вимірами, а до його елементів треба звертатися за домомогою двох або більше індексів.

### 15.1. Двовимірні прямокутні масиви

Найпростіші багатовимірні масиви – двовимірні прямокутні масиви (або матриці)[5]. Для їх оголошення використовують наступний синтаксис:

**<тип>[,] <ім'я_масиву>;**

де **тип** – тип даних, **ім'я_масиву** – ім'я масиву за яким в подальшому можна звертатись до його елементів.

Для створення об'єкту типу двовимірний масив можна використовувати наступний синтаксис:

**<тип>[,] <ім'я_масиву> = new <тип>[<розм_0>,<розм_1>};**

Приклад створення матриці 5 на 7:

```
int[,] matrix = new int[5, 7];
```

Для числових типів даних елементи масиву автоматично заповняться нулями. До цих елементів можна звертатися наступним чином:

```
matrix[0,0] = 7;  //лівий верхній елемент
matrix[3,5] = 10; //десь всередині
matrix[4,6] = 9;  //правий нижній елемент
```

Тепер розглянемо як заповнити та вивести на екран прямокутну матрицю за допомогою циклів:

```
using System;

class Program {
    static void Main() {
        //створюємо матрицю 5x7
        const int M = 5;
        const int N = 7;
        int[,] matrix = new int[M, N];
        //заповнюємо елементи матриці сумою індексів
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
                matrix[i, j] = i + j;
            }
        }
        //виведемо отримані значення на екран
        for (int i = 0; i < M; i++) {
```

```
            for (int j = 0; j < N; j++) {
                Console.Write("{0,3}", matrix[i, j]);
            }
            Console.WriteLine();
        }
        Console.ReadKey();
    }
}
```

Блок-схема цієї програми представлена на рис. 15.1

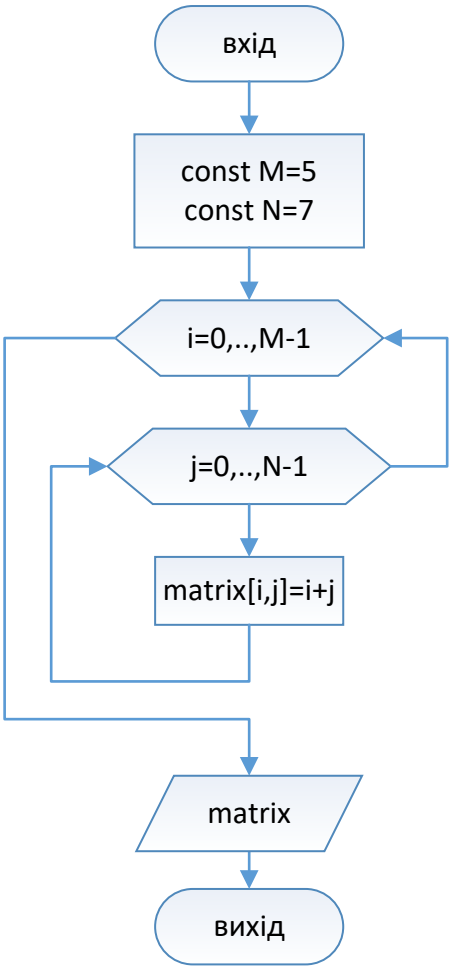


Рисунок 15.1 – Робота з прямокутною матрицею

На екрані отримаємо:

```
 0  1  2  3  4  5  6
 1  2  3  4  5  6  7
 2  3  4  5  6  7  8
 3  4  5  6  7  8  9
 4  5  6  7  8  9 10
```

Зверніть увагу, що в цьому випадку для зручності роботи розмірність матриці задана за допомогою числових констант, що дозволяє в разі необхідності змінити значення розмірності лише в одному місці програми.

Якщо значення елементів матриці відомі заздалегідь, її можна ініціалізувати використовуючи наступний синтаксис:

```
int[,] m = {{ 1, 1, 1}, { 2, 2, 2}};
```

В цьому випадку компілятор сам визначить, що треба створити матрицю з двох рядків і трьох стовпчиків і заповнить перший рядок одиницями, а другий двійками.

## 15.2. Властивості та методи матриць

Двовимірні масиви успадковують ті самі методи та властивості класу **Array**, що й одновимірні, проте слід враховувати особливості їх застосування.

**Length** – повертає загальну кількість елементів матриці (для матриці 5x7 – це буде 35) ;

**Rank** – повертає розмірність масиву (для матриці це буде 2).

Щоб дізнатися кількість рядків та стовпчиків краще використовувати метод **GetLength(Int32)** зі значенням параметра 0 для рядків, 1 для стовпчиків.

Приклад:

```
using System;
class Program {
    static void Main() {
        int[,] arr = new int[3,5];
        Console.WriteLine("Length =       {0,2}", arr.Length);
        Console.WriteLine("Rank =         {0,2}", arr.Rank);
        Console.WriteLine("GetLength(0) = {0,2}",
arr.GetLength(0));
        Console.WriteLine("GetLength(1) = {0,2}",
arr.GetLength(1));
        Console.ReadKey();
    }
}
```

Поверне:

```
Length =       15
Rank =          2
GetLength(0) =  3
GetLength(1) =  5
```

### 15.3. Приклади роботи з двовимірними прямокутними масивами

Приклад 1.

Заповнити матрицю розмірності 7x7 випадковими цілими числами в діапазоні від -20 до 20, відобразити її на екрані, та знайти значення найбільшого елемента під головною діагоналлю (рис. 15.2). Для розв'язання задачі, слід врахувати, що елементи на головній діагоналі мають однакові індекси рядка та стовпчика, а для елементів під головною діагоналлю – індекс стовпчика буде меншим за індекс рядка.

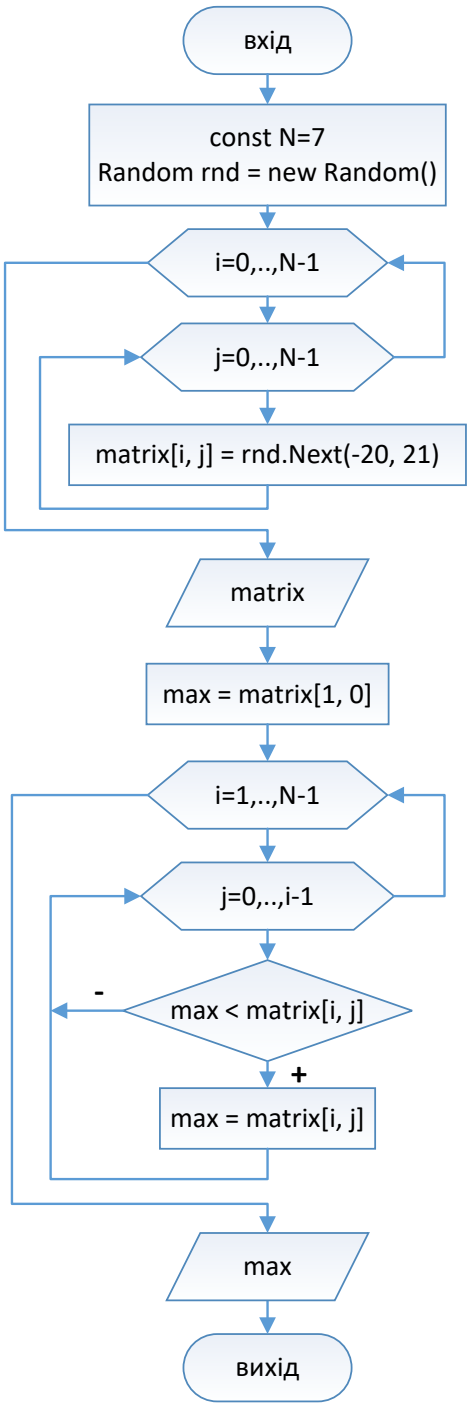


Рисунок 15.2 – Алгоритм прикладу 1

Програма:

```
using System;

class Program {
    static void Main() {
        const int N = 7;
        // Створюємо матрицю 10x10
        int[,] matrix = new int[N, N];
        // Заповнюємо її випадковими числами
        Random rnd = new Random();
        for (int i = 0; i < matrix.GetLength(0); i++) {
            for (int j = 0; j < matrix.GetLength(1); j++) {
                matrix[i, j] = rnd.Next(-20, 21);
            }
        }
        // Виводимо матрицю на консоль
        for (int i = 0; i < matrix.GetLength(0); i++) {
            for (int j = 0; j < matrix.GetLength(1); j++) {
                Console.Write("{0,4}", matrix[i, j]);
            }
            Console.WriteLine();
        }
        // Знаходимо максимальний під діагоналлю
        int max = matrix[1, 0];
        for (int i = 1; i < matrix.GetLength(0); i++) {
            for (int j = 0; j < i; j++) {
                if (max < matrix[i, j]) max = matrix[i, j];
            }
        }
        Console.WriteLine("Max={0}", max);
        Console.ReadKey();
    }
}
```

Результат:

```
   0  17   0  12   7 -17   6
   5  -3  11  19 -10   7  -7
 -15   0   9  -9   3  -4  14
   7  -2  11 -18  14  20   2
  15   4  18   8   2  -4   4
  -4  -4  19   0   2  14   0
   7  -7 -16  14  11   2  18
Max=19
```

Приклад 2.

Заповнити матрицю A розмірності nxn випадковими цілими числами в діапазоні від -20 до 20, відобразити її на екрані, та поміняти елементи зафарбованих областей місцями (рис. 15.3).

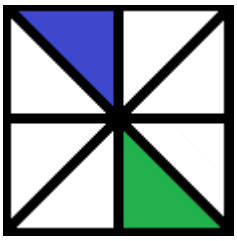

Рисунок 15.3 – Поміняти місцями елементи

Елементи верхнього трикутника знаходяться над головною діагоналлю та обмежені серединою матриці, отже для їх індексів виконуються умови:

$$i = \overline{0,k} \quad j = \overline{i,k} \quad \text{де } k = (n-1)/2$$

Обмін елементів матриці буде виконуватись симетрично відносно центру матриці за правилом:

$$a_{0,0} \leftrightarrow a_{n-1,n-1}$$
$$a_{0,1} \leftrightarrow a_{n-1,n-2}$$
$$a_{1,1} \leftrightarrow a_{n-2,n-2}$$
$$a_{1,2} \leftrightarrow a_{n-2,n-3}$$

Не важко помітити загальне правило обміну:

$$a_{i,j} \leftrightarrow a_{n-1-i,n-1-j}$$

Враховуючи вище наведені формули, програма матиме вигляд:

```
using System;
class Program {
    static void Main() {
        const int n = 7;
        // Створюємо матрицю 10x10
        int[,] a = new int[n, n];
        // Заповнюємо її випадковими числами
        Random rnd = new Random();
        for (int i = 0; i < a.GetLength(0); i++) {
            for (int j = 0; j < a.GetLength(1); j++) {
                a[i, j] = rnd.Next(-20, 21);
            }
        }
        // Виводимо матрицю на консоль
        for (int i = 0; i < a.GetLength(0); i++) {
            for (int j = 0; j < a.GetLength(1); j++) {
                Console.Write("{0,4}", a[i, j]);
```

```
            }
            Console.WriteLine();
        }
        // Міняємо елементи місцями
        int k = (n - 1) / 2;
        for (int i = 0; i <= k; i++) {
            for (int j = i; j <= k; j++) {
                int x = a[i, j];
                a[i, j] = a[n - 1 - i, n - 1 - j];
                a[n - 1 - i, n - 1 - j] = x;
            }
        }
        // Виводимо матрицю на консоль
        Console.WriteLine("----------------------------");
        for (int i = 0; i < a.GetLength(0); i++) {
            for (int j = 0; j < a.GetLength(1); j++) {
                Console.Write("{0,4}", a[i, j]);
            }
            Console.WriteLine();
        }
        Console.ReadKey();
    }
}
```

Результат роботи програми:

```
  -8  15   1  10  11 -14  17
 -15   3  -7  12  14  -5  15
 -12  -9   4  12   2   0 -13
   1  15 -16  -3 -17   9  12
   7  -3  18 -18   8   3  -1
 -14  12 -11   3 -11 -18  11
   7  12   3  18  12  -7   9
----------------------------
   9  -7  12  18  11 -14  17
 -15 -18 -11   3  14  -5  15
 -12  -9   8 -18   2   0 -13
   1  15 -16  -3 -17   9  12
   7  -3  18  12   4   3  -1
 -14  12 -11  12  -7   3  11
   7  12   3  10   1  15  -8
```

Як бачимо, блакитні та зелені елементи матриці помінялися місцями, а решта - залишилися на своїх місцях.

### 15.4. Прямокутні масиви трьох і більше вимірів

Аналогічно до двовимірних можна створювати тривимірні, чотиривимірні і т.д. масиви[5]:

```
int[, ,]matrix3 = new int[5, 7, 5];
int[, , ,]matrix4 = new int[4, 4, 3, 3];
```

Нижче наведено загальна форма оголошення багатовимірного масиву.

```
<тип>[, … ,] <ім'я_масиву> = new <тип>[<розм_0>, … ,
<розм_N>};
```

Приклад, створити тривимірний прямокутний масив 4x4x4, значення елементів якого рівні сумі відповідних координат. Знайти суму елементів, що знаходяться на його головній діагоналі.

Розв'язання:

```
using System;

class Program {
    static void Main() {
        const int N = 4;
        int[, ,] arr = new int[N, N, N];
        // Заповнення
        for (int i = 0; i < arr.GetLength(0); i++) {
            for (int j = 0; j < arr.GetLength(1); j++) {
                for (int k = 0; k < arr.GetLength(2); k++) {
                    arr[i, j, k] = i + j + k;
                }
            }
        }
        // Сума діагональних
        int sum = 0;
        for (int i = 0; i < N; i++) {
            sum += arr[i, i, i];
        }
        Console.WriteLine("Sum={0}",sum);
        Console.ReadKey();
    }
}
```

**Контрольні запитання**

1. Що таке багатовимірний масив?
2. Наведіть синтаксис оголошення двовимірного масиву та форму запису для створення об’єкту типу двовимірний масив.
3. Назвіть основні властивості та методи матриць.
4. Наведіть фрагмент коду, що реалізує заповнення двовимірного масиву.
5. Наведіть загальну форму оголошення багатовимірного масиву.

# 16. СТУПІНЧАСТІ МАСИВИ

## 16.1. Двовимірні ступінчасті масиви

***Ступінчасті або зубчасті масиви*** **(**jagged array**)** – це масиви масивів, елементи яких є одновимірними масивами. Ступінчастими їх називають через те, що їх рядки можуть мати різну довжину. Ступінчасті масиви оголошуються наступним чином[5]:

```
<тип>[][] <ім'я_масиву>;
```

Щоб створити двовимірний ступінчастий масив, спочатку треба задати кількість рядків, наприклад:

```
int[][] arrJ = new int[3][];
```

потім задати розмірність кожного з рядків:

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

звертатися до елементів масиву потрібно так:

```
arrJ[0][0] = 2;
arrJ[0][1] = 4;
//...
arrJ[2][1] = 17;
```

## 16.2. Ініціалізація ступінчастих масивів

Для ініціалізації ступінчастих масивів наперед відомими значеннями, можна використовувати наступний синтаксис:

```
int[][] jagged = { new int[] { 1, 1 }, new int[] { 2, 2, 3 } };
```

Таким чином оголошений мачив, складатиметься з двох рядків, перший - довжиною два, а другий – три елементи.

## 16.3. Приклади застосування ступінчастих масивів

Часто ступінчасті масиви доцільніше використовувати замість прямокутних. Наприклад в методі Гауса для розв'язання системи лінійних рівнянь, якщо в матриці необхідно переставляти місцями рядки, то при використанні зубчастого масиву відпадає необхідність поелементного копіювання даних.

Приклад:

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

```
        //створюємо зубчасту матрицю 3x4

        float[][] arrJ = new float[3][];
        for (int i = 0; i < arrJ.Length; i++) {
            arrJ[i] = new float[4];
        }

        //заповнюємо елементи рядка його номером

        for (int i = 0; i < arrJ.Length; i++) {
            for (int j = 0; j < arrJ[i].Length; j++) {
                arrJ[i][j] = i;
            }
        }

        //міняємо місцями перший рядок з останнім

        float[] x = arrJ[0];
        arrJ[0] = arrJ[2];
        arrJ[2] = x;
    }
}
```

**Контрольні запитання**

1. Які масиви називають ступінчастими?
2. Запишіть синтаксис оголошення ступінчастих масивів.
3. Наведіть приклад звернення до елементів ступінчастих масивів.
4. Наведіть синтаксис ініціалізації ступінчастих масивів.
5. Наведіть приклад використання ступінчастих масивів.

## ЛІТЕРАТУРА


1. Вікіпедія : веб-сайт. URL: https://uk.wikipedia.org/
2. Кнут Д.Э. Искусство программирования. Том 1. Основные алгоритмы. / Кнут Дональд Эрвин (Donald Ervin Knuth). 3-е издание / Пер. с англ. – М: Издательский дом «Вильямс», 2002. - 720с.
3. Типы и переменные : веб-сайт. URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/types-and-variables
4. Руководство по языку C# : веб-сайт. URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/
5. Д.В. Настенко, А. Б. Нестерко. Об’єктно-орієнтоване програмування. Частина 1. Основи об’єктно-орієнтованого програмування на мові C#.: Навчальний посібник /– К.: НТУУ «КПІ», 2016. - 76с.
6. Павловская Т.А. C#. Программирование на языке высокого уровня. Учебник для вузов. – СПб.: Питер, 2009. – 432 с.
7. Char Struct (System) | Microsoft Docs : веб-сайт. URL: https://docs.microsoft.com/ru-ru/dotnet/api/system.char?view=netframework-4.8
8. String Class (System) | Microsoft Docs: веб-сайт. URL: https://docs.microsoft.com/ru-ru/dotnet/api/system.string?view=netframework-4.8
9. Логические операторы (справочник по C#): веб-сайт. URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/boolean-logical-operators#conditional-logical-and-operator-
10. Использование класса StringBuilder в .NET# : веб-сайт. URL: https://docs.microsoft.com/ru-ru/dotnet/standard/base-types/stringbuilder
11. Массивы (Руководство по программированию на C#) : веб-сайт. URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/arrays/
12. Джейсон Прайс. Visual C# .NET. Полное руководство. / Джейсон Прайс. Майк Гандэрлой.: пер. с англ. – К.: ВЕК+, СПб.: КОРОНА принт, К.:НТИ, М.: Энтроп, 2004. - 954 с. - ISBN 966-7140-36-9.
13. Троэлсен Э. C# и платформа .NET. Библиотека программиста. / Эндрю Троелсен – СПб.: Питер, 2007. -900с.- ISBN – 978-5-318-00750-7.
14. Бишоп, Джудит. C# в кратком изложении / Дж. Бишоп, Хорспул Н.; пер. с англ. К. Г. Финогенова. - Москва: Бином. Лаборатория знаний, 2005. - (Программисту). - 472 с.: ил. - ISBN 5-94774-211-X
15. Сухарев, М. В. Основы Delphi. Профессиональный подход / М. В. Сухарев. - СПб. : Наука и Техника, 2004. - 596 с. - ISBN 5-94387-129-2.

16. C# : [наиболее полн. рук. : пер. с англ.] / [Х .М. Дейтел, Пол Дж. Дейтел, Тэм Р. Нието и др.]. - СПб. : БХВ-Петербург, 2006. - 1056 с. : ил. - ISBN 5-94157-817-2
17. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 1 Пер. с англ. – М.: Издательско-торговый дом «Русская редакция», 2002. – 576с.:ил.- ISBN 5-7502-0210-0
18. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 2 Пер. с англ. – М.: Издательско-торговый дом «Русская редакция», 2002. – 624с.:ил.- ISBN 5-7502-0220-8
19. Лабор В.В. Си Шарп: Создание приложений для Windows./ Лабор В.В. Мн.: Харвест, 2003. – 385 с.- ISBN 985-13-1405-6.