Тема 1. Сутність програмування

1. **Алгоритми, мови програмування, програми та оболонки мов програмування.**
2. **Принципи виконання програм персональним комп'ютером. Еволюція розвитку мов програмування.**
3. **Класифікація мов програмування, оболонок мов програмування та програм.**
4. **Класифікація методів програмування.**
5. **Вимоги хорошого стилю програмування.**

1. Будь-яке завдання, що розв'язується на ПК – це процес обробки даних за допомогою алгоритму. **Дані** – це зареєстровані сигнали.

Основний об'єкт дослідження інформатики – інформація. **Інформація** – це продукт взаємодії даних і адекватних їм методів.

Алгоритм – це послідовність команд, які необхідно виконати для розв'язку поставленої задачі.

На ПК алгоритми реалізуються шляхом створення програм згідно стандартів мов програмування в оболонках мов програмування.

Мова програмування – це стандарт згідно якого створюється текст програм для реалізації алгоритмів на ПК. Ми будемо користуватися мовою програмування **C#**. Є ще, наприклад, такі мови програмування: Basic, Pascal, C, Java.

Програма – це сукупність інструкцій та команд для реалізації алгоритмів на ПК згідно синтаксису мови програмування. В C# вихідні тексти програм зберігаються у файлах з розширенням .cs.

Оболонка мови програмування – це середовище для створення, відлагодження, виконання чи перетворення програм у виконувані файли. Як правило, для кожної мови програмування розроблено декілька оболонок мов програмування. Ми працюватимемо в оболонці **Visual Studio Community 2013**.

Швидкість виконання створених програм в основному залежить від:

- тактової частоти процесора та інших характеристик апаратного забезпечення;
- типу операційної системи та інших характеристик системного забезпечення;
- якості алгоритмів, реалізованих в процесі розробки програми.

Тому для успішного розв'язання поставленої задачі потрібно:

1. Розробити логічну схему ефективного алгоритму розв'язку та переконатися, що її реалізація призведе до розв'язання задачі.
2. Описати алгоритм засобами мови програмування.
3. Врахувати всі обмеження мови програм для уникнення аварійного завершення.
4. Створити програму у вибраному середовищі та переконатися в її дієздатності.

2. Одиницею виміру інформації в комп'ютерах є байт. Кожен байт в різні моменти часу може містити команду чи її частину або фрагмент даних. Все залежить від, того як опрацьовується цей байт комп'ютером. На комп'ютерах виконуються файли з розширенням .exe, .com. Ці файли містять в собі машинні команди, які виконуються процесором. Крім цього, машинні команди містяться також у підпрограмах DLL-бібліотек.

При завантаженні виконуваного файлу автоматично формулюється 3 сегменти:

- сегмент даних *(там містяться дані);*
- сегмент коду*, де розташовується послідовність елементарних команд, які виконуються процесором;*
- сегмент стеку *(там розміщуються результати проміжних обчислень). Принцип заповнення стеку: останній зайшов – перший вийшов.*

Крім цього, в процесі виконання програми місце під нові дані може виділятися в динамічній пам'яті – "кучі".

Процесор здатний виконувати лише елементарні машинні команди (+,-,*,/, операції з регістрами, логічні операції і т.д). Якби доводилося програмувати на мові процесора, то розробка програми тривала б декілька років, а створена програма не змогла б працювати на комп'ютерах іншої архітектури. Тому для створення програм використовуються оболонки

мов програмування, які кожній команді ставлять у відповідність до десятків тисяч елементарних машинних команд.

В процесі виконання програм процесор послідовно зчитує і виконує команди з сегменту коду, обробляє дані з сегменту даних, а результат проміжних обчислень розміщує в сегменті стеку. Наприклад, при обчисленні значення виразу (9+7)*4 даними є числа 9, 7, 4; кодами – команди виконання послідовності обчислень, а в стеку тимчасово збережеться число 16.

Еволюція розвитку мов програмування (з Вікіпедії)

Власне перші мови програмування з'явилися задовго до появи перших комп'ютерів. Ще в 19-му столітті існували «програмовані» ткацькі верстати та піаніно-програвачі, спосіб програмування в яких нагадував так звані предметно-орієнтовані мови програмування. На початку 20-го століття починають використовуватись перфокарти, та механічна обробка даних. В 1930 −1940 рр. виникає лямбда-числення та машина Тюринга, які застосовували математичну абстракцію для опису алгоритмів.

В 1940 роках створюються перші електричні, двійкові комп'ютери. Вважається, що першу мову програмування високого рівня — Планкалькюль (нім. *Plankalkül*) розробив німець Конрад Цузе в період 1943–1945 років, але в той час вона не була реалізована і не привернула увагу. Реалізацією мови зайнялися і здійснили лише в 1998—2000 роках.

У кінці 40-их — початку 50-их застосовувалися інтерпретовані системи кодування, коли певні команди мови програмування кодувалися числами, які вже інтерпретувалися машинним кодом. Ці системи називалися «автоматичним програмуванням» і були простішими для програмування, ніж машинні коди, але могли мати значно меншу (до 50 разів) швидкодію, через що часто надавали перевагу машинним кодам. До таких систем належали Short Code для BINAC (1949) і UNIVAC I (1952), Speedcoding для IBM 701, розроблена Джоном Бекусом у 1954.

Першою широковживаною компільованою мовою став розроблений групою *Джона Бекуса* Фортран, анонсований у 1954 році і випущений у 1957 для IBM 704. Основним призначенням Фортрану були швидкі наукові обчислення, зокрема наголошувалося, що швидкодія згенерованого компілятором коду майже не відрізнятиметься від написаного вручну машинного коду. Вже у квітні 1958 близько половини програм для IBM 704 були написані на Фортрані. Випущений у 1958 році Фортран II дозволяв незалежну компіляцію підпрограм, що дало змогу створювати більші програми, оскільки низька надійність IBM 704 не дозволяла скомпілювати без збоїв велику програму (понад 300—400 рядків) одразу. Розроблений у 1960–1962 роках Фортран IV був однією з найпоширеніших мов того часу і лишався стандартною версією Фортрану до появи у 1978 році Фортрану 77.

У 1958 році у MIT розробили LISP — першу функціональну мову, яка понад чверть століття домінувала у програмуванні задач штучного інтелекту.

У кінці 1950-их почали розроблятися різні мови програмування. У 1958 році декілька значних груп комп'ютерних користувачів у США, включаючи SHARE — групу науковців-користувачів IBM і USE (UNIVAC) запропонували ACM і заснувати робочу групу зі створення універсальної мови програмування. Також ще у 1955 році німецьке Товариство прикладної математики і механіки (GAMM) заснувало комітет зі створення універсальної мови програмування. У кінці травня 1958 року було проведено зустріч у Цюриху між ACM і GAMM, на матеріалах якої у грудні опубліковано «ALGOL 58 Report». На його основі було створено 3 значні реалізації — MAD (1961), NELIAC (1963), JOVIAL (1963). З них лише JOVIAL отримав поширення, ставши на чверть століття офіційною мовою програмування у Військово-морських силах США. SHARE і IBM почали створення власної реалізації ALGOL, але припинили, врахувавши витрати на створення і просування Фортрану.

Впродовж 1959 року ALGOL 58 широко обговорювався, була запропонована нотація для опису синтаксису мов програмування — форма Бекуса-Наура. У 1960 було проведено чергову зустріч і опубліковано ALGOL 60 Report. ALGOL вплинув на багато мов програмування і став стандартною мовою для публікації алгоритмів, але через ряд причин не одержав широкого поширення – він був заскладним, і не було реалізацій, які підтримували його повністю, відсутність стандартного введення-виведення привела до появи різних несумісних реалізацій, деякі неоднозначності опису мови так і не були розв'язані. Також широкого вжитку уже набув Фортран, і IBM не підтримала ALGOL.

У 1959 році була проведена зустріч у Пентагоні для створення мови CBL (Common Business Language), засновано комітет з його створення, і у 1960 опубліковано початкову специфікацію COBOL 60, який невдовзі став першою мовою, прийнятою у Міністерстві оборони США. У 1968 році COBOL було стандартизовано ANSI.

У 1964 році було створено спрощену мову BASIC (Beginners All-purpose Symbolic Instruction Code) для навчання програмуванню студентів, які переважно спеціалізувалися у вільних мистецтвах, а не у технічних науках.

Тоді як науковці переважно використовували Фортран, а бізнес – COBOL, у 1963 році в IBM вирішили створити універсальні платформу IBM/360 і мову програмування. У стислі терміни до 1965 року було розроблено мову PL/I, яка поєднувала можливості Фортран, ALGOL і COBOL, і виявилася заскладною, хоча й була у широкому вжитку у 1970-их у наукових і бізнес задачах. Також її підмножини (PL/C, PL/CS) використовувалися для навчання програмуванню.

На початку 1960-их було створено перші мови із динамічною типізацією — APL і SNOBOL. SIMULA 67 була першою об'єктно-орієнтованою мовою програмування.

У 1965 році Ніклаус Вірт і Тоні Гоар запронували комітету з розвитку мови ALGOL свою версію, яку згодом назвали ALGOL-W і застосовували для навчання в деяких університетах. Пропозиція була відхилена через незначну кількість змін на користь значно складнішого ALGOL 68. У ALGOL 68 з'явилися визначення структур даних і динамічні масиви. ALGOL 68 став першою мовою із формальною специфікацією, яка, однак, була складною для розуміння.

У 1971 році Вірт опублікував опис мови Pascal, яка у 70-их стала загальновживаною для навчання студентів.

У 1972 році Деніс Річі розробив у Bell Labs мову C. Тоді ж у Марселі було створено інтерпретатор мови Пролог — першої і найвідомішої мови логічного програмування. Алан Кей у Xerox PARC розробив першу широко вживану об'єктно-орієнтовану мову – Smalltalk.

у 1973 Робін Мілнер в Единбурзькому університеті створив ML.

У 1975 році в Массачусетському технологічному інституті описано спрощений діалект мови Лісп – Scheme.

У 1976 випущено мову для статистичного програмування S, на базі якої в 1993 році створено R.

У 1977 році випущено Bourne shell.

У 1975 Міністерство оборони США утворило міжнародну групу для створення нової мови програмування для власних потреб, конкурс у 1979 виграла мова Ада.

У 1981 випущено dBASE II.

У 1984 році з метою об'єднання різних діалектів Ліспу створено Common Lisp. Було випущено MATLAB.

У 1985 році Б'ярн Страуструп опублікував реалізацію мови C++. Тоді ж випущено AWK.

У 1986 році опублікована мова Objective-C і створено Erlang. Тоді ж Borland і Apple незалежно створили об'єктно-орієнтоване розширення мови Pascal — Object Pascal.

У 1987 році створено Perl.

У 1990 році опубліковано Standard ML і Haskell.

У 1991 році створено Visual Basic і опубліковано Python.

У 1992 випущено Oracle 7 з підтримкою PL/SQL.

У 1993 році створено Lua.

У 1995 році Sun Microsystems випустила Java, Netscape – JavaScript, тоді ж створено PHP і Ruby.

У 1996 році створено OCaml.

У 2001 році створено C#.

У 2002 році створено F#. У 2003 році створено Scala.

3. C# (вимовляється «сі шарп») - об'єктно-орієнтована мова програмування. Вона розроблена у 1998-2001 роках групою інженерів під керівництвом Андерса Хейлсберг в компанії Microsoft як мова створення додатків для платформи Microsoft .NET Framework. C# розроблялася як мова програмування прикладного рівня для CLR і, як така, залежить, насамперед, від можливостей самої CLR. CLR (Common Language Runtime) – загальномовне виконуюче середовище, яке інтерпретує код мовою CIL (в який компілюються код програми C#) в байт-код. Ми будемо вивчати специфікацію C# 5.0, яка використовує платформу .NET Framework 4.5 в середовищі Visual Studio 2013 (Visual Studio Community).

За **рівнем** мови програмування поділяються на мови:

- *високого рівня*, в яких одній команді можуть відповідати сотні тисяч елементарних машинних команд. C# відноситься до мов програмування високого рівня;
- *низького рівня*, де одній команді, як правило, відповідає одна машинна команда, наприклад, мова Асемблера. Мовами низького рівня, як правило, створюються драйвери для апаратного забезпечення.

За **типом перетворення текстів** програм **оболонки та платформи мов програмування** поділяються на:

- *інтерпретатори* – відразу виконують команди програми в процесі перетворення. Переважна більшість оболонок мови програмування Basic є інтерпретаторами. Платформа .NET Framework теж відноситься до інтерпретаторів;
- *компілятори* – перетворюють тексти програм у бібліотеки підпрограм, виконувані EXE чи COM файли, після чого ці файли можуть виконуватися без самої оболонки. Саме оболонки з текстів програм створюють виконувані файли. Visual Studio є компілятором.

Розроблені програми поділяються на:

– *стандартні* – які створюються для розв’язання типових задач і орієнтуються на широке коло користувачів (операційні системи, прикладні пакети та їх частини);
– *зроблені під замовлення* – розробляються для конкретних підприємств та враховують специфіку їх діяльності;
– *гібридні* – створюються великими корпораціями для розв’язання задач притаманних багатьом організаціям з урахуванням специфіки їх реалізації (наприклад 1С).

4. Розрізняють 3 основних підходи до створення програми:
– лінійний;
– процедурний;
– об’єктно-орієнтований.

Перші два підходи реалізуються в алгоритмічних мовах програмування, а всі три – в об'єктно-орієнтованих.

При лінійному підході команди записуються та виконуються послідовно. Такі програми легко розуміються, але довго створюються.

У випадку процедурного підходу типові повторювані дії виділяються в окремі підпрограми і тому для виконання таких дій достатньо викликати створену підпрограму. Створюються такі програми швидше, але розуміються важче, хоча легше піддаються модифікаціям.

При об’єктно-орієнтованому підході програміст оперує з класами, які поєднують в собі дані та методи (процедури) їх обробки, а в процесі виконання програми може бути створено декілька об'єктів (екземплярів) кожного класу. Наприклад, в текстових редакторах описується клас документ, а користувач в процесі роботи може створити безліч екземплярів цього класу. Для цього підходу характерні 3 основні принципи:
– *наслідування* – для кожного класу на його основі можна створити новий породжений клас з доповненням його новими даними і методами;
– *інкапсуляція* – для кожного класу можна забезпечити потрібний рівень автономності від оточуючого середовища;
– *поліморфізм* – різні класи можуть мати методи з однаковою назвою, але для кожного об'єкта виконується метод саме з його класу.

Створення програм з використанням об'єктно-орієнтованого підходу складне і вимагає образного мислення, але такі програми легко модифікуються і займають порівняно мало місця в пам’яті.

5. Основні вимоги хорошого стилю програмування:
– логічно однакові послідовності команд необхідно виділяти в окремі процедури і функції;
– типові процедури і функції необхідно групувати у модулях за призначенням, кожен модуль і кожна підпрограма повинні мати зрозумілу назву;
– слід дотримуватися явного опису змінних і констант (тобто дані повинні бути явно описані перед першим своїм використанням (в C# ця вимога використовується автоматично). Змінні і константи бажано описувати на початку програми для уникнення створення зайвих змінних;
– рівнозначні оператори та команди мають набиратися з однаковим відступом зліва;
– вкладені оператори чи команди мають набиратися з додатковим пробілом зліва;
– бажано після кожної команди, яка впливає на хід виконання програми, давати коментарі, оскільки вони пізніше дадуть змогу зрозуміти логіку функціонування програми. Коментарі на розмір і хід виконання відкомпільованої програми не впливають.

В C# наявні функціональні можливості, що дозволяють створювати надійні та стійкі додатки. Серед них: функція **збору сміття** для автоматичного звільнення пам'яті, займаної об'єктами, які вже не використовуються; функція **обробки виключень**, що забезпечує структурований і розширюваний підхід до виявлення і усунення помилок; а також **строго типізована** структура мови, що не допускає зчитування неініціалізованих змінних, виходу

індексу масиву за межі допустимого діапазону або виконання неперевірених перетворень типів.

В C# застосовується **уніфікована система типів**. Всі типи C#, включаючи типи-примітиви (наприклад, int і double), наслідуються від єдиного кореневого типу object. Таким чином, всі типи використовують набір загальних операцій, що забезпечує узгоджені зберігання, передачу та обробку значень будь-якого типу.

Тема 2. Структура програми консольного додатку в C#. Опис даних засобами мови програмування C#

6. Алфавіт та словник мови програмування. Ідентифікатори.
7. Приклад найпростішого консольного додатку в C#.
8. Типи даних C#.
9. Опис констант в C#.
10. Опис змінних в C#.

6. Програми на мові C# формуються за допомогою набору знаків з його алфавіту. **Алфавіт** мови складається з букв, цифр та спеціальних символів. Неподільні послідовності символів утворюють слова. Слова розмежовуються між собою пробілами та розділовими знаками.

Частини тексту програми, які використовуються для пояснень і не компілюються середовищем називаються коментарями. В C# – це послідовності символів, які йдуть до кінця рядка після // або містяться між /* та */.

Слова та розділові знаки, які компілюють середовищем поділяються на *операнди, оператори, зарезервовані слова,* та *ідентифікатори*.

Ідентифікатори – це слова, що визначають дані, підпрограми, типи користувача чи оболонки мови програмування. Для коректної роботи програм в кожній області використання назва ідентифікатора повинна бути унікальною. Всі ідентифікатори поділяються на стандартні та створені користувачем. Користувачу не бажано давати ідентифікаторам назви, які співпадають з стандартними.

На назви ідентифікаторів накладаються наступні обмеження:
– назва ідентифікатора має починатися з букви;
– в назві можуть міститися букви, цифри і символ підкреслення;
– у назві не можуть використовуватися інші символи пунктуації та пробіли.

Ідентифікаторам слід давати змістовні назви, адже це прискорює розуміння закладених в програмах алгоритмів, а довжина ідентифікатора ніяк не впливає на розмір і швидкість виконання відповідних EXE-файлів. **Після кожного оператора, визначення чи опису в C# ставиться** `;`.

7. В C# використовуються такі основні структурні поняття: **програма**, **простір імен**, **тип**, **член** і **збірка**. Програма C# складається з одного або декількох вихідних файлів. В програмі оголошуються типи, які містять члени і можуть бути впорядковані у просторах імен. Прикладами типів є класи та інтерфейси. Прикладами членів є поля, методи, властивості і події. При компіляції програм C# виконується їх фізична упаковка в збірки. Файли збірок зазвичай мають розширення .exe або .dll і являють собою програми або бібліотеки відповідно. Збірки містять виконуваний код у формі інструкцій проміжної мови (Intermediate Language, IL), а також символьні дані у формі метаданих. Перед виконанням код IL збірки автоматично перетвориться в код для конкретного процесора за допомогою JIT-компілятора середовища .NET CLR.

Приклад найпростішого консольного додатку в C#:

```
using System;
class Hello
{static void Main()
  {Console.WriteLine("Hello, World");
   Consele.ReadKey();
  }
}
```

Програма починається з директиви `using`, яка посилається на простір імен `System`. Простори імен використовуються для ієрархічного впорядкування програм і бібліотек C#. Простори імен можуть містити типи і інші простори імен. Наприклад, простір імен `System` містить набір типів (наприклад, клас `Console`), а також ряд інших просторів імен (наприклад, `IO` і `Collections`). Директива `using` посилається на заданий простір імен і забезпечує можливість використання неповних імен типів, які є його членами. Завдяки

застосуванню директиви `using` в програмі можна використовувати скорочену форму запису `Console.WriteLine` замість `System.Console.WriteLine`.

8. Дані з якими оперує програма поділяються на *константи* та *змінні*. Кожна константа чи змінна повинна належати до певного типу даних. Саме тип даних визначає:

- структуру зберігання даних;
- діапазон можливих значень;
- допустимі операції над значеннями.

В C# всі типи поділяються на дві основні категорії: **типи значень** і **вказівникові типи**. Змінні типу значень безпосередньо містять дані, тоді як змінні вказівникового типу зберігають посилання на відповідні дані (об'єкти). Дві змінні вказівникового типу можуть посилатися на один об'єкт. Це дозволяє змінювати об'єкт, на який посилається одна змінна, виконуючи відповідні операції з іншого. Кожна змінна типу значень містить власну копію даних. У зв'язку з цим операції з однією змінною не впливають на іншу.

Типи значень в C# поділяються на **прості типи**, **перелічувальні типи**, **типи структур** і **обнульовувані типи**. Вказівникові типи в C# поділяються на **типи класів**, **типи інтерфейсів**, **типи масивів** і **типи делегатів**.

Базові типи значень C# наведені в табл. 1.

Таблиця 1

Базові типи значень C#

Категорія	Розрядність	Тип	Діапазон і точність
Типи класів			Початковий базовий клас для всіх типів: `object`
Знакові цілі	8	`sbyte`	–128...127
	16	`short`	–32,768...32,767
	32	`int`	–2,147,483,648...2,147,483,647
	64	`long`	–9,223,372,036,854,775,808...9,223,372,036,854,775,807
Цілі без знаку	8	`byte`	0...255
	16	`ushort`	0...65,535
	32	`uint`	0...4,294,967,295
	64	`ulong`	0...18,446,744,073,709,551,615
З плаваючою крапкою	32	`float`	Від $1{,}5 \times 10^{-45}$ до $3{,}4 \times 10^{38}$ з точністю до 7 знаків
	64	`double`	Від $5{,}0 \times 10^{-324}$ до $1{,}7 \times 10^{308}$ з точністю до 15 знаків
	128	`decimal`	Від $1{,}0 \times 10^{-28}$ до $7{,}9 \times 10^{28}$ з точністю до 28 знаків
Символи Unicod	16	`char`	
Рядки Unicod		`String`	
Логічні значення		`bool`	
Обнульовані типи		`<тип>?`	Крім значень базового типу може містити ще й null

Тип даних необхідно обирати так, щоб діапазон їх можливих значень вкладався в діапазон типу, а розмір був мінімальний. На швидкість виконання програми впливає саме тип даних, а не ім'я ідентифікатора. Ми найчастіше будемо використовувати типи **`int`**, **`double`**, **`char`**, **`bool`** та рядковий тип **`string`.** Тип **`string`** використовується для зберігання текстової інформації, тобто послідовностей символів. Рядки можуть мати максимальну довжину до 2 млрд. символів, якщо вона не обмежена явно. Для фіксування максимальної довжини рядка після ідентифікатора **`string`** її зазначають в квадратних дужках, наприклад, **`string[50]`**.

9. Константа – це ідентифікатор, що позначає визначену незмінну величину конкретного типу. Розділ констант починається з зарезервованого слова **`const`**, після якого перераховуються назви ідентифікаторів та значення їм присвоєні у форматі

```
<назва>=<значення>;
```

Наприклад:

```
const double E = 2.78;
```

10. Змінна – це ідентифікатор, що позначає визначену величину конкретного типу, яка може змінюватися. Для визначення змінних в програмі спочатку вказується їх тип даних, а потім через кому перераховуються ідентифікатори змінних. Наприклад: `int i, j, k;` .

Перед першим використанням змінна обов'язково має бути ініціалізована. Надалі в прикладах будемо наводити лише код процедури `main()`:

Для успішної реалізації алгоритму розв'язку задачі в обраній оболонці мови програмування потрібно:

1. Визначитися з переліком змінних та констант, тобто з'ясувати, який ідентифікатор що буде означати;
2. Забезпечити присвоєння чи введення вхідних даних задачі;
3. Записати та запрограмувати послідовність дій для обчислення вихідних результатів;
4. Вивести початкові дані (при потребі) та результати обчислень.

Тема 3. Операції та оператори

1. **Операції та їх класифікація.**
2. **Оператори. Прості оператори C#.**

11. Операції – це зарезервовані слова чи спеціальні символи мови програмування, що використовуються для виконання певних дій над даними, але не впливають на значення змінних та на хід виконання програми.

Базові операції C# наведені в табл. 2

Таблиця 2

Базові операції C#

Категорія	Вираз	Опис
Основні	`x.m`	**Доступ до члена**
	`x(..)`	**Виклик методів чи делегатів**
	`x[..]`	**Доступ до елементів масиву чи індексатора**
	`x++`	**Постфіксний інкремент**
	`x--`	**Постфіксний декремент**
	`new T(..)`	Створення об'єкта чи делегата
	`new T(..){..}`	Створення об'єкта з використанням ініціалізатора
	`new T[..]`	Створення масиву
	`checked(x)`	Упаковка виразу в контексті `checked`
	`unchecked(x)`	Розпаковка виразу в контексті `unchecked`
	`default(T)`	Отримання значення по замовчуванню для типу `T`
	`delegate {..}`	Анонімна функція (анонімний метод)
Унарні	`+x`	**Ідентифікація**
	`-x`	**Заперечення**
	`!x`	**Логічне заперечення**
	`~x`	**Побітове заперечення**
	`++x`	**Префіксний інкремент**
	`--x`	**Префіксний декремент**
	`(T)x`	**Явне перетворення `x` до типу `T`**
	`await x`	Асинхронне очікування завершення x
Мультиплі-кативні	`x * y`	**Множення**
	`x / y`	**Ділення**
	`x % y`	**Остача від ділення цілих чисел**
Адитивні	`x + y`	**Додавання, об'єднання рядків, об'єднання делегатів**
	`x - y`	**Віднімання, видалення делегата**
Зсуву	`x << y`	**Порозрядний зсув вліво**
	`x >> y`	**Порозрядний зсуввправо**
Відношення і перевірки типу	`x < y`	**Менше**
	`x > y`	**Більше**
	`x <= y`	**Менше або рівне**
	`x >= y`	**Більше або рівне**
	`x == y`	**Рівні**
	`x != y`	**Не рівні**
	`x is T`	Повертає `true`, якщо `x` відноситься до типу `T`, інакше - `false`
	`x as T`	Повертає `x`, що типізується як `T`, або значення `null`, якщо `x` відноситься до типу, відмінного від `T`
Логічні	`x & y`	**Ціле побітове `I`**
	`x ^ y`	**Ціле побітове виключне `АБО`, логічне виключне `АБО`**
	`x | y`	**Ціле побітове `АБО`**
	`x && y`	**Логічне `I`,** обчислює `y` тільки тоді, коли `x` рівне `true`
	`x || y`	**Логічне `АБО`,** обчислює `y` тільки тоді, коли `x` рівне `false`
	`x ?? y`	Обчислює `y`, якщо `x` має значення `null`; інакше обчислює `x`
Умовні	`x ? y : z`	**Обчислює `y`, якщо `x` рівне `true`, і `z`, якщо `x` рівне `false`**

Операції виконуються згідно пріоритетів:

– у **першу** чергу: `!`;
– у **другу** чергу: `*`, `/`, `%`, `&`;
– у **третю** чергу: `+`, `–`, `|`, `^`;
– у **четверту** чергу: `<`, `>`, `<=`, `>=`, `=`, `<>`.

Операції з однаковими пріоритетами виконуються зліва на право. Для зміни порядку виконання операцій ті з них, які потрібно виконати раніше, разом з операндами беруть у круглі дужки.

12. Оператори – це зарезервовані слова чи спеціальні символи мови програмування, що впливають на хід виконання програми або корегують значення змінних. Всі оператори умовно поділяються на прості і складені. Прості оператори, на відміну від складених, не містити в собі інших операторів.

Основні **прості оператори** C#:

– **оператор присвоєння** =, який обчислює значення виразу справа і заносить його у змінну зліва згідно синтаксису

```
<змінна> = <вираз>; .
```

В C#, як і в інших модифікаціях мови C підтримується також складене присвоєння у форматі `x <операція>= y`, що еквівалентно `x = x <операція>= y`. Складене присвоєння допустиме у форматах `*=`, `/=`, `%=`, `+=`, `-=`, `<<=`, `>>=`, `&=`, `^=`, `|=`.

Вирази у правій частині оператора присвоєння можуть містити змінні та константи, поєднані відповідними операціями:

```
x=32+7*y/10-(56+7*x);
prybutok=dohid*0.3; .
```

У простішому випадку змінній може присвоюватись значення іншої змінної чи константи:

```
zminna1=zminna2;
z=y;
str="Рядок";
i=5;     .
```

При використанні оператора присвоєння тип обчисленого справа результату повинен відповідати оголошеному типу змінної, якій присвоюється результат, інакше компілятор сприйме таке присвоєння як помилкове;

– **оператор безумовного переходу** `GOTO <мітка>;` . Мітки записуються у тексті програм в окремих рядках, де після їх ідентифікаторів вказується символ `:`.
– **порожній оператор** `;` не виконує ніяких дій, але може застосовуватися для позиціонування міток.

Для сприйняття декількох операторів як єдиного цілого в C# використовуються операторні дужки

```
{
 <оператори>;
};     .
```

Цей оператор поєднує в одну групу інші оператори. До складених операторів відносяться оператор розгалуження, оператор вибору, оператори циклу та ін., які ми розглянемо далі.

Тема 4. Організація вводу/виводу у вікні консольного додатку

Програма при введенні даних і виведення результатів взаємодіє з зовнішніми пристроями. Сукупність стандартних пристроїв вводу і виводу (екрану) називається консоллю.

У мові C# немає операторів введення та виведення. Замість них для обміну даними з зовнішніми пристроями використовуються спеціальні об'єкти. Зокрема, для роботи з консоллю використовується стандартний клас `Console`, визначений у просторі імен `System`. Розглянемо методи цього класу для виводу даних.

Метод `WriteLine()` виводить повідомлення у спеціальне консольне вікно на екрані дисплея і переводить курсор вікна на початок наступного рядка.

Метод `Write()` виводить текст, але залишає курсор відразу за останнім виведеним символом.

Для виконання обчислень виразів використаємо математичні функції класу `Math` з табл. 3

Таблиця 3

Основні математичні функції C#

Функція C#	Призначення
Math.Abs(X);	Модуль числа X
Math.Ceiling (X);	Заокруглення числа X до більшого цілого
Math.Floor(X);	Заокруглення числа X до меншого цілого
Math.Cos (X);	Косинус аргумента X
Math.E	Число e. e = 2,718282…
Math.Exp (X);	Експонента, число e в степені X
Math.Log(X);	Логарифм натуральний числа X
Math.Log10(X);	Логарифм десятковий числа X
Math.Max(X,Y);	Максимум з двох чисел X та Y.
Math.Min (X,Y);	Мінімум з двох чисел X та Y.
Math.Pi	Число пі.
Math.Pow(X,Y);	Число X в степені Y
Math.Round(X);	Математичне заокруглення числа X
Math.Sing(X);	Знак числа X
Math.Sin(X);	Синус аргумента X
Math.Sqrt(X);	Квадратний корінь числа X
Math.Tan(X);	Тангенс аргумента X

Приклад:

Обчислити значення виразу $z = \frac{x^2 + 7y}{cos(x;y)}$, де $x = e^{7y}$, $y = \pi ln\,78$. Вивести результат проміжних обчислень та значення виразу.

```
const int c = 78;
Double x, y, z;
x = Math.Exp(7*y);
y = Math.Pi*Math.Log(c);
z = (Math.Pow(x,y)+7*y)/Math.Cos(x*y);
Console.WriteLine("При c = {0,7:F2} обчислено x = {1,9}, y
= {2,8:F2} та z = {3:F3}", c, x, y, z);
Console.ReadKey();
```

Якщо в рядок виводу необхідно підставити значення змінних, то у `WriteLine` вони перераховуються після сталого тексту. Кожна з цих змінних має індекс, починаючи від 0 і в місцях вставки змінних у текст вказується цей індекс у фігурних дужках `{}`. Додатково після індекса через кому можна вказати довжину поля виводу і після двокрапки задати формат. В нашому випадку це формат дійсних чисел (`F`) з вказаною кількістю знаків після коми.

Різновиди форматів виводу у C# наведені у табл. 4.

Таблиця 4

Різновиди форматів виводу у C#

Символ	Призначення	Приклад використання
C	Виводить дані як суму грошей, застосовуючи прийняті для локалізації узгодження та позначення грошової одиниці. Число після символа, якщо є – це кількість знаків після коми	`Console.WriteLine("Грошовий формат - {0:C}, {1:C4}", 12.3, 123.45);`
D	Виводить ціле значення. Число після символа, якщо є – це кількість значущих цифр, але застосовується не менше потрібної кількості для виводу всього числа	`Console.WriteLine("Цілий формат - {0:D}, {1:D7}", 123, 12345);`
E	Виводить дійсне значення у форматі з плаваючою крапкою. Число після символа, якщо є – це кількість знаків після коми	`Console.WriteLine("Експоненційний формат - {0:E}, {1:E4}", 12.3, 123.45);`
F	Виводить дійсне значення у форматі з фіксованою крапкою. Число після символа, якщо є – це кількість знаків після коми	`Console.WriteLine("Формат з фіксованою крапкою - {0:F}, {1:F4}", 12.3, 123.45);`
G	Виводить дані у найпридатнішому для них форматі (як ціле або як дійсне з фіксованою крапкою)	`Console.WriteLine("Загальний формат - {0:G}, {1:G}", 123, 123.45);`
N	Виводить дані у форматі, зручному для сприйняття (ціла частина групами по три розряди, розділювач залежить від локалізації)	`Console.WriteLine("Числовий формат - {0:N}, {1:N4}", 1234567.8, 12345.67);`
X	Виводить ціле значення як шістнадцяткове. Число після символа, якщо є – це кількість виведених значущих цифр, але застосовується не менше потрібної кількості для виводу всього числа	`Console.WriteLine("Шістнадцятковий формат - {0:X}, {1:X4}", 123, 123);`

Більшість обчислень, які виконуються програмою, залежать від параметрів, введених користувачем.

Для введення значень від користувача в C# використовується `Console.ReadLine();`. Метод `ReadLine()` підтримує введення **тексту** з клавіатури комп'ютера і здійснює ехо-повтор введеного в консольному вікні, тобто зчитує рядок символів, введений з клавіатури (або іншого пристрою). Тобто ця функція повертає рядок введений користувачем. Результат введення повертається в програму після натиснення клавіш **Enter** чи **Ctrl+Z.** Якщо необхідно перетворити цей рядок до числа, то використовують функції класу `Convert`. Приклад, для перетворення рядка до дійсного числа:

```
Convert.ToDouble(Console.ReadLine());
```

Для переводу до цілого числа крім `Convert.ToInt32` можна також використати метод `Int32.Parse(<рядок>)`, але він орієнтований лише на десяткову систему числення, в той час, як для `Convert.ToInt32` систему числення числа можна задати другим аргументом (2, 8, 10 або 16).

Найчастіше перед введенням користувачу потрібно пояснити, що від нього очікує програма, тобто вивести повідомлення перед мигаючим курсором:

```
Console.Write(<повідомлення>);
```

(нагадаємо, що ця функція на відміну від `WriteLine` після виводу залишає курсор в тому самому рядку).

Приклад:

Обчислити значення виразу, увівши значення параметрів позначених знаком питання (?) від користувача: $y = \sqrt{|\ln x - \ln z| + 1.31}$, де $x = \frac{e^{-2.5a} + \sin^2(a^3)}{z|\ln|ea|}$, $z = \frac{\cos(e-a) + \sqrt[3]{ac}}{1 + \ln|ca|}$; a=?, c=?

```
Double a, c, z, x, y, KL, SY, SV;
Console.Write("Введіть a: ");
a = Convert.ToDouble(Console.ReadLine());
Console.Write("Введіть c: ");
c = Convert.ToDouble(Console.ReadLine());
SV = Math.Pow(a*c,1/3);
z =(Math.Cos(c-a)+SV)/(1+Math.Log(Math.Abs(c*a)));
KL = Math.Sin(Math.Pow(a,3));
x = Math.Exp(-2.5*a)+Math.Pow(KL,2)/
    (2*Math.Log(Math.Abs(c*a)));
SY = Math.Abs(Math.Log(Math.Abs(x))-Math.Log(Math.Abs(z)));
y = Math.Pow(a*c,1/3);
Console.WriteLine("При a = {0}, c = {1} обчислено z =
{2,7}, x = {3}, y = {4}", a, c, z, x, y);
Console.ReadKey();
```

У вікні консольного додатку діють національні стандарти, тому числа вводяться з комою (,), якщо введений рядок не вдалося перетворити в число, то виникає помилка часу виконання і при використанні оболонки та продовженні виконання можна значення змінної ввести ще раз. Якщо ж використовується сам EXE-файл, то він завершує виконання.

Приклад:

Обчислити площу кільця, якщо відомі його внутрішній і зовнішній радіуси.

```
Double R1, R2, S;
Console.Write("Введіть внутрішній радіус кільця: ");
R1 = Convert.ToDouble(Console.ReadLine());
Console.Write("Введіть зовнішній радіус кільця: ");
R2 = Convert.ToDouble(Console.ReadLine());
S = Math.Pi*(Math.Pow(R2,2)-Math.Pow(R1,2));
Console.WriteLine("При зовнішньому радіусі {0} і
внутрішньому радіусі {1} площа кільця становить {2}", R2,
R1, S);
Console.ReadKey();
```

Тема 5. Організація вводу/виводу в діалогових вікнах

1. **Підключення бібліотек для вводу/виводу в діалогових вікнах.**
2. **Організація вводу в діалогових вікнах.**
3. **Організація виводу в діалогових вікнах.**

13. В консольному додатку по замовчуванню ввід/вивід виконується в `Console` за допомогою методів `WriteLine, ReadLine`. Щоб вводити/виводити в діалогових вікнах, спочатку необхідно підключити бібліотеки (.dll), де містяться відповідні методи, а потім вже з них відкривати простори імен і використовувати класи.

Для організації виводу ми використовуватимемо бібліотеку `System.Windows.Forms`, а для організації вводу – `Microsoft.VisualBasic`.

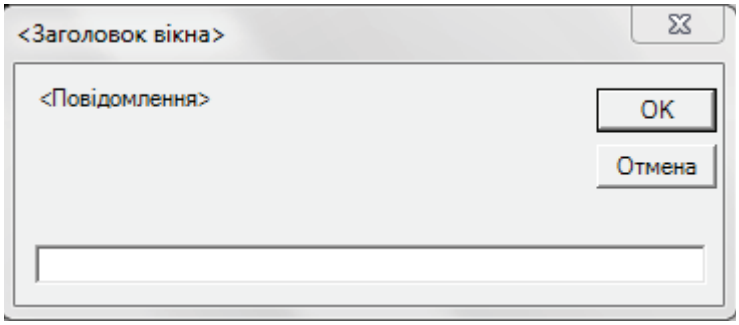
Для підключення бібліотек необхідно у вікні рішення віднайти групу посилань `References` і в її контекстному меню обрати пункт `Добавить ссилку`, віднайти посилання на потрібні бібліотеки та встановити біля них прапорці і натиснути `OK`.

14. Для організації вводу в діалогових вікнах ми використовуватимемо метод `Interaction.InputBox`, який описаний у просторі імен `Microsoft.VisualBasic`.

Синтаксис методу:

```
string Interaction.InputBox(string <повідомлення>, [string
   <заголовок вікна>], [string <значення на замовчування>])
```

Цей метод виводить на екран діалогове вікно з вказаним повідомленням і заголовком, рядком вводу і кнопками OK та Cancel (Отмена).



Обов'язковим параметром методу є лише перший, але хорошим стилем програмування передбачається задання значень по замовчуванню.

Після натиснення `OK` метод повертає в програму введений рядок, якщо ж натиснути `Cancel`, або закрити вікно, то в програму повернеться порожній рядок. Надалі користувач сам має перетворити введений рядок до потрібного типу даних.

Наприклад, введемо радіус зовнішнього кола в попередній програмі:

```
R1 = 5;
S  =  Interaction.InputBox("Введіть радіус зовнішнього кола",
                           "Введення", R1.ToString());
R1 = Convert.ToDouble(S);
```

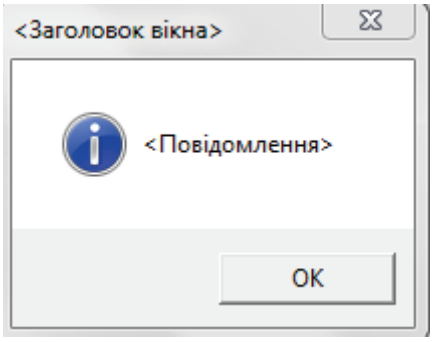
15. Для організації виводу скористаємося методом `Show` класу `MessageBox`. Для цього методу будемо використовувати максимум 4 аргументи.

Синтаксис методу:

```
DialogResult MessageBox.Show(string <повідомлення>, string
   <заголовок вікна>, MessageBoxButtons.<набір кнопок>,
   MessageBoxIcon.<константа значка>)     ,
```

де в перелічуванні `MessageBoxButtons` можливий вибір однієї з констант: `AbortRetryIgnore, OK, OKCancel, RetryCancel, YesNo, YesNoCancel`, а в перелічуванні `MessageBoxIcon` – однієї з констант: `Asterisk, Information` (інформаційні повідомлення), `Stop, Error, Hand` (критичні повідомлення), `Warning,`

Exclamation (знак оклику), None, Question. Результатом виконання функції є константа натиснутої користувачем кнопки з перелічування DialogResult.



Методи InputBox та Show можуть використовуватися з різною кількістю аргументів. Всі ці варіанти реалізовані в бібліотеках. Методи з однаковою назвою, але різною кількістю аргументів називаються **перезавантажувальними**.

Приклад:

Обчисліть периметр і площу прямокутного трикутника, якщо відомі гіпотенуза і один з його катетів.

Для організації вводу/виводу після підключення бібліотек в модулі програми після стандартних відкриваємо додаткові простори імен:

```
using System.Windows.Forms;
using Microsoft.VisualBasic;
```

Текст програми в методі Main:

```
Double a, b, c, P, S;
String s;
c = 5;
a = 3;
s = Interaction.InputBox("Введіть гіпотенузу", "Введення",
                         c.ToString());
c = Convert.ToDouble(s);
s = Interaction.InputBox("Введіть довжину катета",
                        "Введення", a.ToString());
a = Convert.ToDouble(s);
b = Math.Sqrt(c*c-a*a);
P = a+b+c;
S = (a+b)/2;
MessageBox.Show("Якщо гіпотенуза рівна " + c.ToString() +
", а один з катетів " + a.ToString() +
", то другий катет становитиме " + b.ToString() +
", периметр трикутника " + P.ToString() +
", а площа " + S.ToString());
```

При вводі можна сформувати заголовок вікна, кнопки та значки. Для цього після повідомлення в дужках потрібно записати:

```
, "Результати обчислень", MessageBoxButtons.OK,
MessageBoxIcon.Information);
```

Повідомлення, яке виводиться в діалоговому вікні, можна розбивати на декілька фізичних рядків. Для цього в текст додається спецсимвол переводу рядка чи повернення каретки Strings.Chr(13) або Strings.Chr(10).

Тема 6. Програмування з використанням операторів розгалуження

3. **Різновиди та синтаксис операторів розгалуження.**
4. **Використання скороченого та повного операторів розгалуження для обчислення значення виразів.**
5. **Розв'язок прикладних задач з використанням операторів розгалуження.**
6. **Забезпечення коректного вводу даних за допомогою операторів розгалуження.**

16. Оператори розгалуження дають змогу виконати вкладені оператори при забезпеченні виконання чи не виконання вказаної умови.

Використовують два синтаксиси операторів розгалуження:

a) Скорочений `if(умова)<оператор>;`

Оператор виконується лише тоді, коли умова виявляється істинною.

b) Повний `if(умова)<оператор1>else<оператор2>;`

Якщо умова істинна, то виконується оператор1, інакше виконується оператор2.

У випадку, коли при виконанні чи невиконанні умови необхідно виконати декілька операторів, тоді використовуються операторні дужки {}. Ці операторні дужки дозволяють інтерпретувати всі вкладені оператори як єдиний оператор.

Згідно хорошого стилю програмування всі вкладені оператори мають набиратися з додатковим відступом зліва і операторні дужки мають набиратися на однаковому рівні.

17. Приклад: Обчислити значення виразу: $y = \begin{cases} \sin^2 x, 0 \le x < 1 \\ \sqrt{[\square|\ln]\, x - \sin[x|\square]}, 1 \le x \le 3 \\ x\ln^2 x, \quad x > 3, x < 0 \end{cases}$, використовуючи ввід/вивід в консольному додатку.

Умови для операторів розгалуження можуть складатися з декількох логічних частин, які поєднуються між собою операцією «і» AND (&) або операцією «або» OR (|).

```
Double x, y;
        Console.Write("Введіть x = ");
        x = Convert.ToDouble(Console.ReadLine());
        if (x >= 0 & x < 1)
          y = Math.Pow(Math.Sin(x),2);
        if (x >= 1 & x <= 3)
          y = Math.Sqrt(Math.Abs(Math.Log10(x)-Math.Sin(x)));
        if (x > 3 | x < 0)
          y = x*Math.Pow(Math.Log10(x),3);
        Console.WriteLine("При x = {0} обчислено y = {1}", x, y);
        Console.ReadKey();
```

Обчислимо значення цього самого виразу використаємо повний оператор розгалуження і ввід/вивід у діалогових вікнах.

```
        Double x, y;
        String S;
        x = 93;
        S = Interaction.InputBox("Введіть x ", "Введення",
        x.ToString());
        x = Convert.ToDouble(S);
        if (x >= 0 & x < 1)
          y = Math.Pow(Math.Sin(x),2);
        else
          if (x >= 1 & x <= 3)
            y=Math.Sqrt(Math.Abs(Math.Log10(x)-Math.Sin(x)));
          else
            y = x*Math.Pow(Math.Log10(x),3);
```

Для першого варіанту з використанням скороченого оператора розгалуження завжди виконується три таких оператори. Порахуємо складність повного оператора: якщо x належить першому проміжку, то виконається один оператор розгалуження, якщо ж x

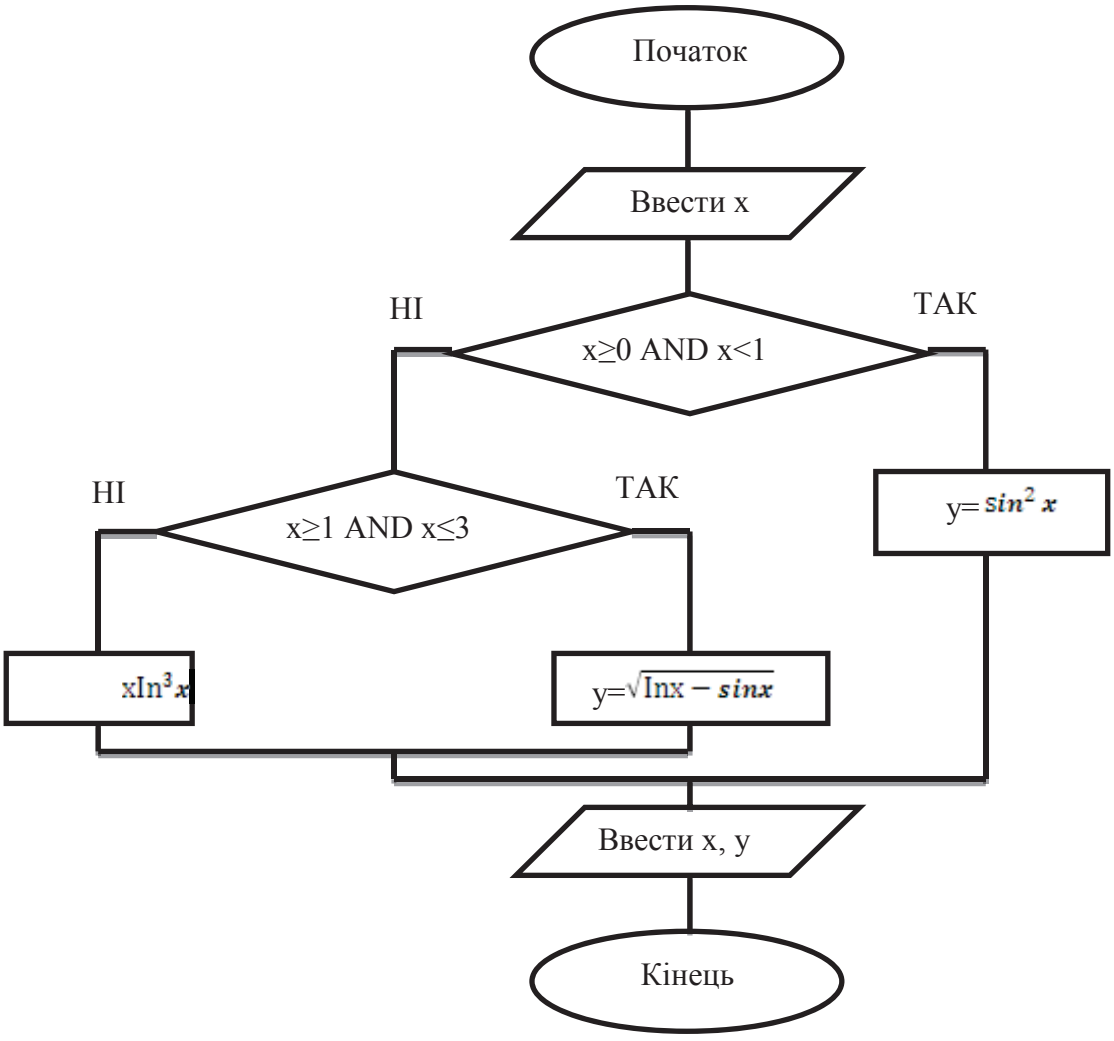
належить другому проміжку, то буде виконуватися два оператори розгалуження, а якщо x належить третьому проміжку, то також виконуватимуться два оператори. Середня складність такого виконання становить: $\frac{1+2+2}{3} = \frac{5}{3} \approx 1.67$.

Для додаткового прискорення третій найширший проміжок можна перевіряти першим і спростити умову другого оператора розгалуження:

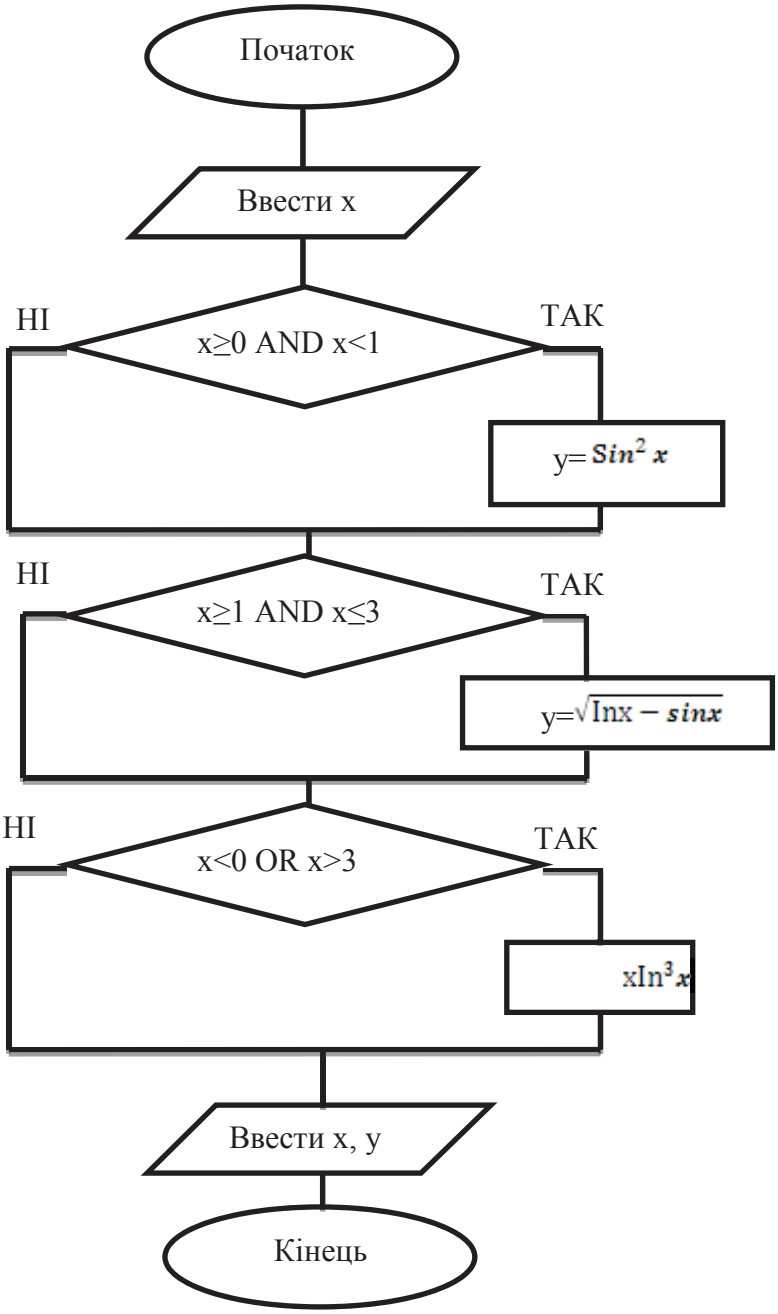
```
if (x > 3 | x < 0)
  y = x*Math.Pow(Math.Log10(x),3);
else
  if (x < 1)
    y = Math.Pow(Math.Sin(x),2);
  else
    y=Math.Sqrt(Math.Abs(Math.Log10(x)-Math.Sin(x)));
MessageBox.Show("При x = " + x.ToString() + " обчислено y =
"     +     y.ToString(),     "Результати     обчислення",
MessageBoxButtons.OK, MessageBoxIcon.Information);
```

Для розуміння суті виконання операторів програм їх зручно подавати у вигляді схем алгоритмічної мови. Для цього в овалі вказується термінатор (початок чи кінець алгоритму з вказуванням цих слів), в паралелограмі - операції вводу/виводу, в прямокутнику -0 лінійна структура, послідовність виконання програм, ромб означає розгалуження, вправо – ТАК, вліво – НІ, вибір рішення підписується.

Створимо блок-схему для повного варіанту розгалуження попередньої програми:



Для скороченого варіанту розгалуження:



18. На практиці часто доводиться обчислювати одне значення виразу з різними початковими даними. В цьому випадку після виводу результату програма має запитати користувача чи рахувати ще і при позитивній відповіді знову повертатися на початок програми. Повернення має відбуватися саме у місце вводу і по замовчуванню має виводитися попереднє введене значення . Для організації переходу в програмі використовується оператор безумовного переходу `goto<мітка>;`. Для опису мітки програми, вона вставляється в новому рядку і після самої мітки ставиться двокрапка «:». Рекомендується міткам давати змістовні назви і записувати їх спочатку рядка.

Приклад:

Обчислити вираз з використанням повного оператора розгалуження та можливістю повторного введення даних: $\begin{cases} \cos^3 x^2 - \sin x, x \le 0 \\ x^2 - 0.82, 0 < x < 3 \\ \dfrac{1.4 + x}{\ln x}, x \ge 3 \end{cases}$.

```
   Double x, y;
   String S;
   x = 5;
Povtor:
   S = Interaction.InputBox("Введіть x ", "Введення",
   x.ToString());
   x = Convert.ToDouble(S);
   if (x <= 0)
     y = Math.Pow(Math.Cos(x*x),3)-Math.Sin(x);
   else
     if (x > 0 & x < 3)
       y=x*x-0.83;
     else
       y = (1.4+x)/Math.Log10(x);
   if (MessageBox.Show ("При x = " + x.ToString() + "
   обчислено y = " + y.ToString() + Strings.Chr(13) +
   Strings.Chr(13) + "Бажаєте рахувати ще?", "Результати
   обчислень",                          MessageBoxButtons.YesNo,
   MessageBoxIcon.Information)=DialogResult.OK)
     goto Povtor;
```

або:

```
   DialogResult R = MessageBox.Show("При x = " + x.ToString()
   + " обчислено y = " + y.ToString() + Strings.Chr(13) +
   Strings.Chr(13) + "Бажаєте рахувати ще?", "Результати
   обчислень",                          MessageBoxButtons.YesNo,
   MessageBoxIcon.Information);
      if (R == DialogResult.Yes)
          goto Povtor;
```

Приклад:

Відомо значення загальних витрат і доходів підприємства. Визначити чи воно прибуткове, чи збиткове.

Звичайно, якщо доходи перевищують витрати, то можна виводити повідомлення, що підприємство прибуткове і питати користувача, чи бажає рахувати ще, аналогічно можна аналізувати збитковість і ні прибутковість, ні збитковість підприємства, але тоді вивід повідомлення буде запрограмований тричі з трьома безумовними переходами. Для забезпечення і виводу, і одного оператора безумовного переходу результат обчислень занесемо в змінну S і виведемо її лише один раз:

```
   Double Prub, Vutr;
   String S;
   Prub = 200 ;
   Vutr = 150;
Povtor:
   S=    Interaction.InputBox("Введіть    розмір    прибутку
   підприємства", "Введення", Prub.ToString());
   Prub = Convert.ToDouble(S);
   S    =    Interaction.InputBox("Введіть    розмір    витрат
   підприємства", "Введення", Vutr.ToString());
   Vutr = Convert.ToDouble(S);
   if (Prub >Vutr)
     S="підприємство прибуткове";
   if(Prub <Vutr)
     S="підприємство збиткове";
   else
     S="ні прибуткове, ні збиткове";
```

```
DialogResult R = MessageBox.Show("При прибутку " +
Prub.ToString() + " та витратах " + Vutr.ToString() + S +
Strings.Chr(13) + Strings.Chr(13) + "Бажаєте повторити?",
"Результати обчислень", MessageBoxButtons.YesNo,
MessageBoxIcon.Information);
   if (R == DialogResult.Yes)
       goto Povtor;
```

19. При організації вводу з діалогового вікна ми застосовували функцію `InputBox`, але ця функція вводиться рядок і при некоректному введенні в місті переводу рядка в число може виникнути помилка часу виконання, яка призведе до аварійного завершення програми. В таких випадках програміст сам організовує обробку виключних операцій. Він відключає стандартну обробку в небезпечних місцях за допомогою оператора `try` і обробляє можливі виключні ситуації, використовуючи обробник `catch`. Якщо в процесі виконання вводу помилок не виявлено, то виконання продовжується після `catch`, якщо ж з'являється помилка часу виконання, то Visual Studio виконує пошук виявленої помилки серед обробників `catch` і застосовує перший з тією ж помилкою. Після `catch` може не вказуватися жодна помилка, тоді він обробляє всі можливі помилки.

Приклад:

Два числа задані формулами: $x = \frac{m - 3n}{7}$, $y = \frac{2m + n}{3 - n^2}$. Перерозподілити значення x та y так, щоб в x виявилося менше значення, а в y – більше.

```
PROM = x;
x=y;
y=PROM;
```

Виконувати перестановку x та y за допомогою двох дій `x=y`, `y=x` неможливо, тому що після першого присвоєння знищується значення `x`. Для його запам'ятовування необхідно створити проміжну змінну `PROM` і виконати перестановку в три присвоєння:

```
Double m, n, x, y, PROM;
String S;
m = 12;
n = 37;
Povtor:
   S = m.ToString();
   Povtor1:
   S = Interaction.InputBox("Введіть m ", "Введення", S);
   try
   {
    m = Convert.ToDouble(S);
   }
   catch(System.FormatException)
   {
     if (MessageBox.Show("Ви ввели не число" + Strings.Chr(13)
   +  Strings.Chr(13)  +  "Бажаєте  повторити?",  "Увага",
   MessageBoxButtons.YesNo,    MessageBoxIcon.Warning)    ==
   DialogResult.Yes)
       goto Povtor1;
     else
       return;
   }
   S = n.ToString();
Povtor2:
   S = Interaction.InputBox("Введіть n ", "Введення", S);
   try
```

```
{
 n = Convert.ToDouble(S);
}
catch (System.FormatException)
{
 if (MessageBox.Show("Ви ввели не число" + Strings.Chr(13)
+   Strings.Chr(13)   +   "Бажаєте   повторити?",   "Увага",
MessageBoxButtons.YesNo,     MessageBoxIcon.Warning)     ==
DialogResult.Yes)
  goto Povtor2;
 else
   return;
}
x = (m-3*n)/7;
y = (2*m+n)/(3-n*n);
if (x>y)
{
 PROM = x;
 x = y;
 y = PROM;
}
if (MessageBox.Show(" При m = " + m.ToString() + "  та n =
" + n.ToString() +
                "  обчислено  x = " + x.ToString()+ "  y =
" + y.ToString() + Strings.Chr(13) + Strings.Chr(13)  + "
Бажаєте      повторити?",      "Результати      обчислень",
MessageBoxButtons.YesNo,   MessageBoxIcon.Information)   ==
DialogResult.Yes)
  goto Povtor;
```

Тема 7. Застосування методів у програмуванні

1. **Призначення та синтаксис опису методів.**
2. **Локальні змінні та константи.**
3. **Формальні та фактичні параметри методів.**

1. Методи використання для виконання повторюваних операцій та для опису дій об'єктів. Типовий приклад методів `WriteLine`, який виводить дані користувача у вікно консольного додатку. Кожен метод має своє унікальне ім'я в сукупності з аргументами. Виклик методу виконується вказуванням його назви, після чого зазначаються круглі дужки, де перераховуються аргументи.

В C# використовуються статистичні методи та методи класу. Статистичні методи розміщуються в пам'яті один раз як і їхні змінні, перед назвою статистичного методу ставиться слово `static`. Методи класу створюються вказаного класу і їх змінні створюються для кожного екземпляра.

Синтексис опису статистичних методів:

```
static<тип результату><назва методу>(<список аргументів>);
```

Для виходу з методу використовується оператор `return`. Якщо метод не повертає жодне значення в місце виклику, то замість типу його результату вказується слово `void`. Тип результату після `return` обов'язково має співпадати з типом результату підпрограми.

2. Всередині кожної підпрограми можуть використовуватися свої змінні і константи, які називають локальними, вони створюються під час виклику методу і зникають після завершення його роботи.кожній локальній змінній може відразу присвоюватися початкове значення. Наприклад: `int i = 0;` . C# забороняє використовувати змінні, доки їм не присвоєно значення.

3. В кожен метод можуть передаватися аргументи, які між собою відмежовані комами. Ім'я аргумента у списку аргументів і при виклику методу можуть не співпадати, але обов'язково має співпадати тип даних. Передача аргумента можлива по значенню і по змінній. Крім цього в C# ще й використовують аргументи результату.

При передачі по значенню створюються формальні параметри фактично формується нова змінна, в яку з основної програми заноситься початкове значення і надалі змінна програми і змінна підпрограми між собою незалежні.

При передачі за адресою дії відбуваються над змінними основної програми. Перед такою змінною в списку аргументів вказується службове слово ref, при цьому використовується фактичний параметр.

Для формування параметра виводу в списку аргументів і при виклику перед ним вказується слово out. В такий аргумент не заноситься початкове значення, а лише повертається результат.

Приклад:

Скласти програму для розв'язку квадратного рівняння: $ax^2 + bx + c = 0$.

Для розв'язку квадратного рівняння необхідно вводити коефіцієнти a, b, c і формувати рядок з результатом. Якщо не використовувати підпрограму, то фрагмент для коректного введення необхідно буде повторити тричі і використовувати три різні мітки. Такі повтори необхідно буде використовувати і в інших програмах для введення дійсних чисел. Тому створимо власну підпрограму для введення дійсного числа і будемо її викликати потрібну кількість разів. Підпрограма може ввести і не ввести дані від користувача, тому буде повертати логічний результат.

```
static bool InputDouble(ref Double x, String Povidom)
{
  String S;
```

```
  S = x.ToString();
Povtor:
  S = Interaction.InputBox(Povidom, "Введення", S);
  try
  {
   x = Convert.ToDouble(S);
  }
  catch (System.FormatException)
  {
   if    (MessageBox.Show("Ви    ввели    не    число"    +
Strings.Chr(13) + Strings.Chr(13) + "Бажаєте повторити?",
"Увага",  MessageBoxButtons.YesNo,  MessageBoxIcon.Warning)
== DialogResult.Yes)
     goto Povtor;
   else
     return false;
  }
     return true;
}
```

Блок-схема:

```
   Double a, b, c, x1, x2, D;
   String REZ;
   a = 25;
   b = 7;
   c = -3;
Povtor:
   if (!InputDouble(ref a, "Введіть a "))
     return;
   if (!InputDouble(ref b, "Введіть b "))
     return;
   if (!InputDouble(ref c, "Введіть c "))
   return;
   if (a != 0)
   {
     D = b * b - 4 * a * c;
     if (D > 0)
     {
       x1 = ((-b - Math.Sqrt(D)) / (2 * a));
       x2 = ((-b + Math.Sqrt(D)) / (2 * a));
       REZ = "рівняння має 2 корені x1 = " + x1.ToString() +
   ", x2 = " + x2.ToString();
      }
      else
        if (D < 0)
          REZ = "рівняння розв'язків не має";
        else
        {
          x1 = -b / (2 * a);
          REZ = "рівняння має кратний корінь x = " +
   x1.ToString();
        }
      }
     else
     {
       if (b == 0)
       {
         if (c == 0)
           REZ = "рівняння має безліч розв'язків";
         else
           REZ = "рівняння розв'язків не має";
       }
```

```
        else
        {
          x1 = -c / b;
          REZ = "рівняння має один корінь x = " +
    x1.ToString();
         }
    }
    if (MessageBox.Show(" При a = " + a.ToString() + ", b = " +
    b.ToString() + ", c = " + c.ToString() + " " + REZ +
    Strings.Chr(13) + Strings.Chr(13) + " Бажаєте рахувати
    ще?", "Результати обчислень", MessageBoxButtons.YesNo,
    MessageBoxIcon.Information) == DialogResult.Yes)
      goto Povtor;
}
```

В окремих випадках замість повного та неповного оператора розгалуження застосовують скорочений оператор розгалуження, наприклад, коли необхідно обчислити значення однієї змінної. Його синтаксис:

```
<змінна>=<логічний вираз>?<значення1>:<значення2>;
```

Застосування скороченої операції розгалуження замість повного оператора розгалуження зменшує розмір тексту і виконуваного файлу, хоча й не впливає на швидкість виконання. Наприклад, при розв’язку квадратного рівняння, коли a=b=0 ми записували:

```
if (a=0 & b=0)
if (c=0)
  REZ = "рівняння має безліч розв'язків";
else
  REZ = "рівняння розв'язків не має";
```

Якщо ж використати скорочену операцію, то цей фрагмент можна переписати так:

```
if (a=0 & b=0)
 REZ=c=0? "рівняння має безліч розв'язків";
          "рівняння розв'язків не має";
```

Тема 8. Використання оператору вибору в програмуванні

Якщо значення змінної чи виразу, яке є дискретним, необхідно перевіряти не рівність декільком константам чи значенням, то замість послідовних операторів розгалуження чи вкладених повних операторів розгалуження, краще застосовувати оператор вибору `switch`, синтаксис якого має вигляд:

```
switch <вираз-селектор>
{case <значення1>:
          <оператори1>;
          break;
  case <значення2>:
          <оператори2>;
          break;
  …
  default:
           <оператори default>
           break;
 }
```

При виконанні оператора вибору спочатку обчислюється значення виразу-селектора, після цього обчислене значення порівнюється зі значенням1. Якщо вони рівні, то виконується оператор1 і виконання оператора вибору завершується. Інакше виконується порівняння зі значенням2 і так далі. Якщо вираз-селектор не рівний жодному зі значень `case`, то при наявності виконуються оператори `default`.

Приклад:

Вивести пору року залежно від номеру введеного місяця.

```
int i=4;
String S;
if (!InputInt(ref i, "Введіть номер місяця"))
  return;
if ((i>=3)&(i<=5))
   S= "Весна"
else
   if ((i>=6)&(i<=8))
     S= "Літо"
  else
     if ((i>=9)&(i<=11))
        S= "Осінь"
    else
        if ((i=12)|(i=1)|(i=2))
           S= "Зима"
       else
          S="Пора року для такого місяця не визначена"
```

Альтернатива цієї самої програми з використанням оператора вибору:

```
if (!InputInt(ref i, "Введіть номер місяця"))
  return;
switch(i)
{ case 3:
  case 4:
  case 5:
    S = "Весна";
    break;
  case 6:
  case 7:
  case 8:
```

```
      S = "Літо";
      break;
    case 9:
    case 10:
    case 11:
      S = "Осінь";
      break;
    case 12:
    case 1:
    case 2:
      S = "Зима";
      break;
        default:
      S = "Пора року не визначена";
      break;
  }
```

В C# вимагається, щоб після операторів завжди йшло `break` (на відміну від C++, де можуть продовжуватися виконання операторів і інших варіантів). Тому після операторів кожного варіанту обов'язково ставиться `break`.

Приклад:

Створити текстову програму "Знання історії групи ЕК-11". Передбачити не менше 5 питань і використання питань з-ох форматів: так/ні, введення відповіді з клавіатури, вибору одного варіанту із запропонованих. За результат тестування вивести відповідні рекомендації.

В програмі будемо послідовно задавати питання і відразу визначати правильність відповідей.

```
Povtor:
   int CountPut = 0, CountBal = 0, CountVirno = 0;
   String S;
   if (MessageBox.Show("Чи з кожної пари відпрошується
   Балаушко?", "Питання №1", MessageBoxButtons.YesNo,
   MessageBoxIcon.Question)==DialogResult.Yes)
   {
     CountBal = CountBal + 2;
     CountVirno++;
   }
   else
   MessageBox.Show("Ви помилилися! Балаушко відпрошується з
   кожної пари", "Увага!",
   MessageBoxButtons.OK,MessageBoxIcon.Error);
   CountPut++;
   if (Interaction.InputBox("Яка абревіатура факультету, де
   навчається група ЕК-11?", "Питання №2", "Факультет
   інформатики")=="ДКМ")
   {
     CountBal = CountBal+2;
     CountVirno++;
   }
   else
   MessageBox.Show("Ви відповіли не вірно! ЕК-11 навчається на
   факультеті ДКМ", "Увага!",
   MessageBoxButtons.OK,MessageBoxIcon.Error);
      CountPut++;
   if (Interaction.InputBox("Хто староста групи ЕК-11?" +
   Strings.Chr(10) +
```

```
    "1. Гординський" + Strings.Chr(10)+
    "2. Пашко" + Strings.Chr(10)+
    "3. Балаушко" + Strings.Chr(10)+
Strings.Chr(13) + "Введіть номер правильної відповіді",
"Питання №3", "3") == "2")
{
  CountBal = CountBal+2;
  CountVirno++;
}
else
MessageBox.Show("Ви відповіли не вірно! Старостою групи ЕК-
11 є Пашко", "Увага!",
MessageBoxButtons.OK,MessageBoxIcon.Error);
  CountPut++;
switch (CountVirno)
{   case 6:
      S = "Відмінно";
      break;
    case 5:
      S = "Дуже добре";
      break;
    case 4:
      S = "добре";
      break;
    case 3:
      S = "задовільно";
      break;
    case 2:
      S = "не задовільно";
      break;
    case 1:
      S = "погано";
      break;
    default:
      S = "дуже погано! Жодна відповідь не правильна";
      break;
}
if (MessageBox.Show("За результатами тестування дано " +
CountVirno.ToString()+" правильних відповідей з " +
CountPut.ToString() + ". Кількість набраних балів - " +
CountBal.ToString() + Strings.Chr(13) + "Ваш результат: " +
S + Strings.Chr(13) + Strings.Chr(13) + "Бажаєте
тестуватися ще?", "Результати тестування",
MessageBoxButtons.YesNo,
MessageBoxIcon.Information)==DialogResult.Yes)
   goto Povtor;
```

Підготовка до МКР1:

1. **Впорядкувати 3 числа a, b, c за зростанням.**
   Забезпечуємо запис в змінну a найменшого значення:

```
if (b<a)
 {prom=a;
   a=b;
   b=prom;
 }
if (c<a)
 {prom=a;
   a=c;
   c=prom;
 }
if (c<b)
 {prom=c;
   c=b;
   b=prom;
 }
```

2. **Визначити максимальне і середнє серед 3-ох чисел.**

```
Max=a;
if (b>Max)
Max=b;
if (c>Max)
Max=c;
```

Для знаходження середнього впорядкуємо числа за зростанням як в попередній програмі і допишемо `Serd=b;`

3. **Визначити кількість мінімальних серед 3-ох чисел.**

```
Min=a;
if (b<Min)
Min=b;
if (c<Min)
Min=c;
Count=0;
if(a==Min)
Count++;
if(b==Min)
Count++;
if(c==Min)
Count++;
```

4. **Знайти кількість різних серед 3-ох чисел.**

```
Count=1; //Вважаємо, що різним є число a:
 if(b!=a)
Count++;
 if(c!=a & c!=b)
Count++;
```

5. **Числа m та n задаються користувачем. x = (m - 3 * n) / 7, y = (2 * m + n) / (3 - n * n), z = x * y. Серед чисел x, y, z визначити ті, що лежать поза проміжком [-5;7] та підрахувати їх кількість.**

а) Розв'язування задачі, використовуючи ввід/вивід з консолі:

```
static void Main(string[] args)
{Double m, n, x, y, z;
 int count;
 String rez;
 //введення початкових значень
 Console.Write("Введіть m: ");
 m = Convert.ToDouble(Console.ReadLine());
 Console.Write("Введіть n: ");
 n = Convert.ToDouble(Console.ReadLine());
 //розв'язування задачі
 x = (m - 3 * n) / 7;
 y = (2 * m + n) / (3 - n * n);
 z = x * y;
 rez = "";
 count = 0;
 if (x < -5 || x > 7)
 {   rez = x.ToString();
     count++;
 }
 if (y < -5 || y > 7)
 {   if (rez != "")
         rez += ", ";
     rez += y.ToString();
     count++;
 }
 if (z < -5 || z > 7)
 {   if (rez != "")
         rez += ", ";
     rez += z.ToString();
     count++;
 }
 rez = rez == "" ? " відсутні" : ": " + rez;
 //вивід результатів
 Console.WriteLine("Обчислено: x={0}, y={1}, z={2}.\nСеред них числа поза проміжком [-5;7]{3}"+
                   ".\nЇх кількість: {4}.", x, y, x, rez, count);
 Console.ReadKey();
}
```

а) Розв'язування задачі, використовуючи ввід/вивід з діалогових вікон та контроль на введення чисел:

```
static void Main(string[] args)
{   Double m, n, x, y, z;
    int count;
    String S, rez;
    m = 12;
    n = 37;
Povtor:
    //введення початкових значень
    S = m.ToString();
Povtor1:
    S = Interaction.InputBox("Введіть m ", "Введення", S);
    try
    {
        m = Convert.ToDouble(S);
    }
    catch (System.FormatException)
    {
        if (MessageBox.Show("Ви ввели не число" + Strings.Chr(13) + Strings.Chr(13) + "Бажаєте повторити?",
                            "Увага", MessageBoxButtons.YesNo, MessageBoxIcon.Warning) == DialogResult.Yes)
            goto Povtor1;
        else
            return;
    }
    S = n.ToString();
```

```
Povtor2:
    S = Interaction.InputBox("Введіть n ", "Введення", S);
    try
    {
        n = Convert.ToDouble(S);
    }
    catch (System.FormatException)
    {
        if (MessageBox.Show("Ви ввели не число" + Strings.Chr(13) + Strings.Chr(13) + "Бажаєте повторити?",
                            "Увага", MessageBoxButtons.YesNo, MessageBoxIcon.Warning) == DialogResult.Yes)
            goto Povtor2;
        else
            return;
    }
    //розв'язування задачі
    x = (m - 3 * n) / 7;
    y = (2 * m + n) / (3 - n * n);
    z = x * y;
    rez = "";
    count = 0;
    if (x < -5 || x>7)
    {   rez = x.ToString();
        count++;
    }
    if (y < -5 || y > 7)
    {   if (rez != "")
            rez += ", ";
        rez+= y.ToString();
        count++;
    }
    if (z < -5 || z > 7)
    {   if (rez != "")
            rez += ", ";
        rez += z.ToString();
        count++;
    }
    rez = rez == "" ? " відсутні" : ": " + rez;
    //вивід результатів
    if (MessageBox.Show("При m = " + m.ToString() + " та n = " + n.ToString() + "\nобчислено: x=" +
                        x.ToString() + ", y=" + y.ToString() + ", z=" + z.ToString() +
                        ".\nСеред них числа поза проміжком [-5;7]" + rez+ ".\nЇх кількість: " +
                        count.ToString()+"." +Strings.Chr(13) + Strings.Chr(13) +
                        " Бажаєте повторити?", "Результати обчислень", MessageBoxButtons.YesNo,
                        MessageBoxIcon.Information) == DialogResult.Yes)
        goto Povtor;
}
```