### Проста програма на C#

Цей розділ буде основою для вивчення C#. У ньому розглядатимуться базові принципи програмування цією мовою, які ілюструватимуться простими прикладами програм, розроблених у середовищі Microsoft Visual Studio. Для розуміння принципів роботи цих програм розглянемо структуру найпростішої консольної програми на C# та загальне призначення різних її елементів. В наступних розділах деякі з цих елементів розглянемо детальніше.

Щоб створити проект консольної програми, у Visual Studio слід вибрати команду меню File\ New\ Project, і вибрати шаблон Visual C#\ Windows\ Console Application. Середовище згенерує проект консольного застосунку з текстом програми як у лістингу 2.1, за винятком рядків 9-10, які містять текстовий коментар та виклик методу WriteLine[[1]](#footnote-1).

|  |  |
| --- | --- |
| Лістинг 2.1. Файл Program.cs найпростішого консольного застосунку  OurFirstApplication | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | using System; namespace OurFirstApplication { class Program { static void Main(string[] args) {  // Наступний рядок слід дописати самому  Console.WriteLine("Наша перша програма :)");  }  }  } |

Під час роботи ця програма виводить у консольне вікно єдину текстову фразу – "Наша перша програма :)" (рис. 2.1). Запустити програму з середовища Microsoft Visual Studio можна у двох режимах:

► У режимі налагодження (натиснувши F5). При цьому, коли консольна програма закінчує роботу, її вікно відразу закривається. І якщо код не містить спеціальних інструкцій очікування, побачити результат роботи програми не вдасться.

► Без налагодження (натиснувши Ctrl+F5). У цьому випадку, після відпрацювання консольної програми, середовище не закриває її вікна, а очікує натискання довільної клавіші клавіатури (про що повідомляє фразою "Press any key to continue…" – див. рис. 2.1).

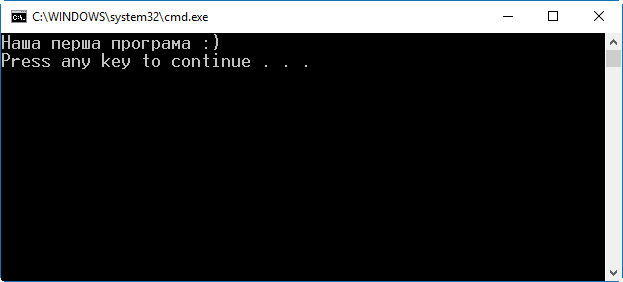


Рис. 2.1. Результат роботи програми з лістингу 2.1

Перевагою режиму налагодження є можливість використання спеціальних засобів для дослідження роботи програми (точок переривання програми, покрокове виконання програми тощо). Тому під час розробки програм найчастіше використовують саме його. Але ми розглядатимемо дуже прості (і готові) приклади, тому будемо приводити знімки екрану консольних програм, запущених без під'єднання налагоджувача (через натискання Ctrl+F5 або команди меню Debug\ Start Without Debugging).

Розглянемо код (лістинг 2.1) детальніше. Він розміщений у файлі, який за замовчуванням називається Program.cs. Рядок 9 починається з двох символів "//": так у коді позначають коментар. Текст після цих символів до кінця рядка може бути будь-яким – компілятор його не розглядає як складову програми. Таким способом у текст програми вписують пояснення до тих чи інших ділянок коду.

У таблиці 2.1 приведено детальний порядковий опис коду з лістингу 2.1.

Таблиця №2.1. Опис коду програми з лістингу 2.1

|  |  |
| --- | --- |
| **Номер рядка** | **Опис** |
| **1** | Повідомляє компілятор, що програма використовує типи з простору імен System. |
| **3** | Оголошення нового простору імен з назвою OurFirstApplication. Цей простір імен починається відкриваючою фігурною дужкою у рядку 4 і закінчується закриваючою фігурною дужкою в рядку 13. Всі типи, оголошені у вказаних межах, є елементами простору імен OurFirstApplication. |
| **Номер рядка** | **Опис** |
| **5** | Оголошення нового класу з назвою Program. Все, що оголошене між парою фігурних дужок у рядках 6 та 12, належить до цього класу і формує його структуру та поведінку. |
| **7** | Оголошення методу Main, елемента класу Program. У цій програмі Main є єдиним елементом класу Program. Функція з назвою Main використовується як початкова точка програми. Реалізація методу Main знаходиться між парою відповідних фігурних дужок у рядках 8 та 11. |
| **10** | Містить єдину просту інструкцію. Цей рядок становить тіло метода Main. У C# кожна інструкція закінчується крапкою з комою. Інструкція в цьому рядку використовує клас із назвою Console, який розміщений у просторі імен System, для виводу рядка у консольне вікно. Без використання інструкції using в рядку 1 компілятор не знав би, де шукати клас Console. Метод WriteLine() класу Console використовують для виведення тексту у вікно. |

Програма на C# складається з одного чи більше оголошень типів, які можуть бути розташовані в довільному порядку. У програмі OurFirstApplication оголошено тільки один тип – клас Program.

#### Функція Main: початкова точка програми

Кожна програма на C# повинна мати один клас з методом із назвою Main (перша літера – велика, на відміну від C та C++). У програмі з лістингу 2.1 цей метод оголошено у класі Program. Виконання будь-якої програми на C# починається з першої інструкції метода Main.

Найпростіша форма цього методу така:

static void Main() {

// Інструкції коду

}

#### Текстовий ввід/ вивід у консольному застосунку

*Консольне вікно* (рис. 2.1) – просте вікно з командним рядком, яке дозволяє здійснювати текстовий ввід-вивід за допомогою клавіатури. Бібліотека базових класів .NET Framework (BCL) підтримує клас Console (в просторі імен System), який містить методи для введення та виведення текстових даних у консольному вікні. Таке вікно містить *курсор* – блимаючий маркер, який позначає поточну позицію у вікні. Ввід та вивід текстових даних здійснюється саме у позиції курсора.

Для виводу тексту в консольне вікно використовують методи (функції) Write та WriteLine класу Console.

#### Метод Write

Метод Write виводить у консольне вікно застосунку переданий йому рядок тексту, починаючи з поточної позиції курсора. У C# (як і в C та C++) значення типу "рядок" має бути оточене подвійними лапками.

Наступний рядок коду демонструє використання метода Write:

Console.Write("Це просто текст");

Рядок тексту для виводу у вікно

Метод

Write

класу

Console

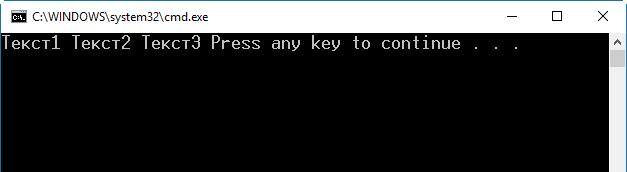
Якщо цей рядок коду ввести у метод Main (замість рядка 10) у лістингу 2.1, то застосунок виведе у вікно фразу "Це просто текст". Після виводу тексту за допомогою методу Write не здійснюється перехід до наступного рядка. Наприклад, послідовний вивід трьох текстових фраз в одному рядку можна здійснити так:

Console.Write("Текст1 ");

Console.Write("Текст2 ");

Console.Write("Текст3 ");

Результат виконання цього коду приведено на рис. 2.2.



Перша

інструкція

Друга

інструкція

Третя

інструкція

Рис. 2.2. Вивід трьох фраз за допомогою методу Write

#### Метод WriteLine

Метод WriteLine, який також є елементом класу Console, виконує те ж, що і метод Write, але додає у кінці виведеного рядка символ переходу до наступного рядка. Якщо переписати попередній фрагмент коду, замінивши Write на

WriteLine, то отримаємо результат, як на рис. 2.3:

Console.WriteLine("Текст1 ");

Console.WriteLine("Текст2 ");

Console.WriteLine("Текст3 ");

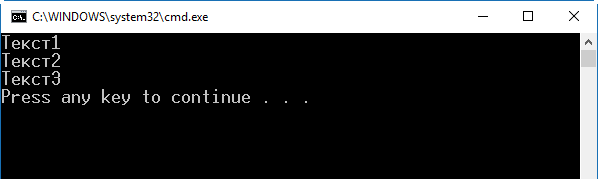


Рис. 2.3. Вивід трьох фраз за допомогою методу WriteLine

При виведенні рядків їх можна поєднувати один з одним та з числовими значеннями за допомогою знаку "+":

Console.WriteLine("Частота: " + 2.4 + " ГГц");

В результаті виконання цього коду у вікно буде виведено рядок: "Частота: 2.4 ГГц".

#### Форматування рядків

Текст при виведенні за допомогою методів Write та WriteLine можна відразу форматувати. У цьому випадку методу передають кілька параметрів і розділяють їх комами. Перший параметр завжди повинен містити рядок, який називають *рядком формату*. Рядок формату повинен містити *маркери підстановки*, які показують позиції, у яких будуть підставлені певні значення. Маркер підстановки є натуральним (невід'ємним цілим) числом у фігурних дужках, яке задає номер значення для підстановки. Маркери нумерують, починаючи з нуля. Значення для підстановки задають у параметрах після рядка формату.

Загальний синтаксис виклику методу для виведення форматованого рядка такий:

Console.WriteLine(FormatString, Value0, Value1, Value2...); де FormatString – рядок формату, ValueN – значення для підстановки.

Наприклад, подана нижче інструкція містить виклик методу WriteLine з рядком формату (він містить три маркери підстановки) та трьома значеннями (10, 20 і 30) для підстановки:

Console.WriteLine("Три числа: {0}, {1} та {2} ", 10, 20, 30);

Рядок формату

Значення для підстановки

Маркери підстановки

Результат виводу цього фрагмента коду приведено на рис. 2.4.

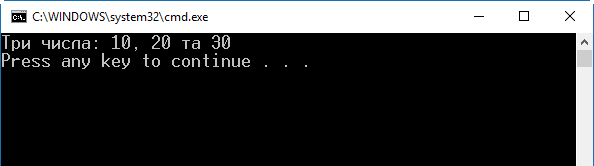


Рис. 2.4. Вивід рядка з форматуванням

У рядках формату може бути довільна кількість маркерів підстановки. При цьому їх можна розміщувати в довільному порядку, а маркер з тим же номером можна використовувати кілька разів.

Маркер підстановки не повинен звертатися до елемента за межами переліку значень для підстановки: для кожного маркера з індексом N має існувати значення у переліку параметрів з індексом N. Якщо задати значень менше, ніж є маркерів, то програма скомпілюється і запуститься успішно, але під час виконання трапиться помилка (виняткова ситуація) з описом: "Index (zero based) must be greater than or equal to zero and less than the size of the argument list" (індекс (з основою нуль) має бути більшим або дорівнювати нулю, і меншим за розмір списку аргументів). Тому виконання наступного рядка коду приведе до помилки:

Console.WriteLine("!Помилка!: {0}, {2} ", 10, 20);

Значення з індексом 2 не існує

Індекс 0

Індекс 1

#### Інтерполяція рядків

Починаючи з версії C# 6.0 синтаксичні конструкції для форматування можна спростити, застосувавши *інтерполяцію рядків (string interpolation)*. Інтерполяція рядків робить код легшим для сприйняття.

Щоб вказати компілятору, що рядок слід інтерполювати, перед ним розміщують символ **$**. У самому рядку безпосередньо вказують назви змінних, оточуючи їх фігурними дужками:

int number1 = 10; int number2 = 20; Console.WriteLine($"Два числа: {number1} та {number2}");

Перше значення

Друге значення

### Ідентифікатори

*Ідентифікатори (identifiers)* – це символьні слова, які використовують для позначення змінних, констант, методів, параметрів та багатьох інших програмних елементів, які розглядатимемо далі.

Прийнято давати самоописувальні ідентифікатори, через поєднання значущих слів у єдину описову назву та розділенням їх за допомогою регістру символів (наприклад, BookId, SecurityNum, FirstName тощо).

Для формування ідентифікаторів можна використовувати символи літер, цифр, а також символи "\_" та "@". Сучасні компілятори підтримують використання національних символів для ідентифікаторів, але з метою уникнення плутанини та переносимості коду рекомендовано використовувати тільки символи латиниці.

При формуванні ідентифікаторів слід пам'ятати такі правила:

► Символи літер та підкреслення "\_" можна розміщувати у будь-якій позиції.

► Символи цифр не можна розміщувати на початку ідентифікатора, але в будь-якому іншому місці – можна.

► Символ "@" можна розміщувати тільки на початку ідентифікатора, і не можна в інших місцях.

Ідентифікатори у C# (як і в C та C++) є чутливими до регістра. Наприклад, ідентифікатори myInt, MyInt та myint позначають різні змінні.

Проте, використання дуже схожих імен погіршує розуміння коду, сприяє допущенню помилок і робить налагодження програми важчим. Тому застосовувати такі ідентифікатори не рекомендовано: це вірно синтаксично, але заплутує код:

// Так робити не варто, хоча синтаксично це правильно int totalSumm; int TotalSumm; int Totalsumm; TotalSumm = totalSumm + Totalsumm;

Використовують різні стилі іменування[[2]](#footnote-2) ідентифікаторів, які відрізняються використанням регістру символів (таблиця 2.2).

Часто для назв також використовують символ підкреслення "\_":

► Для групи "особливих" ідентифікаторів назву починають підкресленням: \_GetName, \_GetObjectType.

► Окремі слова в ідентифікаторі розділяють символом "\_": Department\_Id, Language\_Code, department\_id, language\_code.

В подальшому у прикладах коду будемо дотримуватися застосування стилів іменування за таблицею 2.2.

Таблиця №2.2. Стилі іменування

|  |  |  |  |
| --- | --- | --- | --- |
| **Назва стилю** | **Опис** | **Рекомендовано використовувати для:** | **Приклади** |
| Pascal | Кожне слово в ідентифікаторі починають з великої літери | Назв типів, класів та їх елементів | TotalSumm,  DepartmentId |
| Camel | Кожне слово в ідентифікаторі, крім першого, починають з великої літери | Локальних змінних та параметрів | totalSumm, departmentId |
| Uppercase | Всі літери великі | Тільки для абревіатур | TS, AXML |

### Ключові слова

*Ключове слово (keyword)* – це формальний символьний рядок, який визначено мовою програмування. Ключові слова не можна використовувати в ролі ідентифікаторів (тобто, для назв змінних методів тощо). Якщо за логікою програми потрібно використати ідентифікатори, які співпадають з ключовими словами C#, їх можна доповнювати символами "@" чи "\_" (наприклад, @float,

@int, float\_, int\_).

Всі ключові слова C# містять тільки малі літери. Перелік ключових слів C# приведено у таблиці 2.3.

Таблиця №2.3. Ключові слова мови C#

|  |  |  |  |
| --- | --- | --- | --- |
| abstract | as | base bool | |
| break | byte | case | catch |
| char | checked | class | const |
| continue | decimal | default | delegate |
| do | double | else | enum |
| event | explicit | extern | false |
| finally | fixed | float | for |
| foreach | goto | if | implicit |
| in | in (generic modifier) | int | interface |
| internal | is | lock | long |
| namespace | new | null | object |
| operator | out | out (generic modifier) | override |
| params | private | protected | public |
| readonly | ref | return | sbyte |
| sealed | short | sizeof | stackalloc |
| static | string | struct | switch |
| this | throw | true | try |
| typeof | uint | ulong | unchecked |
| unsafe | ushort | using | virtual |
| void | volatile | while |  |

Існують також *контекстні ключові слова*. Вони діють як ключові слова тільки в деяких конструкціях мови програмування. В інших місцях вони діють як звичайні ідентифікатори. Таблиця 2.4 містить перелік контекстних ключових слів мови C#.

Таблиця №2.4. Контекстні ключові слова мови C#

|  |  |  |
| --- | --- | --- |
| add | alias | ascending |
| async | await | descending |
| dynamic | from | get |
| global | group | into |
| join | let | orderby |
| partial (type) | partial (method) | remove |
| select | set | value |
| var | where (generic type constraint) | where (query clause) |
| yield |  |  |

### Простори імен

*Простір імен* *(namespace)* – це сукупність оголошень типів, співставлених та пов'язаних з певною назвою. Програма з лістингу 2.1 використовує два простори імен. По-перше, вона створює новий простір імен OurFirstApplication (за замовчуванням його назва співпадає з назвою проекту, яку задали при його створенні). У ньому оголошено власний тип – клас Program. По-друге, програма використовує простір імен System, клас Console з якого вжито для виводу тексту у вікно.

Простір імен оголошують так:

namespace TheNameOfNameSpace {

//Елементи простору імен

}

TheNameOfNameSpace – це назва простору імен. Між фігурними дужками описують елементи простору імен – типи та класи. Простір імен є контейнером, у якому описують різні типи. Він забезпечує заданий програмістом вид групування коду, тому в одному просторі імен доцільно групувати логічно чи функціонально пов'язані типи.

Звертання до типу в просторі імен роблять через його назву, розділюючи їх ідентифікатори крапкою:

TheNameOfNameSpace.ClassName

Назва простору імен

Назва типу (класу)

Якщо код має використовувати типи з певного простору імен, то, щоб постійно не вказувати назву простора імен, можна вказати компілятору, що ми використовуємо даний простір імен. Для цього використовують ключове слово using:

using TheNameOfNameSpace;

Ключове слово

Назва простору імен

Тоді звертатися до елемента простору імен можна коротше, опускаючи його назву:

using System; // Директива використання простору імен ...

// Використовуємо повну назву, яка містить назву простору імен

System.Console.WriteLine("This is the text 1");

// Назву простору імен опускаємо

Console.WriteLine("This is the text 2");

Зауважимо, що починаючи з C# 6.0 є ще зручніший спосіб використання методів класу Console – за допомогою інструкції using static: using static System.Console;

Тоді у коді можна навіть не вказувати назви класу Console:

WriteLine("This is the text");

### Пропуски у коді

Пропуски у коді програми формуються символами тексту, які не відображуються на екрані. Компілятор ігнорує пропуски, але вони потрібні і використовуються програмістами для того, щоб зробити програму зрозумілішою, краще структурованою і легшою для читання.

Пропуски формують:

► Пробілами.

► Табуляцією.

► Переходом до наступного рядка.

Розглянемо два фрагменти коду:

1. 1 class Program
   1. {
   2. static void Main(string[] args)
   3. {
   4. Console.WriteLine("Наша перша програма :)");
   5. }
   6. }
2. 1 class Program{static void Main(string[] args){ 2 Console.WriteLine("Наша перша програма :)");}}

З точки зору компілятора вони ідентичні: однаково компілюватимуться і однаково виконуватимуться. Але перший фрагмент значно легше читається: відступи зліва формують інтуїтивне сприйняття вкладеності блоків коду. Тому відразу зрозуміло, яка ділянка коду формує тіло класу Program (рядки 3-6), а яка – тіло метода Main (рядок 5).

При введенні тексту програми редактор коду Microsoft Visual Studio переважно автоматично вирівнює код, виконуючи основну роботу по форматуванню замість програміста.

1. Лістинг містить стовпчик з номерами рядків для зручності його опису. Номери рядків не є складовою частиною програми. [↑](#footnote-ref-1)
2. Їх ще називають угодами іменування (naming conventions). [↑](#footnote-ref-2)