Лабораторна робота № 2

Програмування лінійних обчислювальних процесів

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

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

знання:

класифікацій базових типів даних та їх основні характеристики;

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

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

пріоритетів операцій;

правил перетворення типів;

основних бібліотечних математичних функції мови C#;

уміння:

складати лінійні програми з використанням стандартних бібліотечних функцій;

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

комунікації:

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

автономність і відповідальність:

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

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

Основні положення

Лексичні елементи мови C#.

Будь яка алгоритмічна мова містить три складові частини: алфавіт (кінцева множина відмінних між собою символів, що використовуються у даній мові); синтаксис (сукупність правил, що визначають припустимі (правильні) конструкції даної мови. Синтаксис мови C# визначений у

спеціальній граматиці); семантика (сукупність правил, що визначають значеннєвий зміст окремих конструкцій. Семантика забезпечує однозначність тлумачення всіх понять мови).

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

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

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

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

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

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

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

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

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

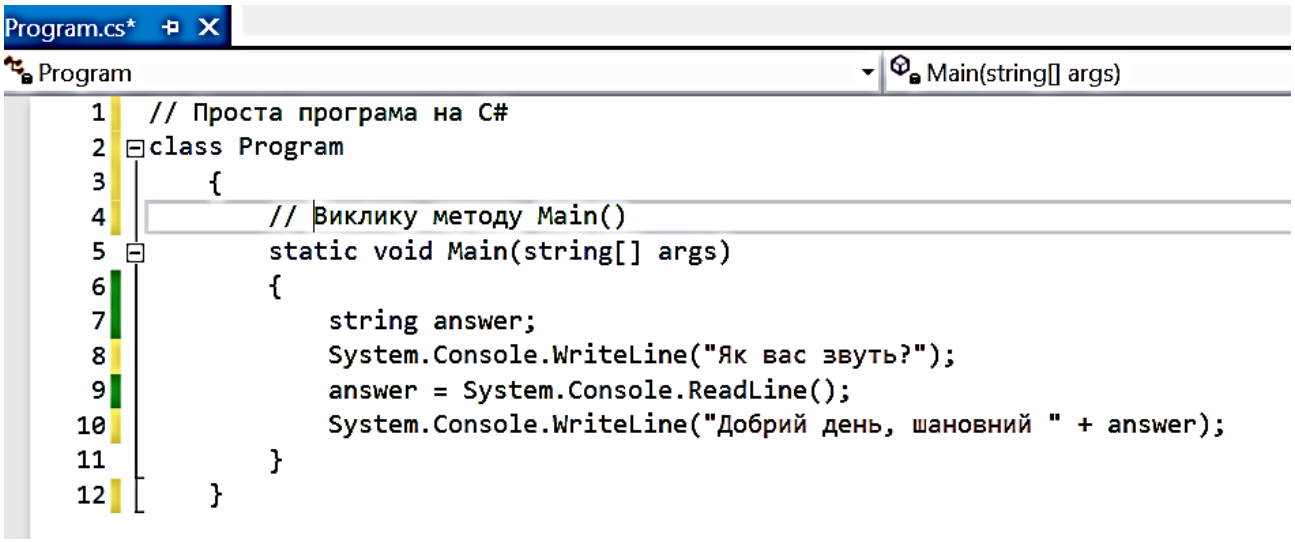

Рис. 25. **Базова структура C# програми**

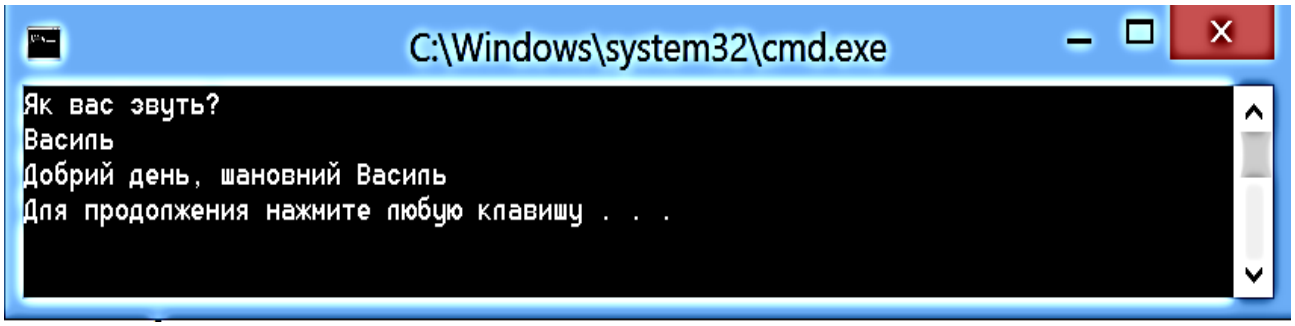

Рис. 26. **Результат виконання базової структури програми**

Слід розглянути кожну частину програми більш докладно.

Коментарі.

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

01: // Проста програма на C#

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

class Program // Проста програма на C#

Інший варіант коду некоректний:

// Проста програма на C# class Program

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

Визначення класу.

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

У рядку 2 для визначення класу використовується ключове слово class.

02: class Program

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

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

Ідентифікатори (імена).

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

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

Фігурні дужки та блоки вихідного коду.

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

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

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

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

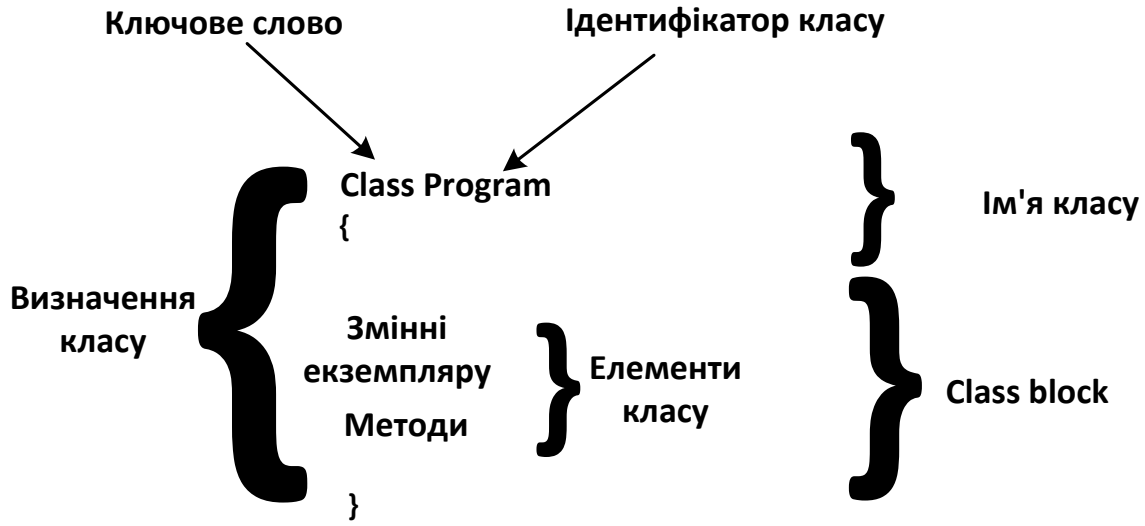

Рис. 27. **Визначення класу**

У рядку 4

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

знаходиться вже знайомий символ коментаря //.

Метод Main() та його визначення.

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

```
05:     static void Main(string[] args)
```

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

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

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

Основні елементи визначення методу ілюструються на рис. 27.

Кожна програма на C# повинна містити метод Main().Під час запуску середовище виконання .NET, у першу чергу, шукає цей метод.

Якщо він знайдений, з нього починається виконання, якщо ні – виводиться повідомлення про помилку.

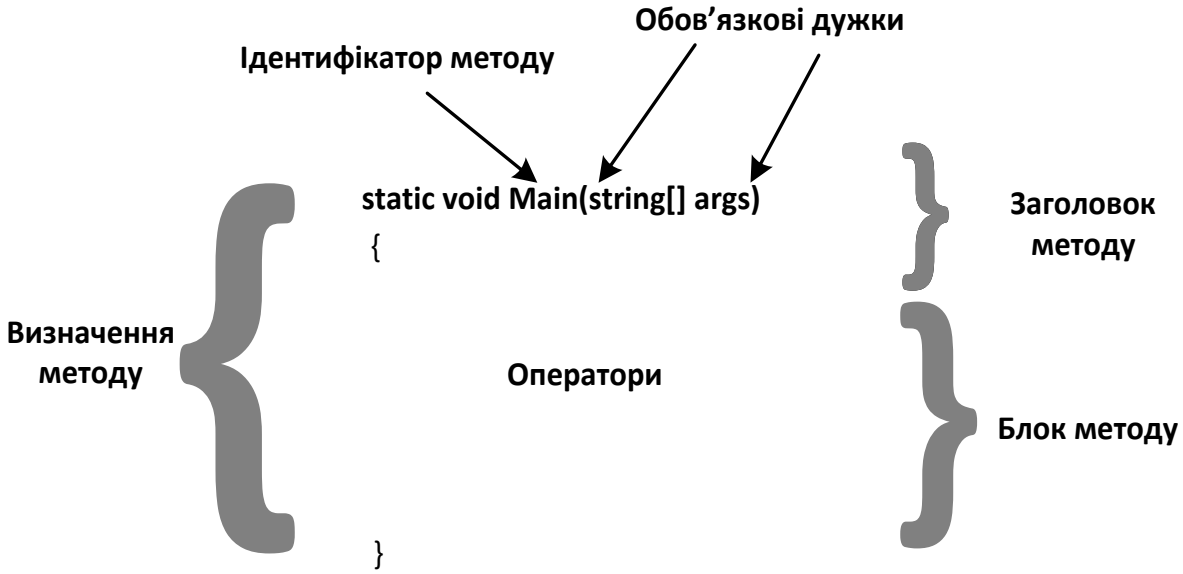

Рис. 28. **Визначення методу**

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

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

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

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

Щоб зрозуміти значення ключового слова void в рядку 5, необхідно звернутися безпосередньо до того, як працюють методи. У цьому розділі

буде наведено лише коротке пояснення: void означає, що Main() не повертає значення в точку виклику.

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

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

Змінні.

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

```
07:     string  answer;
```

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

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

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

Змінна answer застосовується в рядках 9 та 10.

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

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

ідентифікатора (в даному випадку, answer);

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

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

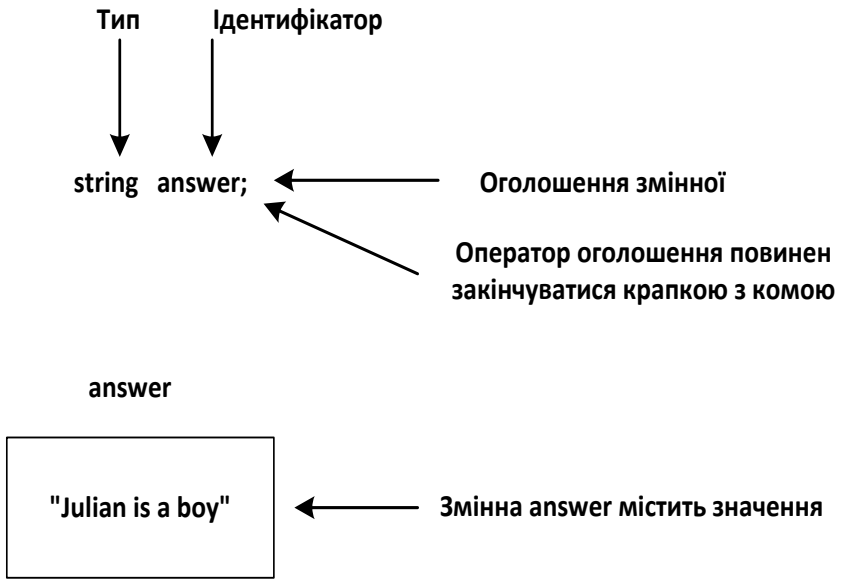

Рис. 29. **Тип, ідентифікатор та значення змінної**

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

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

Запуск методів .NET-платформи.

Оператор в рядку 8

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

змушує програму вивести на екран таке:

Як вас звуть?

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

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

Коли метод виконує певну задачу в програмі, це називають викликом. Елемент усередині круглих дужок (текст " Як вас звуть? " в

прикладі) називається аргументом. Аргумент містить інформацію, необхідну методу, що викликається, для виконання завдання. Аргумент передається методу WriteLine під час виклику. Після цього метод звертається до даних вже за допомогою своїх внутрішніх операторів. Рядок 8, як і 7, містить оператор і тому закінчується крапкою з комою.

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

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

Загальний механізм виклику методу.

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

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

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

Виклик нестатичного методу:

ІмяОб’єкта.ІмяМетоду(Необов'язкові_аргументи)

Виклик статичного методу:

ІмяКласу.ІмяМетоду (Необов'язкові_аргументи)

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

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

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

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

На рисунку 30 можна виділити такі кроки:

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

2. Запустити рядок 8.

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

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

5. Повернути керування операторові в рядку після опреатора 8.

6. Виконати інші оператори методу Main.

Присвоювання значення змінної.

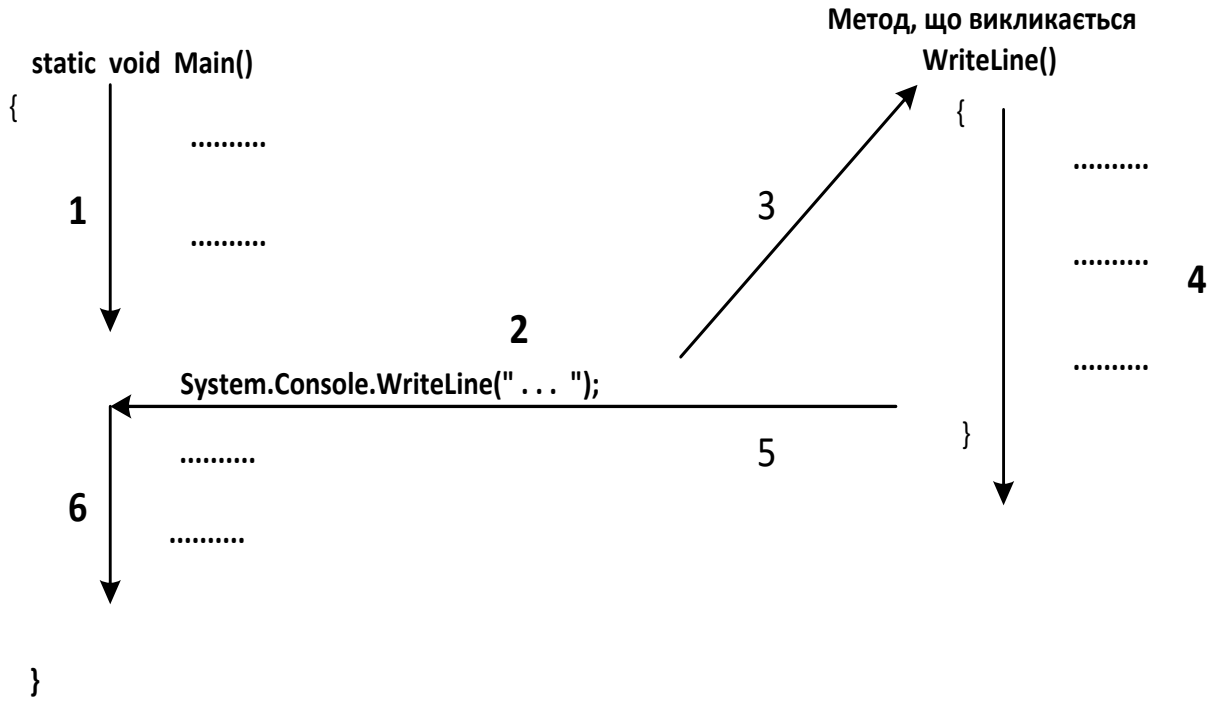

Рис. 30. **Рух потоку виконання програми під час виклику метода**

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

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

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

Механізм задання нового значення змінній answer називається присвоюванням. Говорять, що введений текст присвоюється змінній answer. Загальне вираження в рядку називають оператором

присвоювання, а сам знак рівності (=) називають операцією присвоювання (в даному контексті). Якщо знак «рівність» використовується в інших контекстах, він має інші назви.

Завершення методу Main() і класу Program.

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

11: }

У рядку 12 дужка } завершує блок класу Program.

12: }

Формат вихідного коду C#.

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

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

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

Поняття типу даних

Концепція типу даних.

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

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

ім'я змінної або константи;

розмір пам'яті, необхідної для зберігання значень;

які дії можна виконувати зі змінною або константою;

вид та спосіб виділення пам'яті;

початкове значення змінної або значення константи.

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

Класифікація типів даних.

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

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

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

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

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

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

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

Усі класи є посилальними типами.

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

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

Особливості застосування простих типів.

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

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

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

Діапазон значень змінної. Наприклад, діапазон типу int: від -2147483648 до 2147483647.

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

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

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

Таблиця 1

Прості типи в мові C#

Ключове слово мови C#	Tun .NET CTS	Вид значення	Використовувана пам'ять	Діапазон і точність
1	2	3	4	5
sbyte	System.SByte	Ціле число	8 бітів	Від -128 до 127
byte	System.Byte	Ціле число	8 бітів	Від 0 до 255
short	System.Int16	Ціле число	16 бітів	Від -32 768 до 32 767
ushort	System.Ulnt16	Ціле число	16 бітів	Від 0 до 65 535
int	System.Int32	Ціле число	32 біти	Від -2 147 483 648 до 2 147 483 647
uint	System.Ulnt32	Ціле число	32 біти	Від 0 до 4 294 967 295

Закінчення табл. 1

1	2	3	4	5
long	System.Int64	Ціле число	64 біти	Від **-9 223 372 036 854 775 808** до **9 223 372 036 854 775 807**

ulong	System.Ulnt64	Ціле число	64 біти	Від 0 до **18 446 744 073 709 551 615**
char	System.Char	Ціле число (один символ)	16 бітів	Всі символи Unicode
float	System.Single	Число з плаваючою точкою	32 біти	Від (+/-)1.5 * 1045 до (+/-)3.4 * 1038 Приблизно 7 значущих цифр
double	System.Double	Число з плаваючою точкою	64 біти	Від (+/-) 5.0 * 10-324 до (+/-)3.4 * 1030 15-16 значущих цифр
decimal	System.Decimal	Десяткове число (високої точності)	128 бітів	Від (+/-) 1.0 * 10-28 до (+/-)7.9 * 1028
bool	System.Boolean	true або false	1 біт	Немає

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

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

Перетворення вбудованих типів.

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

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

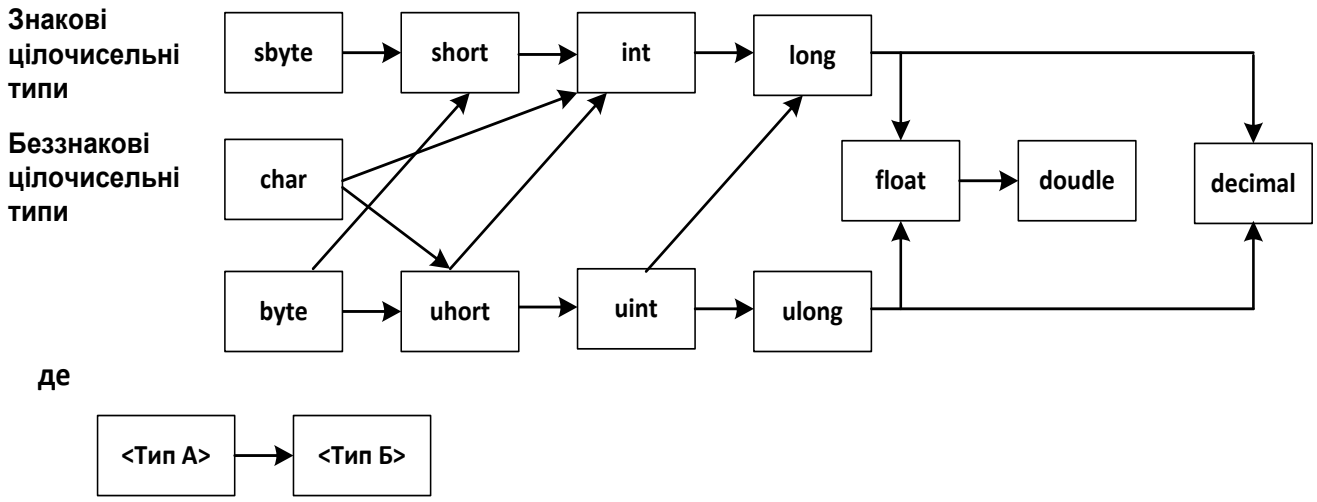

Рис. 31. **Шляхи неявного перетворення числових типів**

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

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

Якщо робиться зворотне перетворення, то, звичайно ж, можна втратити інформацію. Якщо значення в int більше, ніж 32.767, воно буде усічене під час перетворення. Компілятор не стане виконувати неявне перетворення від int до short:

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

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

Програміст повинен виконати явне перетворення, використовуючи оператор приведення:

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

Константні величини.

Часто у вихідній програмі використовуються фіксовані числа, наприклад: 3.141592 або інші величини, які не змінюються протягом

виконання програми. Мова C# дозволяє оголошувати імена для літералів або інших виражень і використовувати їх замість запису фактичного значення.

Наприклад, замість

distance = secondsTraveled * 186000;

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

distance = secondsTraveled * SpeedOfLight;

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

int SpeedOfLight = 186000;

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

const int SpeedOfLight = 186000;

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

Константи поділяються на такі групи:

- Числові
  - Цілі
  - Речовинні
- Перелічувані
- Символьні (літерні)
  - Клавіатурні
  - Кодові (керуючі або розділові символи)
  - Кодові числові
- Строкові (рядки або літерні рядки)
- Іменовані (символічні)

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

Розрізняють такі символьні константи.

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

Наприклад, char t = 'd'; //

Кодові (таблиця 2) – для завдання деяких керуючих і розділових символів, наприклад, ‘\n’, ‘\t’;

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

Таблиця 2

Керуючі послідовності

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

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

char t = '\x1A' ; // код символу «←»

Програмування лінійних обчислювальних процесів

Основні операції мови C#.

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

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

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

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

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

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

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

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

Операції відношення дозволяють порівнювати значення, наприклад sum < 100.

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

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

Побітові операції дозволяють працювати з окремими бітами відповідних операндів.

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

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

distance1 + distance2,

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

distance1 * 100 + distance2 * 300.

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

Пріоритети операцій.

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

Слід розглянути вираз:

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

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

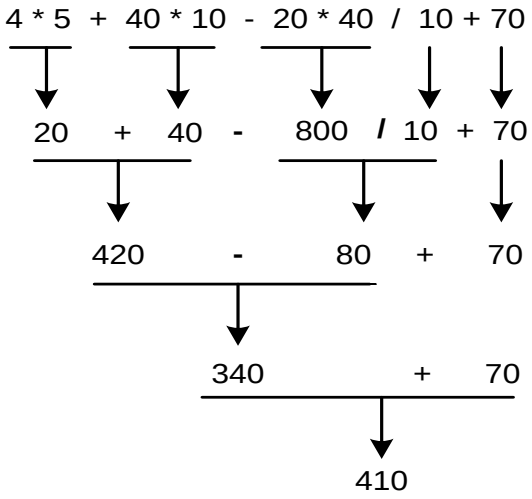

Рис. 32. **Черговість виконання операцій під час обчислення виразу**

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

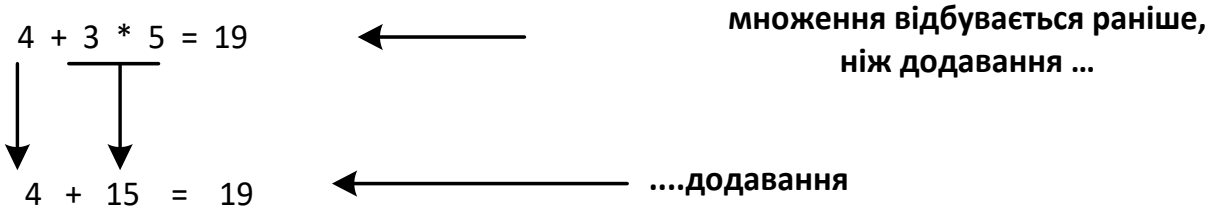

Рис. 33. **Множення відбувається до додавання**

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

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

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

Огляд операцій та їхніх пріоритетів наведено у табл. 3.

Форматування числових значень

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

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

При цьому на консоль відображається таке:

Distance traveled: 10000000.432

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

Стандартне форматування.

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

Таблиця 3

Пріоритети операцій

Категорії	Операції	Асоціативність	Значення
1	2	3	4

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

Продовження табл. 3

1	2	3	4
Адитивні	< Вираз1> + < Вираз2>	Зліва направо	Додавання
	<< Вираз1> - < Вираз2>	Зліва направо	Віднімання

Відно-шення	< Вираз1> < < Вираз2> < Вираз1> <= < Вираз2> < Вираз1> > < Вираз2> < Вираз1> >= < Вираз2> <Об'єкт> is <Тип>	Зліва направо Зліва направо Зліва направо Зліва направо Зліва направо	Менше Менше або дорівнює Більше Більше або дорівнює Приналежність типу
Рівність	< Вираз1> == < Вираз2> < Вираз1> != < Вираз2>	Зліва направо Зліва направо	Дорівнює Не дорівнює
Логічні побітові	< Логічний_вираз1> & < Логічний_вираз2> < Логічний_вираз1> ^ < Логічний_вираз2> < Логічний_вираз1> | < Логічний_вираз2>	Зліва направо Зліва направо Зліва направо	Побітове І Побітове АБО, що виключає Побітове АБО
Логічні	< Логічний_вираз1> && < Логічний_вираз2> < Логічний_вираз1> || < Логічний_вираз2> < Логічний_вираз> ? < Вираз1> : < Вираз2>	Зліва направо Зліва направо Зліва направо	Логічне І Логічне АБО Умовна операція

Закінчення табл. 3

1	2	3	4
Присвоювання	<Змінна> = < Вираз>	Справа наліво	Просте присвоювання

	<Змінна> *= < Вираз>	Справа наліво	Множення і присвоювання
	<Змінна> /= < Вираз>	Справа наліво	Ділення і присвоювання
	<Змінна> % = < Вираз >	Справа наліво	Ділення за модулем і присвоювання
	<Змінна> += < Вираз >	Справа наліво	Додавання і присвоювання
	<Змінна> - = < Вираз >	Справа наліво	Вирахування і присвоювання

Таблиця 4

Приклади форматування чисел

Ім'я змінної	Число	Відформатоване число
distance	7 000 000 000 000 000	7.00E+015
Mass	3.87839029837898773 62	3.8784
Length	20 000 000	20,000,000

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

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

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

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

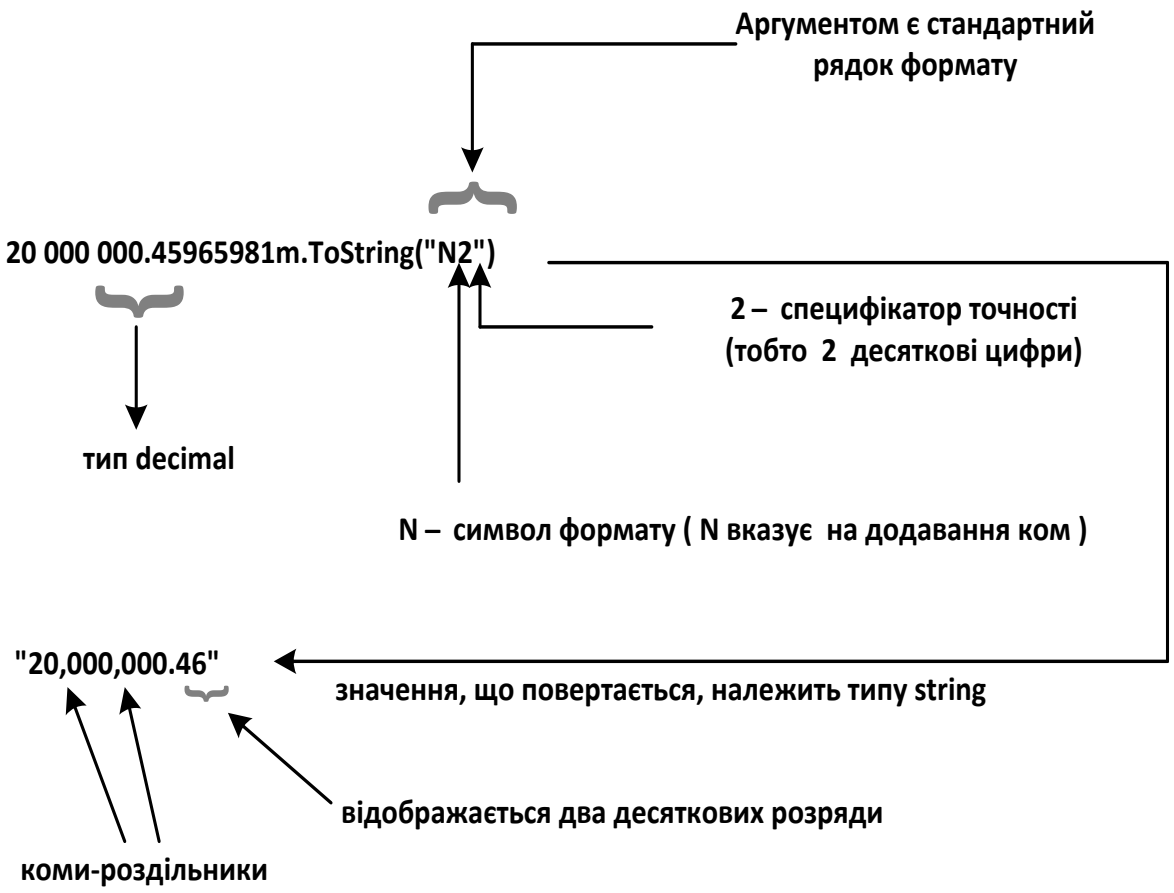

Рис. 34. **Форматування літерала типу decimal**

Таблиця 5

C# символи формату

Символи	Опис	Приклади
1	2	3
C, c	Валюта. Форматування, специфічне для настроювань локалізації. Вони містять інформацію про тип грошової одиниці й інших пара-метрів, які можуть змінюватися залежно від країни	2000000.456m.ToString("C") Повертає "$2,000,000,46" (Якщо операційна система налаштована відповідно до американських стандартів)

Продовження табл. 5

1	2	3
D, d	Ціле число. Специфікатор точності встановлює мінімальне число цифр. Виведення доповнюється ведучими нулями, якщо кількість цифр фактичного числа менше, чим специфікатор точності. Примітка: цей символ формату застосовується тільки для цілочисельних типів)	45687.ToString("D8") Повертає: "00045678"
E, e	Експонентна (наукова) нотація. Специфікатор точності визначає кількість десяткових цифр, за замовчуванням рівне 6	345678900000.ToString("E3") Повертає "3.457E+011" 345678912000.ToString("e") Повертає "3.456789e011"
F, f	Фіксована точка. Специфікатор точності вказує кількість десяткових цифр	3.7667892.ToString("F3") Повертає "3.767"
G, g	Загальний. Найбільш компактний формат під час вибору E або F. Специфікатор точності встановлює максимальну кількість цифр у поданні числа	65432.98765.ToString("G") Повертає "65432.98765" 65432.98765.ToString("G7") Повертає "65432.99" 65432.98765.ToString("G4") Повертає "6.543E4"
N, n	Число. Число з комами-роздільниками. Специфікатор точності встановлює кількість десяткових цифр	1000000.123m.ToString("N2") Повертає "1,000,000.12"

Закінчення табл. 5

1	2	3
X, x	Шістнадцятиричне число. Специфікатор точності встановлює мінімальну кількість цифр, що подаються в рядку. Для досягнення певної ширини додаються ведучі нулі	950.ToString("x") Повертає "3b6" 950.ToString("X6") Повертає "0003B6"

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

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

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

На екран виводиться таке:

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

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

Мова C# надає рішення цієї проблеми.

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

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

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

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

Mass: 100 Distance 50 Energy 35

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

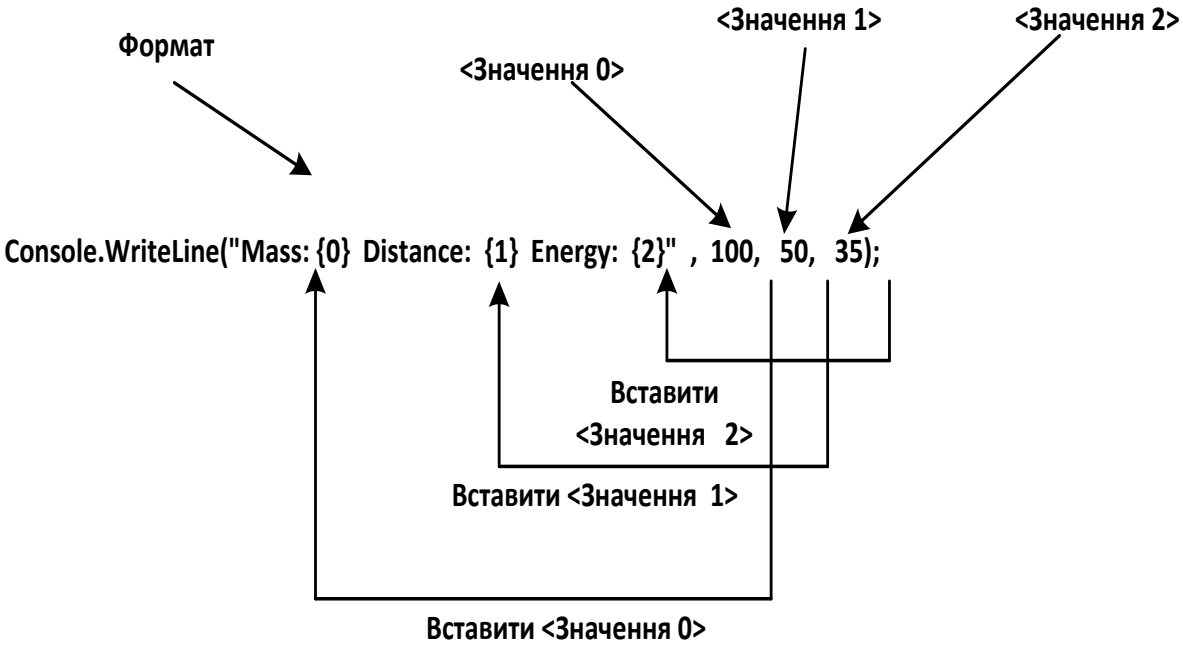

Рис. 35. **Приклад виклику методу Console.WriteLine**

Позитивне число означає вирівнювання праворуч, а негативне – ліворуч.

Наприклад:

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

виводить

Distance: 100 miles

ширина дорівнює 10 символів і число вирівняне праворуч,

тоді, як код

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

негативне число вказує на вирівнювання ліворуч.

Виводить:

Distance: 100 miles

ширина дорівнює 10 символів і число вирівняне ліворуч.

Порядок виконання лабораторної роботи

Загальна частина.

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

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

Слід проекспериментувати з програмами:

• змінити вихідні дані;

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

Індивідуальна частина.

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

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

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

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

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

6. Дослідити роботу програми, аналізуючи виконання контрольних прикладів.

Зміст звіту

1. Титульний лист.

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

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

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

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

6. Висновки.

Контрольні запитання

1. Опишіть відомі вам алгоритмічні структури.
2. Дайте огляд основних операцій C#.
3. Навіщо потрібні логічні операції? Наведіть приклади.
4. Що таке пріоритети операцій? Де і коли вони використовуються?
5. Що таке асоціативність операцій? Де і коли вони використовуються?
6. Розкрийте суть понять: простір імен, область видимості змінних, область видимості і час існування змінних.
7. Напишіть програму обчислення суми, добутку, максимуму і мінімуму трьох чисел.
8. Як перетворити значення типу string в тип int?
9. Опишіть загальну схему створення і виклику призначених для користувача методів.
10. Охарактеризуйте клас Math. Наведіть приклад програми, де використовуються вбудовані математичні методи.